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

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| |  |  | | --- | --- | | Draft | N. Sakimura, Ed. | |  | NRI | |  | J. Bradley | |  | Protiviti | |  | B. de Medeiros | |  | Google | |  | M. Jones | |  | Microsoft | |  | E. Jay | |  | MGI1 | |  | October 14, 2011 | |

# OpenID Connect Standard 1.0 - draft 06

### Abstract

OpenID Connect 1.0 is a simple identity layer on top of OAuth 2.0 protocol. It allows a web site or application to verify the identity of the user based on the authentication performed by the authorization server, as well as to obtain basic profile information about the user in an interoperable and RESTful manner.

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

### Table of Contents

[**1.**](#rnc)  Requirements Notation and Conventions  
[**2.**](#terminology)  Terminology  
[**3.**](#anchor1)  HTTP Protocol Binding  
[**4.**](#anchor2)  Authorization Endpoint  
    [**4.1.**](#openid_scopes)  OpenID Connect Scopes  
    [**4.2.**](#protocol_flows)  Protocol Flows  
        [**4.2.1.**](#retrieving_code_token)  How to Get an Authorization Code, Access Token, and ID Token  
        [**4.2.2.**](#code_flow)  Authorization Code Flow  
        [**4.2.3.**](#implicit_flow)  Implicit Flow  
    [**4.3.**](#auth_request)  Authorization Request  
        [**4.3.1.**](#rf_prep)  Client Prepares an Authorization Request  
        [**4.3.2.**](#anchor6)  Authorization Server Authenticates the End-User  
        [**4.3.3.**](#anchor7)  Authorization Server Obtains the End-User Consent/Authorization  
        [**4.3.4.**](#art_res)  Authorization Server Sends the End-User Back to the Client   
[**5.**](#token_ep)  Token Endpoint  
    [**5.1.**](#anchor8)  Requesting an Access Token  
        [**5.1.1.**](#anchor9)  Access Token Request  
        [**5.1.2.**](#anchor10)  Access Token Response  
    [**5.2.**](#anchor13)  Refreshing an Access Token  
        [**5.2.1.**](#anchor14)  Positive Assertion  
[**6.**](#userinfo_ep)  UserInfo Endpoint  
    [**6.1.**](#anchor15)  UserInfo Request  
    [**6.2.**](#id_res)  UserInfo Response  
        [**6.2.1.**](#anchor16)  UserInfo Error Response  
[**7.**](#check_id_ep)  Check ID Endpoint  
    [**7.1.**](#anchor17)  Check ID Requests  
    [**7.2.**](#intro_res)  Check ID Response  
        [**7.2.1.**](#anchor18)  Check ID Error Response  
[**8.**](#session_eps)  Session Management Endpoints  
[**9.**](#disco_reg)  Discovery and Registration  
[**10.**](#security_considerations)  Security Considerations  
    [**10.1.**](#anchor19)  Implicit Grant Flow Threats  
[**11.**](#IANA)  IANA Considerations  
[**12.**](#rfc.references1)  Normative References  
[**Appendix A.**](#anchor21)  Footnotes  
    [**A.1.**](#anchor22)  Example JWT Values  
[**Appendix B.**](#anchor23)  Acknowledgements  
[**Appendix C.**](#anchor24)  Notices  
[**Appendix D.**](#anchor25)  Document History  
[**§**](#rfc.authors)  Authors' Addresses

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

### 1.  Requirements Notation and Conventions

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

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

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

### 2.  Terminology

Followings are the additional terminology defined in this specification in addition to those defined in [OpenID Connect Messages 1.0 (Sakimura, N., Recordon, D., Bradley, J., de Medeiros, B., Jones, M., and E. Jay, “OpenID Connect Messages 1.0,” October 2011.)](#OpenID.Messages) [OpenID.Messages] and [OAuth 2.0 (Hammer-Lahav, E., Ed., Recordon, D., and D. Hardt, “OAuth 2.0 Authorization Protocol,” September 2011.)](#OAuth.2.0) [OAuth.2.0].

Request File

A JSON structure that captures the [OpenID Connect Messages 1.0 (Sakimura, N., Recordon, D., Bradley, J., de Medeiros, B., Jones, M., and E. Jay, “OpenID Connect Messages 1.0,” October 2011.)](#OpenID.Messages) [OpenID.Messages] Authorization Request parameters that can be pointed at by a URL that is reachable by the Authorization Server.

Request URI

A URL that points to the Request File. It MUST be accessible by the Authorization Server.

Request Registration Endpoint

An HTTPS Endpoint URL provided by the Authorization Server that enables the client to store a Request File at the Authorization Server and receive back a Request URI that uniquely identifies the registered Request File.

