N. Sakimura
NAT.Consulting
J. Bradley
Yubico
M. Jones
Microsoft
B. de Medeiros
Google
C. Mortimore
VISA (was at Salesforce)
July 24, 2020

OpenID Connect Basic Client Implementer's Guide 1.0 - draft 40

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 OpenID Connect Basic Client Implementer's Guide 1.0 contains a subset of the OpenID Connect Core 1.0 specification that is designed to be easy to read and implement for basic Web-based Relying Parties using the OAuth Authorization Code Flow. This document intentionally duplicates content from the Core specification to provide a self-contained implementer's guide for basic Web-based Relying Parties using the OAuth Authorization Code Flow.

OpenID Providers and non-Web-based applications should instead consult the Core specification.

Table of Contents

1. Introduction

This OpenID Connect Basic Client Implementer's Guide 1.0 contains a subset of the **OpenID Connect Core 1.0** [OpenID.Core] specification that is designed to be easy to read and implement for basic Web-based Relying Parties using the OAuth 2.0 **[RFC6749]** Authorization Code Flow. This document intentionally duplicates content from the Core specification to provide a self-contained implementer's guide for basic Web-based Relying Parties using the OAuth Authorization Code Flow. Should there be any conflicts between the contents of this implementer's guide and the OpenID Connect Core specification, the latter takes precedence.

See the **OpenID Connect Implicit Client Implementer's Guide 1.0** [OpenID.Implicit] for a related guide for basic Web-based Relying Parties using the OAuth Implicit Flow. OpenID Providers and non-Web-based applications should instead consult the Core specification. This guide omits implementation and security considerations for OpenID Providers and non-Web-based applications.

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. An Authorization Request using these extensions is called an Authentication Request.

Information about the authentication performed is returned in a **JSON Web Token (JWT)** [JWT] called an ID Token (see **Section 2.2**). 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 document 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 document 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.

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 **[RFC2119]**.

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

All uses of **JSON Web Signature (JWS)** [JWS] data structures in this document utilize the JWS Compact Serialization; the JWS JSON Serialization is not used.

When the RFC 2119 language applies to the behavior of OpenID Providers, it is in this document for explanatory value to help Client implementers understand the expected behavior of OpenID Providers.

1.2. Terminology

TOC

This document 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 Owner", "Resource Server", "Response Type", and "Token Endpoint" defined by **OAuth 2.0** [RFC6749], the terms "Claim Name", "Claim Value", "JSON Web Token (JWT)", and "JWT Claims Set" defined by **JSON Web Token (JWT)** [JWT], the terms "Header Parameter" and "JOSE Header" defined by **JSON Web Signature (JWS)** [JWS], and the term "User Agent" defined by **RFC 7230** [RFC7230].

This document 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.

Claim
: Piece of information asserted about an Entity.

Claims Provider
: Server that can return Claims about an Entity.

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.

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.
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.
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.
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.
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.
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 document, imposing requirements upon implementations. All the capitalized words in the text of this document, such as "Issuer Identifier", reference these defined terms. Whenever the reader encounters them, their definitions found in this section must be followed.

TOC

1.3. Overview

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:

```
+---------+                                   +---------+
|         |                                   |         |
|         |---------(1) AuthN Request-------->|         |
|         |                                   |         |
|         |  +---------+                      |         |
|         |  |         |                      |         |
|         |  |  End-   |<--(2) AuthN & AuthZ-->|         |
|         |  |  User   |                      |         |
|   RP    |  |         |                        |   OP    |
|         |  +---------+                        |         |
|         |                                     |         |
|         |<--------(3) AuthN Response--------|         |
|         |                                     |         |
|         |---------(4) UserInfo Request----->|         |
|         |                                     |         |
|         |<--------(5) UserInfo Response-----|         |
|         |                                     |         |
+---------+                                     +---------+
```

2. Protocol Elements

TOC

Authentication Requests can follow one of three paths: the Authorization Code Flow, the Implicit Flow, or the Hybrid Flow. The Authorization Code Flow is intended for Clients that can securely maintain a Client Secret between themselves and the Authorization Server, whereas the Implicit Flow is intended for Clients that cannot. However, the Authorization Code flow is sometimes also used by Native applications and other Clients in order to be able to obtain a Refresh Token, even when they cannot ensure the secrecy of the Client Secret value. The Hybrid Flow combines aspects of the Authorization Code Flow and the Implicit Flow. It enables Clients to obtain an ID Token and optionally an Access Token with only one round trip to the Authorization Server, possibly minimizing latency, while still enabling Clients to later get tokens from the Token Endpoint -- especially a Refresh Token.

This document only provides information that is sufficient for basic Clients using the Code Flow.

2.1. Code Flow

TOC

The Code Flow consists of 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 `code`.
6. Client sends the `code` to the Token Endpoint to receive an Access Token and ID Token in the response.
7. Client validates the tokens and retrieves the End-User's Subject Identifier.

2.1.1. Client Prepares Authentication Request

TOC

When the RP wishes to Authenticate the End-User or determine whether the End-User is already Authenticated, the Client prepares an Authentication Request to be sent to the Authorization Endpoint.

Communication with the Authorization Endpoint MUST utilize TLS. See **Section 7.1** for more information on using TLS.

Clients MAY construct the request using the HTTP `GET` or the HTTP `POST` method as defined in **RFC 7231** [RFC7231].

If using the HTTP `GET` method, the parameters are serialized using the Query String Serialization, per **Section 3.1**. If using the HTTP `POST` method, the request parameters are added to the HTTP request entity-body using the `application/x-www-form-urlencoded` format as defined by **[W3C.REC-html401-19991224]**.

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

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

TOC

2.1.1.1. Request Parameters

This subset of OpenID Connect uses the following OAuth 2.0 request parameters:

response_type
: REQUIRED. This value MUST be `code`. This requests that both an Access Token and an ID Token be returned from the Token Endpoint in exchange for the `code` value returned from the Authorization Endpoint.

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

scope
: REQUIRED. OpenID Connect requests MUST contain the `openid` scope value. OPTIONAL scope values of `profile`, `email`, `address`, `phone`, and `offline_access` are also defined. See **Section 2.4** for more about the scope values defined by this document.

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). 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. 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.