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

### 3.  HTTP Protocol Binding

This specification describes the binding of the HTTP protocol with the various endpoints described in [OpenID Connect Messages (Sakimura, N., Recordon, D., Bradley, J., de Medeiros, B., Jones, M., and E. Jay, “OpenID Connect Messages 1.0,” October 2011.)](#OpenID.Messages) [OpenID.Messages].

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

### 4.  Authorization Endpoint

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

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

### 4.1.  OpenID Connect Scopes

Clients MUST specify the desired scopes in an authorization request to obtain an Access Token with the proper permissions.

OpenID Connect clients use scopes as defined in 3.3 of [OAuth 2.0 (Hammer-Lahav, E., Ed., Recordon, D., and D. Hardt, “OAuth 2.0 Authorization Protocol,” September 2011.)](#OAuth.2.0) [OAuth.2.0] to specify what access privileges are 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 request what information is to be made available from the UserInfo Endpoint and to request an ID Token. OAuth 2.0 allows additional scopes to be specified as extensions. This specification only describes scopes that are part of OpenID Connect.

OpenID Connect defines the following scopes:

openid

REQUIRED. Informs the Authorization Server that the client is making an OpenID request. If the openid scope is not specified, the Authorization Server SHOULD treat the request as a generic OAuth 2.0 request, and perform no OpenID Connect processing.

profile

OPTIONAL. Requests default profile information.

email

OPTIONAL. Requests an email address.

address

OPTIONAL. Requests an address.

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

The End-User may decline a scope request by the client.

To increase conversion, a client may elect to only request a subset of the information from the UserInfo Endpoint.

The following is a non-normative example of a Scope Request.

scope=openid profile email phone

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

### 4.2.  Protocol Flows

Authorization requests follow two main paths to obtain Access Tokens and ID Tokens, the Implicit Flow and the Authorization Code Flow. The flows determine how the Access Token and ID Token are returned to the client. Access Tokens contain Resource Owners' credentials to access protected resources. As such, they SHOULD NOT be exposed.

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

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

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

### 4.2.1.  How to Get an Authorization Code, Access Token, and ID Token

The Authorization server MUST support both the "code" and the "id\_token token" response\_type.

The client may request any OAuth 2.0 registered response type supported by the Authorization Server.

The OAuth 2.0 specification documents two response types:

code

When supplied as the value for the response\_type parameter, a successful response MUST include an authorization code as defined in the OAuth 2.0 specification. Both successful and error responses MUST be added as parameters to the query component of the response. All tokens are returned from the Token Endpoint. Authorization servers MUST support this response\_type.

token

When supplied as the value for the response\_type parameter, a successful response MUST include an access token as defined in the OAuth 2.0 specification. Both successful and error responses MUST be fragment-encoded. No id\_token is provided to the client.

Openid Connect supports these [additional flows (de Medeiros, B., Scurtescu, M., and P. Tarjan, “OAuth 2.0 Multiple Response Type Encoding Practices,” October 2011.)](#RESPONSE.TYPES) [RESPONSE.TYPES] that have been registered:

id\_token token

When supplied as the value for the response\_type parameter, a successful response MUST include both an access token as well as an id\_token. Both success and error responses SHOULD be fragment-encoded. Authorization servers MUST support this response\_type.

code token

When supplied as the value for the response\_type parameter, a successful response MUST include both an access token and an authorization code as defined in the OAuth 2.0 specification. Both successful and error responses SHOULD be fragment-encoded.

code id\_token

When supplied as the value for the response\_type parameter, a successful response MUST include both an authorization code as well as an id\_token. Both success and error responses SHOULD be fragment-encoded.

code id\_token token

When supplied as the value for the response\_type parameter, a successful response MUST include an authorization code, an id\_token, and an access token. Both success and error responses SHOULD be fragment-encoded.

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

### 4.2.2.  Authorization Code Flow

The Authorization Code Flow goes through the following steps.

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

Note that in each step, the party that receives a message MUST verify it according to the verification rule set in [OpenID Connect Messages 1.0 (Sakimura, N., Recordon, D., Bradley, J., de Medeiros, B., Jones, M., and E. Jay, “OpenID Connect Messages 1.0,” October 2011.)](#OpenID.Messages) [OpenID.Messages].

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

### 4.2.3.  Implicit Flow

The implicit flow follows the following steps:

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

Note that in each step, the party that receives a message MUST verify it according to the verification rule set in [OpenID Connect Messages 1.0 (Sakimura, N., Recordon, D., Bradley, J., de Medeiros, B., Jones, M., and E. Jay, “OpenID Connect Messages 1.0,” October 2011.)](#OpenID.Messages) [OpenID.Messages].

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

### 4.3.  Authorization Request

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

Authorization servers MUST require the use of a transport-layer security mechanism at the Authorization Endpoint. The Authorization Server MUST support TLS 1.2 as described in [RFC 5246 (Dierks, T. and E. Rescorla, “The Transport Layer Security (TLS) Protocol Version 1.2,” August 2008.)](#RFC5246) [RFC5246] and MAY support other transport-layer mechanisms with equivalent security.