This document 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. One method to achieve this is to store a cryptographically random value as an HttpOnly a session cookie and use a cryptographic hash of the value as the `nonce` parameter. In that case, the `nonce` in the returned ID Token is compared to the hash of the session cookie to detect ID Token replay by third parties. Use of the nonce is OPTIONAL when using the code flow.

display
: OPTIONAL. ASCII **[RFC20]** 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.

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`. 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.

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. 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.

claims_locales
: OPTIONAL. End-User's preferred languages and scripts for Claims being returned, represented as a space-separated list of **BCP47** [RFC5646] language tag values, ordered by preference. 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 logged in or is logged in by the request, then the Authorization Server returns a positive response; otherwise, it SHOULD return an error. 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.

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

TOC

2.1.2. Client Sends Request to Authorization Server

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

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
  &client_id=s6BhdRkqt3
  &redirect_uri=https%3A%2F%2Fclient.example.org%2Fcb
  &scope=openid%20profile
  &state=af0ifjsldkj
```

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

2.1.3. Authorization Server Authenticates End-User

The Authorization Server logs in the End-User or verifies whether the End-User is logged in, depending upon the request parameter values used. If interaction with the End-User occurs over an HTTP channel, it MUST use TLS, as per **Section 7.1**. The exact authentication methods used are out of scope for this document.

2.1.4. Authorization Server Obtains End-User Consent/Authorization

The Authorization Server obtains an authorization decision for the requested Claims. This can be done by presenting the End-User with a dialogue that enables the End-User to recognize what is being consenting to and grant consent or by establishing consent via other means (for example, via previous administrative consent).

The `openid` scope value declares that this OAuth 2.0 request is an OpenID Connect request. Use of all other scope values is OPTIONAL.

2.1.5. Authorization Server Sends End-User Back to Client

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

2.1.5.1. End-User Grants Authorization

If the End-User grants the access request, the Authorization Server issues a `code` and delivers it to the Client by adding the following query parameters to the query component of the Redirection URI using the `application/x-www-form-urlencoded` format as defined in Section 4.1.2 of **OAuth 2.0** [RFC6749].

code
: REQUIRED. OAuth 2.0 Authorization Code.

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.

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

```
HTTP/1.1 302 Found
Location: https://client.example.org/cb?
  code=Splxl0BeZQQYbYS6WxSbIA
  &state=af0ifjsldkj
```

2.1.5.2. End-User Denies Authorization or Invalid Request

If the End-User denies the authorization or the End-User authentication fails, the Authorization Server MUST return the error Authorization Response as defined in 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.)

2.1.6. Client Obtains ID Token and Access Token

The Client then makes an Access Token Request using the Authorization Code to obtain tokens from the Token Endpoint in the following manner:

2.1.6.1. Client Sends Code

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]. The Client MUST authenticate to the Token Endpoint using the HTTP Basic method, as described in 2.3.1 of OAuth 2.0. (This method is the one identified by using the `client_secret_basic` authentication method value in **OpenID Connect Discovery 1.0** [OpenID.Discovery]).

The Client sends the parameters to the Token Endpoint using the HTTP `POST` method and the Form Serialization, per **Section 3.2**, as described in Section 4.1.3 of **OAuth 2.0** [RFC6749].

Communication with the Token Endpoint MUST utilize TLS. See **Section 7.1** for more information on using TLS.

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

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

grant_type=authorization_code&code=SplxlOBeZQQYbYS6WxSbIA
  &redirect_uri=https%3A%2F%2Fclient%2Eexample%2Ecom%2Fcb
```

2.1.6.2. Client Receives Tokens

The Client receives a response with the following parameters as described in Section 4.1.4 of **OAuth 2.0** [RFC6749]. The response SHOULD be encoded using UTF-8 **[RFC3629]**.

access_token
: REQUIRED. Access Token for the UserInfo Endpoint.

token_type
: REQUIRED. OAuth 2.0 Token Type value. The value MUST be `Bearer`, as specified in **OAuth 2.0 Bearer Token Usage** [RFC6750], for Clients using this subset. Note that the `token_type` value is case insensitive.

id_token
: REQUIRED. ID Token.

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

refresh_token
: OPTIONAL. Refresh Token.

The Client can then use the Access Token to access protected resources at Resource Servers.

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

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

```
Pragma: no-cache

{
 "access_token":"SlAV32hkKG",
 "token_type":"Bearer",
 "expires_in":3600,
 "refresh_token":"tGzv3JOkF0XG5Qx2TlKWIA",
 "id_token":"eyJ0 ... NiJ9.eyJ1c ... I6IjIifX0.DeWt4Qu ... ZXso"
}
```

2.2. ID Token

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:

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

aud
: 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 for processing. The processing of this parameter requires that the current date/time MUST be before the expiration date/time listed in the value. Implementers MAY provide for some small leeway, usually no more than a few minutes, to account for clock skew. Its value is a JSON **[RFC7159]** 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.

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 then this Claim is REQUIRED; otherwise, its inclusion is OPTIONAL.

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. The Client 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. The `nonce` value is a case-sensitive string.

at_hash
: OPTIONAL. Access Token hash value. This is OPTIONAL when the ID Token is issued from the Token Endpoint, which is the case for this subset of OpenID

Connect; nonetheless, an `at_hash` Claim MAY be present. 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.

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. 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 definition of particular values to be used in the `amr` Claim is beyond the scope of this document. Parties using this claim will need to agree upon the meanings of the values used, which may be context specific. The `amr` value is an array of case-sensitive strings.

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. This Claim is only needed when the ID Token has a single audience value and that audience is different than the authorized party. It MAY be included even when the authorized party is the same as the sole audience. The `azp` value is a case-sensitive string containing a StringOrURI value.

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

ID Tokens SHOULD NOT use the JWS or JWE `x5u`, `x5c`, `jku`, or `jwk` Header Parameter fields. Instead, keys used for ID Tokens are communicated in advance using Discovery and Registration parameters.

The following is a non-normative example of the set of Claims (the JWT Claims Set) base64url-decoded from an ID Token:

```
{
 "iss": "https://server.example.com",
 "sub": "24400320",
 "aud": "s6BhdRkqt3",
 "exp": 1311281970,
 "iat": 1311280970
}
```

TOC

2.2.1. ID Token Validation

If any of the validation procedures defined in this document 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.

The Client MUST validate the ID Token in the Token Response. To do this, the Client can split the ID Token at the period (".") characters, take the second segment, and base64url decode it to obtain a JSON object containing the ID Token Claims, which MUST be validated as follows:

1. The Issuer Identifier for the OpenID Provider (which is typically obtained during Discovery) MUST exactly match the value of the `iss` (issuer) Claim.
2. 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 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.
3. If the ID Token contains multiple audiences, the Client SHOULD verify that an `azp` Claim is present.
4. If an `azp` (authorized party) Claim is present, the Client SHOULD verify that its `client_id` is the Claim Value.
5. The current time MUST be before the time represented by the `exp` Claim (possibly allowing for some small leeway to account for clock skew).
6. 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.
7. 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 document.
8. When a `max_age` request is made, 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.

2.3. UserInfo Endpoint

The UserInfo Endpoint is an OAuth 2.0 Protected Resource that returns Claims about the authenticated End-User. The location of the UserInfo Endpoint MUST be a URL using the `https` scheme, which MAY contain port, path, and query parameter components. The returned Claims are represented by a JSON object that contains a collection of name and value pairs for the Claims.

Communication with the UserInfo Endpoint MUST utilize TLS. See **Section 7.1** for more information on using TLS.

2.3.1. UserInfo Request

Clients send requests to the UserInfo Endpoint to obtain Claims about the End-User using an Access Token obtained through OpenID Connect Authentication. The UserInfo Endpoint is an **OAuth 2.0** [RFC6749] Protected Resource that complies with the **OAuth 2.0 Bearer Token Usage** [RFC6750] specification. The request SHOULD use the HTTP `GET` method and the Access Token SHOULD be sent using the `Authorization` header field.

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

```
GET /userinfo HTTP/1.1
Host: server.example.com
Authorization: Bearer SlAV32hkKG
```

2.3.2. Successful UserInfo Response

The UserInfo Claims MUST be returned as the members of a JSON object. The response body SHOULD be encoded using UTF-8. The Claims defined in **Section 2.5** can be returned, as can additional Claims not specified there.

If a Claim is not returned, that Claim Name SHOULD be omitted from the JSON object representing the Claims; it SHOULD NOT be present with a null or empty string value.

The `sub` (subject) Claim MUST always be returned in the UserInfo Response.

NOTE: Due to the possibility of token substitution attacks, the UserInfo Response is not guaranteed to be about the End-User identified by the `sub` (subject) element of the ID Token. The `sub` Claim in the UserInfo Response MUST be verified to exactly match the `sub` Claim in the ID Token; if they do not match, the UserInfo Response values MUST NOT be used.

The Client MUST verify that the OP that responded was the intended OP through a TLS server certificate check, per **RFC 6125** [RFC6125].

2.3.3. UserInfo Error Response

When an error condition occurs, the UserInfo Endpoint returns an Error Response as defined in Section 3 of **OAuth 2.0 Bearer Token Usage** [RFC6750].

2.4. Scope Values

OpenID Connect Clients use `scope` values as defined in 3.3 of **OAuth 2.0** [RFC6749] to specify what access privileges are being requested for Access Tokens. The scopes associated with Access Tokens determine what resources will be available when they are used to access OAuth 2.0 protected endpoints. For OpenID Connect, scopes can be used to request that specific sets of information be made available as Claim Values. This document describes only the scope values used by OpenID Connect.

OpenID Connect allows additional scope values to be defined and used. Scope values used that are not understood by an implementation SHOULD be ignored.

Claims requested by the following scopes are treated by Authorization Servers as Voluntary Claims.

OpenID Connect defines the following `scope` values:

openid
: REQUIRED. Informs the Authorization Server that the Client is making an OpenID Connect request. If the `openid` scope value is not present, the behavior is entirely unspecified.

profile
: OPTIONAL. This scope value requests access to the End-User's default profile Claims, which are: `name`, `family_name`, `given_name`, `middle_name`, `nickname`, `preferred_username`, `profile`, `picture`, `website`, `gender`, `birthdate`, `zoneinfo`, `locale`, and `updated_at`.

email
: OPTIONAL. This scope value requests access to the `email` and `email_verified` Claims.

address
: OPTIONAL. This scope value requests access to the `address` Claim.

phone
: OPTIONAL. This scope value requests access to the `phone_number` and `phone_number_verified` Claims.

offline_access
: OPTIONAL. This scope value requests that an OAuth 2.0 Refresh Token be issued that can be used to obtain an Access Token that grants access to the End-User's UserInfo Endpoint even when the End-User is not present (not logged in).

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

The Claims requested by the `profile`, `email`, `address`, and `phone` scope values are returned from the UserInfo Endpoint, as described in **Section 2.3.2**.

In some cases, the End-User will be given the option to have the OpenID Provider decline to provide some or all information requested by RPs. To minimize the amount of information that the End-User is being asked to disclose, an RP can elect to only request a subset of the information available from the UserInfo Endpoint.

The following is a non-normative example of a `scope` Request:

```
scope=openid profile email phone
```

TOC

2.5. Standard Claims

This subset of OpenID Connect defines a set of standard Claims. They are returned in the UserInfo Response.

Member	Type	Description
sub	string	Subject - Identifier for the End-User at the Issuer.
name	string	End-User's full name in displayable form including all name parts, possibly including titles and suffixes, ordered according to the End-User's locale and preferences.
given_name	string	Given name(s) or first name(s) of the End-User. Note that in some cultures, people can have multiple given names; all can be present, with the names being separated by space characters.
family_name	string	Surname(s) or last name(s) of the End-User. Note that in some cultures, people can have multiple family names or no family name; all can be present, with the names being separated by space characters.
middle_name	string	Middle name(s) of the End-User. Note that in some cultures, people can have multiple middle names; all can be present, with the names being separated by space characters. Also note that in some cultures, middle names are not used.
nickname	string	Casual name of the End-User that may or may not be the same as the `given_name`. For instance, a `nickname` value of `Mike` might be returned alongside a `given_name` value of `Michael`.
preferred_username	string	Shorthand name by which the End-User wishes to be referred to at the RP, such as `janedoe` or `j.doe`. This value MAY be any valid JSON string including special characters such as `@`, `/`, or whitespace. The RP MUST NOT rely upon this value being unique, as discussed in **Section 2.5.3**.
profile	string	URL of the End-User's profile page. The contents of this Web page SHOULD be about the End-User.
picture	string	URL of the End-User's profile picture. This URL MUST refer to an image file (for example, a PNG, JPEG, or GIF image file), rather than to a Web page containing an image. Note that this URL SHOULD specifically reference a profile photo of the End-User suitable for displaying when describing the End-User, rather than an arbitrary photo taken by the End-User.
website	string	URL of the End-User's Web page or blog. This Web page SHOULD contain information published by the End-User or an organization that the End-User is affiliated with.
email	string	End-User's preferred e-mail address. Its value MUST conform to the **RFC 5322** [RFC5322] addr-spec syntax. The RP MUST NOT rely upon this value being unique, as discussed in **Section 2.5.3**.
email_verified	boolean	True if the End-User's e-mail address has been verified; otherwise