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

Clients MAY use the HTTP "GET" or "POST" method 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 as defined in [OpenID Connect Messages 1.0 (Sakimura, N., Recordon, D., Bradley, J., de Medeiros, B., Jones, M., and E. Jay, “OpenID Connect Messages 1.0,” October 2011.)](#OpenID.Messages) [OpenID.Messages]. If using the HTTP "POST" method, the request parameters are added to the HTTP request entity-body using "application/x-www-form-urlencoded" format.

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

### 4.3.1.  Client Prepares an Authorization Request

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

The required Authorization Request parameters are as follows:

response\_type

An OAuth 2.0 registered response type that determines how the authorization response is returned to the client. As described in [How to Get an Authorization Code, Access Token, and ID Token (How to Get an Authorization Code, Access Token, and ID Token)](#retrieving_code_token) , the following registered values are supported:

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

client\_id

The client identifier.

scope

It MUST include openid as one of the strings. Other values that MAY be included are profile, email, address, and PPID. The values specify an additive list of claims that are returned by the UserInfo Endpoint as described by [OpenID Connect Scopes (OpenID Connect Scopes)](#openid_scopes).

redirect\_uri

A redirection URI where the response will be sent.

nonce

A random, unique string value used to mitigate replay attacks. The value is passed through unmodified to the id\_token.

The request can contain the following optional parameters:

state

An opaque value used to maintain state between the request and the callback.

request

A [JWT (Jones, M., Balfanz, D., Bradley, J., Goland, Y., Panzer, J., Sakimura, N., and P. Tarjan, “JSON Web Token,” July 2011.)](#JWT) [JWT] encoded [OpenID Request Object](http://openid.net/specs/openid-connect-messages-1_0.html#OpenIDReq).

request\_uri

A URL that points to an [OpenID Request Object](http://openid.net/specs/openid-connect-messages-1_0.html#OpenIDReq).

display

A string value that specifies how the Authorization Server displays the authentication page to the user. The following value is supported: none. Refer to [OpenID Connect Messages 1.0 (Sakimura, N., Recordon, D., Bradley, J., de Medeiros, B., Jones, M., and E. Jay, “OpenID Connect Messages 1.0,” October 2011.)](#OpenID.Messages) [OpenID.Messages] for more information.

prompt

A space delimited list that can contain login, consent, and select\_account. Refer to [OpenID Connect Messages 1.0 (Sakimura, N., Recordon, D., Bradley, J., de Medeiros, B., Jones, M., and E. Jay, “OpenID Connect Messages 1.0,” October 2011.)](#OpenID.Messages) [OpenID.Messages] for more information

audience

The identifier of the target audience for an ID token.

There are three methods to construct and send the request to the authorization endpoint:

a.

Simple Request Method

b.

Request Parameter Method

c.

Request File Method

The Simple Request Method is used in simple cases when default UserInfo and ID Token claims are desired and requests and responses do not need to be signed or encrypted.

The Request Parameter Method is used by sending an OpenID Request Object when the client desires to retrieve a different set of UserInfo and ID Token claims. The request parameter method also allows requests and responses to be signed or encrypted.

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

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

### 4.3.1.1.  Simple Request Method

The Client prepares an Authorization Request to the Authorization endpoint using the appropriate parameters. If using the HTTP "GET" method, the request parameters are serialized using URI query string serialization as defined in [OpenID Connect Messages 1.0 (Sakimura, N., Recordon, D., Bradley, J., de Medeiros, B., Jones, M., and E. Jay, “OpenID Connect Messages 1.0,” October 2011.)](#OpenID.Messages) [OpenID.Messages]. If using the HTTP "POST" method, the request parameters are added to the HTTP request entity-body using "application/x-www-form-urlencoded" format.

The following is a non-normative example of an Authorization Request URL. Note that the line wraps within the values are for display purpose only:

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

response\_type=code%20id\_token

&client\_id=s6BhdRkqt3

&redirect\_uri=https%3A%2F%2Fclient%2Eexample%2Ecom%2Fcb

&scope=openid

&nonce=n-0S6\_WzA2Mj

&state=af0ifjsldkj

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

### 4.3.1.1.1.  Client sends a request to the Authorization Server

Having constructed the Authorization Request, the client sends it to the HTTPS End-User Authorization Endpoint. This MAY happen via HTTPS redirect, hyperlinking, or any other means of directing the User-Agent to the Authorization Endpoint URL.

Following is a non-normative example using HTTP redirect. Note: Line wraps are for display purpose only.

HTTP/1.1 302 Found

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

response\_type=code%20id\_token

&client\_id=s6BhdRkqt3

&redirect\_uri=https%3A%2F%2Fclient%2Eexample%2Ecom%2Fcb

&scope=openid

&nonce=n-0S6\_WzA2Mj

&state=af0ifjsldkj

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

### 4.3.1.2.  Request Parameter Method

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

The Authorization Request MUST include the request parameter defined in the [Client Prepares an Authorization Request (Client Prepares an Authorization Request)](#rf_prep) section. The Authorization Request MUST NOT include the request\_uri parameter.

The request parameter is a [JWT (Jones, M., Balfanz, D., Bradley, J., Goland, Y., Panzer, J., Sakimura, N., and P. Tarjan, “JSON Web Token,” July 2011.)](#JWT) [JWT] encoded [OpenID Request Object](http://openid.net/specs/openid-connect-messages-1_0.html#OpenIDReq) which contains the Authorization Request and also specifies the content and format of the responses returned from the UserInfo and Check ID Endpoints. The JWT object MAY be signed or signed and encrypted via [JWS (Jones, M., Balfanz, D., Bradley, J., Goland, Y., Panzer, J., Sakimura, N., and P. Tarjan, “JSON Web Signatures,” April 2011.)](#JWS) [JWS] and [JWE (Jones, M., Bradley, J., and N. Sakimura, “JSON Web Encryption,” July 2011.)](#JWE) [JWE] respectively, thereby providing authentication, integrity, non-repudiation and/or confidentiality.

All Authorization Request parameters that are part of the Authorization Request excluding the request and request\_uri parameters MUST also be [JSON Serialized](http://openid.net/specs/openid-connect-messages-1_0.html#js) into the [OpenID Request Object](http://openid.net/specs/openid-connect-messages-1_0.html#OpenIDReq) with the same values. This allows the client to send signed and/or encrypted requests to the Authorization Server and maintain conformance to OAuth 2.0.

The following is a non-normative example of an OpenID Request Object before JWT encoding. Note that the line wraps within the values are for display purpose only:

{

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

"client\_id": "s6BhdRkqt3",

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

"scope": "openid profile",

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

"nonce": "af0ifjsldkj",

"userinfo":

{

"claims":

{

"name": null,

"nickname": {"optional": true},

"email": null,

"verified": null,

"picture": {"optional": true},

},

"format": "signed"

}

"id\_token":

{

"max\_age": 86400,

"iso29115": "2"

}

}

The following is a non-normative example of a [JWT (Jones, M., Balfanz, D., Bradley, J., Goland, Y., Panzer, J., Sakimura, N., and P. Tarjan, “JSON Web Token,” July 2011.)](#JWT) [JWT] encoded OpenID Request Object. Note that the line wraps within the values are for display purpose only:

JWT algorithm = HS256

HMAC HASH Key = 'aaa'

JSON Encoded Header = "{"alg":"HS256","typ":"JWT"}"

JSON Encoded Payload = "{"response\_type":"code id\_token",

"client\_id":"s6BhdRkqt3",

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

"scope":"openid profile",

"state":"af0ifjsldkj",

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

"userinfo":{"claims":{"name":null,"nickname":{"optional":true},

"email":null,"verified":null,

"picture":{"optional":true}},"format":"signed"},

"id\_token":{"max\_age":86400,"iso29115":"2"}}"

JWT = eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9.eyJyZXNwb25zZV90eXBlIjoiY29kZ

SBpZF90b2tlbiIsImNsaWVudF9pZCI6InM2QmhkUmtxdDMiLCJyZWRpcmVjdF91cmkiO

iJodHRwczpcL1wvY2xpZW50LmV4YW1wbGUuY29tXC9jYiIsInNjb3BlIjoib3BlbmlkI

HByb2ZpbGUiLCJzdGF0ZSI6ImFmMGlmanNsZGtqIiwidXNlcmluZm8iOnsiY2xhaW1zI

jp7Im5hbWUiOm51bGwsIm5pY2tuYW1lIjp7Im9wdGlvbmFsIjp0cnVlfSwiZW1haWwiO

m51bGwsInZlcmlmaWVkIjpudWxsLCJwaWN0dXJlIjp7Im9wdGlvbmFsIjp0cnVlfX0sI

mZvcm1hdCI6InNpZ25lZCJ9LCJpZF90b2tlbiI6eyJtYXhfYWdlIjo4NjQwMCwiaXNvM

jkxMTUiOiIyIn19.2OiqRgrbrHkA1FZ5p\_7bc\_RSdTbH-wo\_Agk-ZRpD3wY

The following is a non-normative example of an Authorization Request with the OpenID Request Method. Note that the line wraps within the values are for display purpose only:

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

response\_type=code%02id\_token

&client\_id=s6BhdRkqt3

&redirect\_uri=https%3A%2F%2Fclient%2Eexample%2Ecom%2Fcb

&scope=openid

&state=af0ifjsldkj

&nonce=n-0S6\_WzA2Mj

&request=eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9.eyJyZXNwb25zZV90eXBlIjoiY2