		false. When this Claim Value is `true`, this means that the OP took affirmative steps to ensure that this e-mail address was controlled by the End-User at the time the verification was performed. The means by which an e-mail address is verified is context specific, and dependent upon the trust framework or contractual agreements within which the parties are operating.
gender	string	End-User's gender. Values defined by this document are `female` and `male`. Other values MAY be used when neither of the defined values are applicable.
birthdate	string	End-User's birthday, represented as an **ISO 8601:2004** [ISO8601-2004] `YYYY-MM-DD` format. The year MAY be `0000`, indicating that it is omitted. To represent only the year, `YYYY` format is allowed. Note that depending on the underlying platform's date related function, providing just year can result in varying month and day, so the implementers need to take this factor into account to correctly process the dates.
zoneinfo	string	String from zoneinfo **[zoneinfo]** time zone database representing the End-User's time zone. For example, `Europe/Paris` or `America/Los_Angeles`.
locale	string	End-User's locale, represented as a **BCP47** [RFC5646] language tag. This is typically an **ISO 639-1 Alpha-2** [ISO639-1] language code in lowercase and an **ISO 3166-1 Alpha-2** [ISO3166-1] country code in uppercase, separated by a dash. For example, `en-US` or `fr-CA`. As a compatibility note, some implementations have used an underscore as the separator rather than a dash, for example, `en_US`; Relying Parties MAY choose to accept this locale syntax as well.
phone_number	string	End-User's preferred telephone number. **E.164** [E.164] is RECOMMENDED as the format of this Claim, for example, `+1 (425) 555-1212` or `+56 (2) 687 2400`. If the phone number contains an extension, it is RECOMMENDED that the extension be represented using the **RFC 3966** [RFC3966] extension syntax, for example, `+1 (604) 555-1234;ext=5678`.
phone_number_verified	boolean	True if the End-User's phone number has been verified; otherwise false. When this Claim Value is `true`, this means that the OP took affirmative steps to ensure that this phone number was controlled by the End-User at the time the verification was performed. The means by which a phone number is verified is context specific, and dependent upon the trust framework or contractual agreements within which the parties are operating. When true, the `phone_number` Claim MUST be in E.164 format and any extensions MUST be represented in RFC 3966 format.
address	JSON object	End-User's preferred postal address. The value of the `address` member is a JSON **[RFC4627]** structure containing some or all of the members defined in **Section 2.5.1**.
updated_at	number	Time the End-User's information was last updated. 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.

Table 1: Reserved Member Definitions

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

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

The UserInfo Endpoint MUST return Claims in JSON format unless a different format was specified during Registration **[OpenID.Registration]**. The UserInfo Endpoint MUST return a content-type header to indicate which format is being returned. The following are accepted content types:

Content-Type	Format Returned
application/json	plain text JSON object
application/jwt	JSON Web Token (JWT)

2.5.1. Address Claim

The Address Claim represents a physical mailing address. Implementations MAY return only a subset of the fields of an `address`, depending upon the information available and the End-User's privacy preferences. For example, the `country` and `region` might be returned without returning more fine-grained address information.

Implementations MAY return just the full address as a single string in the formatted sub-field, or they MAY return just the individual component fields using the other sub-fields, or they MAY return both. If both variants are returned, they SHOULD be describing the same address, with the formatted address indicating how the component fields are combined.

formatted
: Full mailing address, formatted for display or use on a mailing label. This field MAY contain multiple lines, separated by newlines. Newlines can be represented either as a carriage return/line feed pair ("\r\n") or as a single line feed character ("\n").

street_address
: Full street address component, which MAY include house number, street name, Post Office Box, and multi-line extended street address information. This field MAY contain multiple lines, separated by newlines. Newlines can be represented either as a carriage return/line feed pair ("\r\n") or as a single line feed character ("\n").

locality
: City or locality component.

region
: State, province, prefecture, or region component.

postal_code
: Zip code or postal code component.

country
: Country name component.

2.5.2. Claims Languages and Scripts

Human-readable Claim Values and Claim Values that reference human-readable values MAY be represented in multiple languages and scripts. To specify the languages and scripts, **BCP47** [RFC5646] language tags are added to member names, delimited by a `#` character. For example, `family_name#ja-Kana-JP` expresses the Family Name in Katakana in Japanese, which is commonly used to index and represent the phonetics of the Kanji representation of the same represented as `family_name#ja-Hani-JP`. As another example, both `website` and `website#de` Claim Values might be returned, referencing a Web site in an unspecified language and a Web site in German.

Since Claim Names are case sensitive, it is strongly RECOMMENDED that language tag values used in Claim Names be spelled using the character case with which they are registered in the IANA "Language Subtag Registry" **[IANA.Language]**. In particular, normally language names are spelled with lowercase characters, region names are spelled with uppercase characters, and scripts are spelled with mixed case characters. However, since BCP47 language tag values are case insensitive, implementations SHOULD interpret the language tag values supplied in a case-insensitive manner.

Per the recommendations in BCP47, language tag values for Claims SHOULD only be as specific as necessary. For instance, using `fr` might be sufficient in many contexts, rather than `fr-CA` or `fr-FR`. Where possible, OPs SHOULD try to match requested Claim locales with Claims it has. For instance, if the Client asks for a Claim with a `de` (German) language tag and the OP has a value tagged with `de-CH` (Swiss German) and no generic German value, it would be appropriate for the OP to return the Swiss German value to the Client. (This intentionally moves as much of the complexity of language tag matching to the OP as possible, to simplify Clients.)

A `claims_locales` request can be used to specify the preferred languages and scripts to use for the returned Claims.