9kZSBpZF90b2tlbiIsImNsaWVudF9pZCI6InM2QmhkUmtxdDMiLCJyZWRpcmVjdF91cmkiOi

JodHRwczpcL1wvY2xpZW50LmV4YW1wbGUuY29tXC9jYiIsInNjb3BlIjoib3BlbmlkIHByb2

ZpbGUiLCJzdGF0ZSI6ImFmMGlmanNsZGtqIiwidXNlcmluZm8iOnsiY2xhaW1zIjp7Im5hbW

UiOm51bGwsIm5pY2tuYW1lIjp7Im9wdGlvbmFsIjp0cnVlfSwiZW1haWwiOm51bGwsInZlcm

lmaWVkIjpudWxsLCJwaWN0dXJlIjp7Im9wdGlvbmFsIjp0cnVlfX0sImZvcm1hdCI6InNpZ2

5lZCJ9LCJpZF90b2tlbiI6eyJtYXhfYWdlIjo4NjQwMCwiaXNvMjkxMTUiOiIyIn19.2OiqR

grbrHkA1FZ5p\_7bc\_RSdTbH-wo\_Agk-ZRpD3wY

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

### 4.3.1.2.1.  Client Sends a Request to the Authorization Server

Having constructed the Authorization Request, the client sends it to the HTTPS Authorization Endpoint. This MAY happen via HTTPS redirect, hyperlinking, or any other means of directing the User-Agent to the Authorization Endpoint.

Following is a non-normative example using HTTP redirect. Note: Line wraps are for display purpose only.

HTTP/1.1 302 Found

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

response\_type=code%20id\_token

&client\_id=s6BhdRkqt3

&redirect\_uri=https%3A%2F%2Fclient%2Eexample%2Ecom%2Fcb

&scope=openid

&state=af0ifjsldkj

&nonce=n-0S6\_WzA2Mj

&request=eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9.eyJyZXNwb25zZV90eXBlIjoiY2

9kZSBpZF90b2tlbiIsImNsaWVudF9pZCI6InM2QmhkUmtxdDMiLCJyZWRpcmVjdF91cmkiOi

JodHRwczpcL1wvY2xpZW50LmV4YW1wbGUuY29tXC9jYiIsInNjb3BlIjoib3BlbmlkIHByb2

ZpbGUiLCJzdGF0ZSI6ImFmMGlmanNsZGtqIiwidXNlcmluZm8iOnsiY2xhaW1zIjp7Im5hbW

UiOm51bGwsIm5pY2tuYW1lIjp7Im9wdGlvbmFsIjp0cnVlfSwiZW1haWwiOm51bGwsInZlcm

lmaWVkIjpudWxsLCJwaWN0dXJlIjp7Im9wdGlvbmFsIjp0cnVlfX0sImZvcm1hdCI6InNpZ2

5lZCJ9LCJpZF90b2tlbiI6eyJtYXhfYWdlIjo4NjQwMCwiaXNvMjkxMTUiOiIyIn19.2OiqR

grbrHkA1FZ5p\_7bc\_RSdTbH-wo\_Agk-ZRpD3wY

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

### 4.3.1.3.  Request File Method

The Request File Method differs from the other methods in that it uses a request file which contains a JSON encoded [OpenID Request Object](http://openid.net/specs/openid-connect-messages-1_0.html#OpenIDReq). It then sends the request file URL as part of the Authorization Request.

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

The Authorization Request MUST NOT include the request parameter. The Authorization Request MUST include the request\_uri parameter which is a globally reachable URL. The contents of the URL is a JWT which is a [OpenID Request Object](http://openid.net/specs/openid-connect-messages-1_0.html#OpenIDReq).

All Authorization Request parameters that are part of the Authorization Request excluding the request parameter MUST also be [JSON Serialized](http://openid.net/specs/openid-connect-messages-1_0.html#js) into the [OpenID Request Object](http://openid.net/specs/openid-connect-messages-1_0.html#OpenIDReq) with the same values. This allows the client to send signed and/or encrypted requests to the Authorization Server and maintain conformance to OAuth 2.0. The [OpenID Request Object](http://openid.net/specs/openid-connect-messages-1_0.html#OpenIDReq) MAY be signed and encrypted via [JWS (Jones, M., Balfanz, D., Bradley, J., Goland, Y., Panzer, J., Sakimura, N., and P. Tarjan, “JSON Web Signatures,” April 2011.)](#JWS) [JWS] and [JWE (Jones, M., Bradley, J., and N. Sakimura, “JSON Web Encryption,” July 2011.)](#JWE) [JWE] respectively, thereby providing authentication, integrity, non-repudiation and/or confidentiality.

Following is a non-normative example of a Request File. Note that the line wraps within the values are for display purpose only:

{

eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9.eyJyZXNwb25zZV90eXBlIjoiY29kZSBpZF9

0b2tlbiIsImNsaWVudF9pZCI6InM2QmhkUmtxdDMiLCJyZWRpcmVjdF91cmkiOiJodHRwczp

cL1wvY2xpZW50LmV4YW1wbGUuY29tXC9jYiIsInNjb3BlIjoib3BlbmlkIHByb2ZpbGUiLCJ

zdGF0ZSI6ImFmMGlmanNsZGtqIiwidXNlcmluZm8iOnsiY2xhaW1zIjp7Im5hbWUiOm51bGw

sIm5pY2tuYW1lIjp7Im9wdGlvbmFsIjp0cnVlfSwiZW1haWwiOm51bGwsInZlcmlmaWVkIjp

udWxsLCJwaWN0dXJlIjp7Im9wdGlvbmFsIjp0cnVlfX0sImZvcm1hdCI6InNpZ25lZCJ9LCJ

pZF90b2tlbiI6eyJtYXhfYWdlIjo4NjQwMCwiaXNvMjkxMTUiOiIyIn19.2OiqRgrbrHkA1F

Z5p\_7bc\_RSdTbH-wo\_Agk-ZRpD3wY

}

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

### 4.3.1.3.1.  Client Obtains the URL of the Request File

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

Optionally, the Authorization Server may provide the Request File registration service at the Request Registration Endpoint, which allows the Client to register the Request File and obtain the URL for it in exchange. This is especially useful for the cases when the RP is behind the firewall or lives on a client device that cannot be accessed from the Authorization Server.

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

### 4.3.1.3.2.  Client Sends a Request to Authorization Server via HTTPS Redirect

The Client sends the Authorization Request to the Authorization Endpoint.

The entire URL MUST NOT exceed 512 bytes.

Following is a non-normative example. Note: Line wraps are for display purpose only:

HTTP/1.1 302 Found

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

?response\_type=code%20id\_token

&cliend\_id=s6BhdRkqt3

&request\_uri=https%3A%2F%2Fclient%2Eexample%2Ecom%2Frf%2Ejs

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

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

### 4.3.1.3.3.  Authorization Server Fetches the Request File

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

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

Following is a non-normative example of this fetch process. Note: Line wraps are for display purpose only:

GET /rf.js HTTP/1.1

Host: client.example.com

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

### 4.3.2.  Authorization Server Authenticates the End-User

The Authorization Server validates the request to ensure all required parameters are present and valid. If the request is valid, the authorization server MUST authenticate the End-User. The way in which the authorization server authenticates the End-User (e.g. username and password login, session cookies) is beyond the scope of this specification. An authentication user interface MAY be displayed by the Authorization Server depending on the authentication method used.

The Authorization Server MUST attempt to authenticate the user in the following cases:

* The End-User is not already authenticated by the Authorization Server.
* The Authorization Request contains the prompt parameter with the value "login". The Authorization Server MUST reauthenticate the user even if the user is already authenticated.

The Authorization Server MUST NOT attempt authentication in the following cases:

* The Authorization Request contains the "display" parameter with the value "none". The Authorization Server MUST return an error if the user is unauthenticated.

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

### 4.3.3.  Authorization Server Obtains the End-User Consent/Authorization

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

The Authorization Server MUST attempt to request authorization from the user in the following cases:

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

The Authorization Server MUST NOT request user authorization in the following cases:

* The Authorization Request contains the "display" parameter with the value "none". The Authorization Server MUST return an error if the user has not pre-authorized the client.

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

### 4.3.4.  Authorization Server Sends the End-User Back to the Client

Once the authorization is determined, the Authorization Server returns a positive or negative response.

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

### 4.3.4.1.  End-User Grants Authorization

If the resource owner grants the access request, the Authorization Server issues an Authorization Response and delivers it to the client by adding the response parameters to redirection URI specified in the Authorization Request using the "application/x-www-form-urlencoded" format.

If the response\_type parameter in the Authorization Request includes the string value "code", the following response parameters are included:

code

REQUIRED. The Authorization Code.

Upon receipt of the "code", the Client requests an Assertion that includes the "access\_token" and other variables. The requests and responses are described in the [Token endpoint (Token Endpoint)](#token_ep) section.

If the response\_type parameter in the Authorization Request includes the string value "token", the following response parameters are included:

access\_token

REQUIRED. The Access Token

token\_type

REQUIRED. This specification only supports the [Bearer Token (Jones, M., Hardt, D., and D. Recordon, “OAuth 2.0 Protocol: Bearer Tokens,” September 2011.)](#OAuth.2.0.Bearer) [OAuth.2.0.Bearer] scheme. As such, this value MUST be set to "Bearer".

expires\_in

OPTIONAL. The lifetime in seconds of the Access Token.

scope

OPTIONAL. The scopes of the issued Access Token.

If the response\_type parameter in the Authorization Request includes the string value "id\_token", the following response parameters are returned:

id\_token

REQUIRED. The ID Token of the for the authentication session.