When the OP determines, either through the `claims_locales` parameter, or by other means, that the End-User and Client are requesting Claims in only one set of languages and scripts, it is RECOMMENDED that OPs return Claims without language tags when they employ this language and script. It is also RECOMMENDED that Clients be written in a manner that they can handle and utilize Claims using language tags.

2.5.3. Claim Stability and Uniqueness

The `sub` (subject) and `iss` (issuer) Claims, used together, are the only Claims that an RP can rely upon as a stable identifier for the End-User, since the `sub` Claim MUST be locally unique and never reassigned within the Issuer for a particular End-User, as described in **Section 2.2**. Therefore, the only guaranteed unique identifier for a given End-User is the combination of the `iss` Claim and the `sub` Claim.

All other Claims carry no such guarantees across different issuers in terms of stability over time or uniqueness across users, and Issuers are permitted to apply local restrictions and policies. For instance, an Issuer MAY re-use an `email` Claim Value across different End-Users at different points in time, and the claimed `email` address for a given End-User MAY change over time. Therefore, other Claims such as `email`, `phone_number`, and `preferred_username` and MUST NOT be used as unique identifiers for the End-User.

3. Serializations

A request message MAY be serialized using one of the following methods:

1. Query String Serialization
2. Form Serialization

3.1. 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 using the `application/x-www-form-urlencoded` format as defined by **[W3C.REC-html401-19991224]**. Query String Serialization is typically used in HTTP `GET` requests.

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

```
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
```

3.2. Form Serialization

TOC

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]**. Form Serialization is typically used in HTTP `POST` requests.

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

```
    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
```

4. String Operations

TOC

Processing some OpenID Connect messages requires comparing values in the messages to known values. For example, the Claim Names returned by the UserInfo Endpoint might be compared to specific Claim Names such as `sub`. Comparing Unicode **[UNICODE]** strings, however, has significant security implications.

Therefore, comparisons between JSON strings and other Unicode strings MUST be performed as specified below:

1. Remove any JSON applied escaping to produce an array of Unicode code points.
2. Unicode Normalization **[USA15]** MUST NOT be applied at any point to either the JSON string or to the string it is to be compared against.
3. Comparisons between the two strings MUST be performed as a Unicode code point to code point equality comparison.

In several places, this document uses space-delimited lists of strings. In all such cases, the ASCII space character (0x20) MUST be the only character used for this purpose.

5. TLS Version

TOC

Whenever Transport Layer Security (TLS) is used by this document, the appropriate version (or versions) of TLS will vary over time, based on the widespread deployment and known security vulnerabilities. At the time of this writing, TLS version 1.2 **[RFC5246]** is the most recent version, but has a very limited deployment base and might not be readily available for implementation. TLS version 1.0 **[RFC2246]** is the most widely deployed version and will provide the broadest interoperability.

6. Implementation Considerations

TOC

This document defines features used by Relying Parties using the OAuth Authorization Code Flow. These Relying Parties MUST implement the features that are listed in this document as being "REQUIRED" or are described with a "MUST".

6.1. 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** [OpenID.Discovery] and **OpenID Connect Dynamic Client Registration 1.0** [OpenID.Registration] specifications.

7. Security Considerations

For security considerations other than those listed below, refer to the **OpenID Connect Core 1.0** [OpenID.Core] specification.

7.1. 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]** 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]** 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** [RFC6125].

8. Privacy Considerations

8.1. Personally Identifiable Information

The UserInfo Response typically contains Personally Identifiable Information (PII). 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_uris`.

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

8.2. Data Access Monitoring

The Resource Server SHOULD make End-Users' UserInfo access logs available to them so that they can monitor who accessed their data.

8.3. Correlation

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.

8.4. Offline Access

Offline access enables access to Claims when the user is not present, posing greater privacy risk than the Claims transfer when the user is present. Therefore, it is prudent to obtain explicit consent for offline access to resources. This document mandates the use of the `prompt` parameter to obtain consent unless it is already known that the request complies with the conditions for processing the request in each jurisdiction.

When an Access Token is returned in the front channel, there is a greater risk of it being exposed to an attacker, who could later use it to access the UserInfo endpoint. If the Access Token does not enable offline access and the server can differentiate whether the Client request has been made offline or online, the risk will be substantially reduced. Therefore, this document mandates ignoring the offline access request when the Access Token is transmitted in the front channel. Note that differentiating between online and offline access from the server can be difficult especially for native clients. The server may well have to rely on heuristics. Also, the risk of exposure for the Access Token delivered in the front channel for the Response Types of `code token` and `token` is the same. Thus, the implementations should be prepared to detect the channel from which the Access Token was issued and deny offline access if the token was issued in the front channel.

Note that although these provisions require an explicit consent dialogue through the `prompt` parameter, the mere fact that the user pressed an "accept" button etc., might not constitute a valid consent. Developers should be aware that for the act of consent to be valid, typically, the impact of the terms have to be understood by the End-User, the consent must be freely given and not forced (i.e., other options have to be available), and the terms must fair and equitable. In general, it is advisable for the service to follow the required privacy principles in each jurisdiction and rely on other conditions for processing the request than simply explicit consent, as online self-service "explicit consent" often does not form a valid consent in some jurisdictions.

9. IANA Considerations

This document makes no requests of IANA.

10. References

10.1. Normative References

[E.164]	International Telecommunication Union, “**E.164: The international public telecommunication numbering plan**,” 2010.
[IANA.Language]	IANA, “**Language Subtag Registry**.”
[ISO29115]	International Organization for Standardization, “**ISO/IEC 29115:2013 -- Information technology - Security techniques - Entity authentication assurance framework**,” ISO/IEC 29115, March 2013.
[ISO3166-1]	International Organization for Standardization, “**ISO 3166-1:1997. Codes for the representation of names of countries and their subdivisions -- Part 1: Country codes**,” 1997.
[ISO639-1]	International Organization for Standardization, “**ISO 639-1:2002. Codes for the representation of names of languages -- Part 1: Alpha-2 code**,” 2002.
[ISO8601-2004]	International Organization for Standardization, “**ISO 8601:2004. Data elements and interchange formats - Information interchange - Representation of dates and times**,” 2004.
[JWS]	Jones, M., Bradley, J., and N. Sakimura, “**JSON Web Signature (JWS)**,” RFC 7515, DOI 10.17487/RFC7515, May 2015.