The response parameters MAY also be include:

state

REQUIRED if the state parameter is present in the Authorization Request. Set it to the exact value of the state parameter received from the client.

No other parameter SHOULD be returned.

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

The following are some non-normative examples. Line wraps are for display purpose only:

HTTP/1.1 302 Found

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

code=Qcb0Orv1zh30vL1MPRsbm-diHiMwcLyZvn1arpZv-Jxf\_11jnpEX3Tgfvk

&state=af0ifjsldkj

HTTP/1.1 302 Found

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

token=jHkWEdUXMU1BwAsC4vtUsZwnNvTIxEl0z9K3vx5KF0Y

&token\_type=Bearer

&state=af0ifjsldkj

HTTP/1.1 302 Found

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

code=Qcb0Orv1zh30vL1MPRsbm-diHiMwcLyZvn1arpZv-Jxf\_11jnpEX3Tgfvk

&token=jHkWEdUXMU1BwAsC4vtUsZwnNvTIxEl0z9K3vx5KF0Y

&token\_type=Bearer

&state=af0ifjsldkj

HTTP/1.1 302 Found

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

code=Qcb0Orv1zh30vL1MPRsbm-diHiMwcLyZvn1arpZv-Jxf\_11jnpEX3Tgfvk

&token=jHkWEdUXMU1BwAsC4vtUsZwnNvTIxEl0z9K3vx5KF0Y

&token\_type=Bearer

&id\_token=eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9.eyJpc3MiOiJodHRwOlwvXC9zZ

XJ2ZXIuZXhhbXBsZS5jb20iLCJ1c2VyX2lkIjoiMjQ4Mjg5NzYxMDAxIiwiYXVkIjoiaHR0c

DpcL1wvY2xpZW50LmV4YW1wbGUuY29tIiwiZXhwIjoxMzExMjgxOTcwfQ.eDesUD0vzDH3T1

G3liaTNOrfaeWYjuRCEPNXVtaazNQ

&state=af0ifjsldkj

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

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

If the user denies the authorization or the user authentication fails, the authorization server MUST return the negative authorization response as defined in [OpenID Connect Messages 1.0 (Sakimura, N., Recordon, D., Bradley, J., de Medeiros, B., Jones, M., and E. Jay, “OpenID Connect Messages 1.0,” October 2011.)](#OpenID.Messages) [OpenID.Messages]. The authorization server returns the client to the redirection URI specified in the authorization request with the appropriate error parameters. No other parameters SHOULD be returned.

The error response parameters are the following:

error

REQUIRED. The error code.

error\_description

OPTIONAL. A human-readable UTF-8 encoded text description of the error.

error\_uri

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

state

REQUIRED if the Authorization Request included the state parameter. Set to the exact value received from the client.

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

The following is a non-normative example. Line wraps after the second line is for the display purpose only:

HTTP/1.1 302 Found

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

error=invalid\_request

&error\_description=the%20request%20is%20not%20valid%20or%20malformed

&state=af0ifjsldkj

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

### 5.  Token Endpoint

The Token endpoint handles requests for retrieving and refreshing access tokens.

Clients MUST use the HTTP "POST" method to make requests to the Token endpoint. Request parameters are added to the HTTP request entity-body using the application/x-www-form-urlencoded format.

Clients MAY provide authentication parameters in the request to the Token endpoint as described in section 3.2 of [OpenID Connect Messages (Sakimura, N., Recordon, D., Bradley, J., de Medeiros, B., Jones, M., and E. Jay, “OpenID Connect Messages 1.0,” October 2011.)](#OpenID.Messages) [OpenID.Messages] and section 3.2.1 of [OAuth 2.0 (Hammer-Lahav, E., Ed., Recordon, D., and D. Hardt, “OAuth 2.0 Authorization Protocol,” September 2011.)](#OAuth.2.0) [OAuth.2.0].

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

Authorization servers MUST require the use of a transport-layer security mechanism. The authorization server MUST support TLS 1.2 as described in [RFC 5246 (Dierks, T. and E. Rescorla, “The Transport Layer Security (TLS) Protocol Version 1.2,” August 2008.)](#RFC5246) [RFC5246] and MAY support other transport-layer mechanisms with equivalent security.

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

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

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

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

### 5.1.  Requesting an Access Token

To retrieve an access token, a client MUST have an Authorization Code obtained via the method as described in [Authorization Code Flow (Authorization Code Flow)](#code_flow).