[JWT]	Jones, M., Bradley, J., and N. Sakimura, "**JSON Web Token (JWT)**," RFC 7519, DOI 10.17487/RFC7519, May 2015.
[OpenID.Core]	Sakimura, N., Bradley, J., Jones, M., de Medeiros, B., and C. Mortimore, "**OpenID Connect Core 1.0**," November 2014.
[OpenID.Discovery]	Sakimura, N., Bradley, J., Jones, M., and E. Jay, "**OpenID Connect Discovery 1.0**," November 2014.
[OpenID.Registration]	Sakimura, N., Bradley, J., and M. Jones, "**OpenID Connect Dynamic Client Registration 1.0**," November 2014.
[RFC20]	Cerf, V., "**ASCII format for Network Interchange**," STD 80, RFC 20, DOI 10.17487/RFC0020, October 1969.
[RFC2119]	Bradner, S., "**Key words for use in RFCs to Indicate Requirement Levels**," BCP 14, RFC 2119, DOI 10.17487/RFC2119, March 1997.
[RFC2246]	Dierks, T. and C. Allen, "**The TLS Protocol Version 1.0**," RFC 2246, DOI 10.17487/RFC2246, January 1999.
[RFC3339]	Klyne, G. and C. Newman, "**Date and Time on the Internet: Timestamps**," RFC 3339, DOI 10.17487/RFC3339, July 2002.
[RFC3629]	Yergeau, F., "**UTF-8, a transformation format of ISO 10646**," STD 63, RFC 3629, DOI 10.17487/RFC3629, November 2003.
[RFC3966]	Schulzrinne, H., "**The tel URI for Telephone Numbers**," RFC 3966, DOI 10.17487/RFC3966, December 2004.
[RFC3986]	Berners-Lee, T., Fielding, R., and L. Masinter, "**Uniform Resource Identifier (URI): Generic Syntax**," STD 66, RFC 3986, DOI 10.17487/RFC3986, January 2005.
[RFC4627]	Crockford, D., "**The application/json Media Type for JavaScript Object Notation (JSON)**," RFC 4627, DOI 10.17487/RFC4627, July 2006.
[RFC5246]	Dierks, T. and E. Rescorla, "**The Transport Layer Security (TLS) Protocol Version 1.2**," RFC 5246, DOI 10.17487/RFC5246, August 2008.
[RFC5322]	Resnick, P., Ed., "**Internet Message Format**," RFC 5322, DOI 10.17487/RFC5322, October 2008.
[RFC5646]	Phillips, A., Ed. and M. Davis, Ed., "**Tags for Identifying Languages**," BCP 47, RFC 5646, DOI 10.17487/RFC5646, September 2009.
[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)**," RFC 6125, DOI 10.17487/RFC6125, March 2011.
[RFC6711]	Johansson, L., "**An IANA Registry for Level of Assurance (LoA) Profiles**," RFC 6711, DOI 10.17487/RFC6711, August 2012.
[RFC6749]	Hardt, D., Ed., "**The OAuth 2.0 Authorization Framework**," RFC 6749, DOI 10.17487/RFC6749, October 2012.
[RFC6750]	Jones, M. and D. Hardt, "**The OAuth 2.0 Authorization Framework: Bearer Token Usage**," RFC 6750, DOI 10.17487/RFC6750, October 2012.
[RFC7159]	Bray, T., Ed., "**The JavaScript Object Notation (JSON) Data Interchange Format**," RFC 7159, DOI 10.17487/RFC7159, March 2014.
[RFC7230]	Fielding, R., Ed. and J. Reschke, Ed., "**Hypertext Transfer Protocol (HTTP/1.1): Message Syntax and Routing**," RFC 7230, DOI 10.17487/RFC7230, June 2014.
[RFC7231]	Fielding, R., Ed. and J. Reschke, Ed., "**Hypertext Transfer Protocol (HTTP/1.1): Semantics and Content**," RFC 7231, DOI 10.17487/RFC7231, June 2014.
[UNICODE]	The Unicode Consortium, "**The Unicode Standard**."
[USA15]	**Davis, M.** and **K. Whistler**, "**Unicode Normalization Forms**," Unicode Standard Annex 15, 06 2015.
[W3C.REC-html401-19991224]	Raggett, D., Hors, A., and I. Jacobs, "**HTML 4.01 Specification**," World Wide Web Consortium Recommendation REC-html401-19991224, December 1999 (**HTML**).
[zoneinfo]	Public Domain, "**The tz database**," June 2011.

10.2. Informative References

TOC

[OpenID.Implicit]	Sakimura, N., Bradley, J., Jones, M., de Medeiros, B., and C. Mortimore, "**OpenID Connect Implicit Client Implementer's Guide 1.0**," July 2020.

Appendix A. Acknowledgements

TOC

The OpenID Community would like to thank the following people for their contributions to this document:

Naveen Agarwal (Naveen.Agarwal@microsoft.com), Microsoft

Casper Biering (cb@peercraft.com), Peercraft

John Bradley (ve7jtb@ve7jtb.com), Yubico

Tim Bray (tbray@textuality.com), Independent

Johnny Bufu (johnny.bufu@gmail.com), Independent

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

Pamela Dingle (Pamela.Dingle@microsoft.com), Microsoft

George Fletcher (george.fletcher@corp.aol.com), AOL

Roland Hedberg (roland@catalogix.se), Independent

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

Edmund Jay (ejay@mgi1.com), Illumila

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

Torsten Lodderstedt (torsten@lodderstedt.net), yes.com

Nov Matake (nov@matake.jp), Independent

Chuck Mortimore (charliemortimore@gmail.com), VISA

Anthony Nadalin (tonynad@microsoft.com), Microsoft

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

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

David Recordon (recordond@gmail.com), Independent

Justin Richer (justin@bspk.io), Bespoke Engineering

Nat Sakimura (nat@nat.consulting), NAT.Consulting

Luke Shepard (luke@lukeshepard.com), Facebook

Andreas Åkre Solberg (andreas.solberg@uninett.no), UNINET

Paul Tarjan (paul@paultarjan.com), Facebook

TOC

Appendix B. Notices

proprietary rights that may cover technology that may be required to practice this specification.

Appendix C. Document History

[[To be removed from the final document]]

-40

- Updated affiliations and acknowledgements.

-39

- Tracked corresponding errata changes to Core draft -25.

-38

- Updated copyright.

-37

- Referenced completed RFCs.
- Added missing URLs in references.
- Changed to use "Cache-Control: no-cache, no-store" and "Pragma: no-cache" in examples.
- Tracked terminology changes made in the referenced IETF specs since errata set 1.
- Updated the RFC 2616 references to RFC 7230 or RFC 7231, as appropriate.

-36

- Referenced specification versions incorporating errata set 1.

-35

- Updated dates for specs containing errata updates.
- Updated references to pre-final IETF specs.
- Replaced uses of the terms JWS Header, JWE Header, and JWT Header with the JOSE Header term that replaced them in the JOSE and JWT specifications.
- Fixed #954 - Added "NOT RECOMMENDED" to the list of RFC 2119 terms.

-34

- Fixed #918 - Wording inconsistency in Token Request language.
- Changed uses of "this specification" to "this document".

-33

- Updated dates for final OpenID Connect specifications.

-32

- Editorial corrections.

-31

- Fixed #896 - Replaced the term Authorization Request with Authentication Request, where applicable.
- Incorporated terms defined by the JWT specification.
- Applied proofreading corrections by Michael B. Jones.

-30

- Updated the `response_type` language.
- Fixed #878 - Generalized description of errors that can be returned when `id_token_hint`" is used.
- Provided more context in the introduction.

- Expanded the Authentication Request example to show both the 302 redirect response by the Client and the resulting HTTP GET request sent by the User Agent.

-29

- Tracked editorial changes applied to OpenID Connect Core.
- Fixed #862 - Clarified `azp` definition.
- Fixed #878 - Defined negative response for "id_token_hint".
- Replaced uses of the OpenID Connect Messages and OpenID Connect Standard specifications with OpenID Connect Core.
- Fixed #884 - Changed the descriptions of Basic and Implicit from being profiles to being implementer's guides containing subsets of OpenID Connect Core.

-28

- Fixed #847 - Corrected type of `updated_at` to number.
- Stated that `redirect_uri` matches must be exact, with matching performed as described in Section 6.2.1 of RFC 3986 (Simple String Comparison).
- Fixed #854 - Clarified that the `acr_values` values are in order of preference and that `acr_values` requests the `acr` Claim as a Voluntary Claim.
- Fixed #858 - Incorporated elements of the Issuer Identifier definition into the `iss` Claim description.
- Fixed #859 - Added IMPORTANT NOTE TO READERS about the terminology definitions being a normative part of the specification.

-27

- Fixed #834 - Described how to optionally use `nonce` values in the Basic specification.
- Fixed #833 - Stated that an `at_hash` Claim MAY be present in the ID Token.
- Stated that sufficient entropy must be present in `nonce` values to prevent attackers from guessing values.
- Stated that the Authorization Server need not be listed as an audience of the ID Token when it is used as an `id_token_hint` value.
- Restricted the meaning of the `azp` (authorized party) Claim to simply be the single party to which the ID Token was issued.
- Stated that the JWS Compact Serialization is always used for JWS data structures.

-26

- Fixed #825 - Replaced `updated_time`, which used the RFC 3339 textual time format, with `updated_at`, using the numeric time format used by `iat`, etc.
- Fixed #829 - Stated that additional scope values can be defined and used and that scope values that are not understood should be ignored.
- Fixed #831 - Stated that JWS and JWE header parameters used to communicate key values and key references should not be used in ID Tokens, since these are communicated in advance using Discovery and Registration parameters.
- Fixed #712 and #830 - Clarified the `azp` description and made `azp` multi-valued, like `aud`.

-25

- Fixed #802 - Clarified recommendations and responsibilities for producing and consuming Claims with and without language tags.
- Fixed #797 - Clarified the intended semantics of e-mail verification and that the precise verification rules are context-specific.
- Fixed #806 - Added `phone_number_verified` Claim.
- Fixed #800 - Specified that phone number extensions are to be represented using RFC 3966 extension syntax.
- Fixed #795 - Specified that e-mail addresses must conform to the RFC 5322 addr-spec syntax.
- Fixed #808 - Specified that phone numbers may be used as `login_hint` values.
- Fixed #801 - Removed `schema` and `id` parameters to UserInfo Endpoint. Also fixed related issue #791 - Removed `invalid_schema` error.

- Fixed #793, #796, and #799 - Allow name Claims to contain multiple space-separated names.
- Fixed #794 - Required `picture` to refer to an image file that is a picture of the End-User.
- Fixed #811 - Specify that language tag components should be spelled using the character cases registered in the IANA Language Subtag Registry.
- Fixed #812 - Clarified that language tag values used need not be unnecessarily specific.
- Fixed #816 - Changed "must understand" language to "MUST be ignored if not understood".

-24

- Fixed #711 - Awkward phrase "The following Claims are REQUIRED and OPTIONAL".
- Fixed #712 - "azp" definition clarification.
- Fixed #713 - Explicitly require "sub" claim to be returned from UserInfo endpoint.
- Fixed #716 - Client/server 2119 blurriness.
- Fixed #732 - Capitalize name of "Bearer" authentication scheme.
- Fixed #738 - Behavior when "openid" scope is omitted.
- Added Security Considerations section about TLS version requirements and usage.
- Removed language about clients that do not support TLS. Also 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.
- Added `id_token_hint` parameter to Basic, since it SHOULD be present when `prompt=none` is used.
- Fixed #742 - Added new `ui_locales` parameter.
- Fixed #743 - Added `claims_locales` parameter.
- Fixed #744 - Added `max_age` parameter.
- Fixed #765 - Added new `acr_values` parameter.
- Fixed #597 - Changed representation of omitted year in `birthdate` from `9999` to `0000`.
- Fixed #726 - Client authentication clarifications.
- Clarified when the `http` scheme can and cannot be used in `redirect_uri` values.
- Stated that the `azp` Claim is only needed when the party requesting the ID Token is different than the audience of the ID Token.
- Use legal `acr` values in examples.
- Fixed #789 - Added `amr` (authentication methods references) Claim.

-23

- Fixed #620 - Update Section 2.2.6.2. to allow for other token types, but make bearer mandatory to support for basic clients.
- Added Implementation Considerations section.
- Fixed #698 - Inconsistent use of articles.
- Added auth_time definition to ID Token schema.
- Fixed #655 - Specify UTF-8 as encoding scheme whenever necessary.

-22

- 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 #660 - Clarified that returning the `sub` value from the UserInfo endpoint is mandatory.
- Fixed #636 - ID Token authorized party claim.
- Fixed #539 - Add scope for offline access.
- Fixed #689 - added caution about unrecognized audiences.
- Fixed #693 Added login_hint

- Updated scopes text.

-21

- added informative definition of nonce in 2.2.1
- Clarified that the client MUST check that the issuer is valid for the token endpoint
- RE #607 add example decoded id_token for non self-issued.
- Fixed #666 - JWS signature validation vs. verification.
- Fixed #682 - Change remaining uses of "birthday" to "birthdate".
- Referenced OAuth 2.0 RFCs -- RFC 6749 and RFC 6750.

-20

- Added `preferred_username` claim under `profile` scope
- Added ID Token section to describe required claims
- Added section on claim stability

-19

- Fixed Section 2.2.5.1 to return code in a query parameter rather than a fragment
- Removed `claims_in_id_token` scope value, per decision on June 15, 2012 special working group call

-18

- Use "code" response_type instead of "token id_token" in Basic Profile, per issue #567
- Changed `verified` to `email_verified`, per issue #564
- Removed Check ID Endpoint, per issue #570
- Removed requirement for ID Token signature validation from Basic Profile, per issue #568
- Removed use of `nonce` from Basic Profile, per issue #569
- Changed client.example.com to client.example.org, per issue #251
- Added claims_in_id_token scope definition to Basic and Implicit, per issue #594
- Use standards track version of JSON Web Token spec (draft-ietf-oauth-json-web-token)

-17

- Removed "embedded" display type, since its semantics were not well defined, per issue #514
- Add hash and hash check of access_token and code to id_token, per issue #510
- Add example JS code for client
- Updated Notices
- Updated References

-16

- Added iat as a required claim in ID Tokens
- Enumerated claims requested by the "profile" scope value
- Added text about implicit flow to Abstract

-15

- Removed definition and usage for assertion and claim object
- email scope allows access to the 'verified' claim
- Removed language pertaining to custom userinfo schemas
- Moved display=none to prompt=none
- Added additional 'display' parameter options
- Redefined 'nonce' in Authorization Request. Changed to REQUIRED parameter.
- Changed usage of "approval" to "consent"
- Use RFC 6125 to verify TLS endpoints
- Allow other gender strings in UserInfo schema
- ID Token MUST be JWT
- RECOMMENDED E.164 format for UserInfo 'phone_number' claim
- Changed UserInfo Error Response to augment and return OAuth 2.0 Bearer Token Error Response

- Check ID Endpoint SHOULD use POST
- Added section about string comparison rules needed
- Added Response Encoding according to Multiple Response Types spec
- Make openid scope provide `user_id` from userinfo endpoint
- Changed Security Considerations to refer to corresponding section in Standard
- 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_id_token error codes
- Replace queryString with postBody variable in example JS

-14

- Changed section 3.2.1 to refer to access_token ticket #134.
- Bumped version + date.
- Changed 7.4 in security considerations to show none is REQUIRED.
- Changed 3.2.4.1 User Info to UserInfo per Ticket #137.
- Changed formatting of 7.1 per ticket #140.

-13

- Changed check_session to check_id.
- schema=openid now required when requesting UserInfo.
- Removed issued_to, since not well defined.
- Removed display values popup, touch, and mobile, since not well defined.

-12

- Ticket #48 Changed Check Session to take the id_token as a parameter.

-11

- Renamed from "Lite" to "Basic Client".
- Numerous cleanups, including updating references.

-10

- Add back id_token to the response type per issue 27.
- Changed endpoint name in example from id_token to check_session.
- Added token_type to the response and explanations of the optional parameters.

-09

- Clean up typos.
- Clean up scope explanation.
- Fix 3.2.4.1 to include id_token in response.

-08

- Added note about OP needing to read the full spec.
- Reverted back to GET for introspection based on Google feedback.
- Changed scopes to `openid`, `profile`, `address`, and `email` to make them additive.
- Changed introspection to Check Session Endpoint to be consistent with session management.
- Changed validation rules, the Check session endpoint will return an error for expired or invalid tokens, so the Client does not need to check expiration.
- Added explanation of why an id_token is used to verify identity rather than the userinfo Access Token.

-07

- Changed introspection to post
- Changed userinfo from `id` to `user_id` to be consistent with introspection endpoint.
- Fixed introspection example to use id_token rather than access token.
- Removed asking for id_token in response type.

- Fixed Section 3 to be clear it is client secret that is maintained between the client and the OP.

-06

- Only require the `token` flow in Lite. Removed `code` flow.
- Make `id_token` required. The `id_token` is treated as opaque.
- Rearranged sections for readability.
- Dropped the `schema` parameter to the Introspection endpoint, which was formerly a string with the value `user_id`. This is unnecessary since the `id_token` parameter already can be used to disambiguate the intended uses(s) of the endpoint.
- Dropped the requested audience from the Lite spec, which was formerly the identifier of the target audience of the response. This could be part of the Standard spec, but is an advanced scenario, and so not appropriate for Lite.
- Reference the Discovery and Registration specs, since they're needed for interaction between non-pre-configured parties (so that OpenID Connect installations can be Open).

-05

- Corrected issues raised by Casper Biering.
- Created the OpenID Connect Lite specification.

-04

- Correct issues raised by Pam Dingle and discussed on the mailing list after the 7-Jul-11 working group call.
- Adopted long_names.

-03

- Correct issues raised by Johnny Bufu and discussed on the 7-Jul-11 working group call.

-02

- Consistency and cleanup pass, including removing unused references.

-01

- Initial draft

Authors' Addresses

Nat Sakimura
NAT.Consulting
Email: nat@nat.consulting
URI: http://nat.sakimura.org/

John Bradley
Yubico
Email: ve7jtb@ve7jtb.com
URI: http://www.thread-safe.com/

Michael B. Jones
Microsoft
Email: mbj@microsoft.com
URI: http://self-issued.info/

Breno de Medeiros
Google
Email: breno@google.com
URI: http://stackoverflow.com/users/311376/breno

Chuck Mortimore
VISA
Email: charliemortimore@gmail.com
URI: https://twitter.com/cmort