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

### 5.1.1.  Access Token Request

To obtain an access token assertion, the client MUST send the following parameters via HTTPS POST to the Token endpoint using application/x-www-form-urlencoded format in the HTTP request entity-body as specified in Section 4.1.3 of [OAuth 2.0 (Hammer-Lahav, E., Ed., Recordon, D., and D. Hardt, “OAuth 2.0 Authorization Protocol,” September 2011.)](#OAuth.2.0) [OAuth.2.0]:

grant\_type

REQUIRED. A string "authorization\_code".

code

REQUIRED. The authorization code received from the authorization server.

redirect\_uri

REQUIRED, if the redirect\_uri parameter was included in the authorization request described in Section 4.1.1 of [OAuth 2.0 (Hammer-Lahav, E., Ed., Recordon, D., and D. Hardt, “OAuth 2.0 Authorization Protocol,” September 2011.)](#OAuth.2.0) [OAuth.2.0], and their values MUST be identical.

If the client type is confidential or was issued client credentials (or assigned other authentication requirements), the client MUST authenticate with the authorization server as described in Section 3.2.1 of [OAuth 2.0 (Hammer-Lahav, E., Ed., Recordon, D., and D. Hardt, “OAuth 2.0 Authorization Protocol,” September 2011.)](#OAuth.2.0) [OAuth.2.0].

The client MAY authenticate by sending the client\_id and client\_secret using the HTTP Basic authentication scheme as defined in [RFC2617 (Franks, J., Hallam-Baker, P., Hostetler, J., Lawrence, S., Leach, P., Luotonen, A., and L. Stewart, “HTTP Authentication: Basic and Digest Access Authentication,” June 1999.)](#RFC2617) [RFC2617]. Alternatively, the client MAY send the following client credentials in the HTTP request-entity body:

client\_id

REQUIRED. The client identifier of the client.

secret\_type

OPTIONAL. Specifies the client authentication type which determines how the client\_secret parameter is interpreted. It can be one of "basic" or "JWT". The defaults value is "basic". If the value is "basic", the client is performing symmetric key authentication as specified in OAuth 2.0. If the value is "JWT", the client is performing asymmetric key authentication.

client\_secret

REQUIRED. The client secret. If the secret\_type is "basic", the value is the pre-shared secret that was issued to the client during client registration. If the "secret\_type" is "JWT", the value is the [JWS (Jones, M., Balfanz, D., Bradley, J., Goland, Y., Panzer, J., Sakimura, N., and P. Tarjan, “JSON Web Signatures,” April 2011.)](#JWS) [JWS] signature of a JSON object containing one of the following claims:

code

REQUIRED. The authorization code that was issued by the authorization server.

refresh\_token

REQUIRED. The refresh token that was issued.

Authorization servers MUST support both forms of client authentication at the Token Endpoint.

The following is a non-normative example. Line wraps after line 4 are for display purpose only:

POST /token HTTP/1.1

Host: server.example.com

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

Authorization: Basic czZCaGRSa3F0MzpnWDFmQmF0M2JW

Content-Type: application/x-www-form-urlencoded;charset=UTF-8

grant\_type=authorization\_code&code=SplxlOBeZQQYbYS6WxSbIA

&redirect\_uri=https%3A%2F%2Fclient%2Eexample%2Ecom%2Fcb

The authorization server MUST:

* Require client authentication for confidential clients or for any client issued client credentials (or with other authentication requirements),
* Authenticate the client if client authentication is included and ensure the authorization code was issued to the authenticated client,
* Verify that the authorization code is valid, and
* Ensure that the redirect\_uri parameter is present if the redirect\_uri parameter was included in the initial authorization request and that their values are identical.

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

### 5.1.2.  Access Token Response

Upon receipt of the Token Request, the Authorization Server MUST return either a Positive or a Negative Assertion that corresponds to the received Authorization Code.

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

### 5.1.2.1.  Positive Assertion

A Positive Assertion response returns the "application/json" media type and the response body is the Access Token Response of the [OpenID Connect Messages 1.0 (Sakimura, N., Recordon, D., Bradley, J., de Medeiros, B., Jones, M., and E. Jay, “OpenID Connect Messages 1.0,” October 2011.)](#OpenID.Messages) [OpenID.Messages].

The assertion is a JSON structure which contains the following values:

access\_token

REQUIRED. The Access Token.

token\_type

REQUIRED. Specifies the access token type. This specification only supports the "Bearer" token type.

id\_token

REQUIRED if scope in the Authorization Request contains the value "openid". The ID Token associated with the authentication session.

refresh\_token

OPTIONAL. The Refresh Token that can be used to obtain a new Access Token.

expires\_in

OPTIONAL. The lifetime of the Access Token in seconds.

scope

OPTIONAL. The scope of the Access Token.

Following is a non-normative example of the Positive Assertion:

HTTP/1.1 200 OK

Content-Type: application/json

Cache-Control: no-store

Pragma: no-cache

{

"access\_token": "SlAV32hkKG",

"token\_type": "Bearer",

"refresh\_token": "8xLOxBtZp8",

"expires\_in": 3600,

"id\_token": "eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9.eyJpc3MiOiJodHRwOl

wvXC9zZXJ2ZXIuZXhhbXBsZS5jb20iLCJ1c2VyX2lkIjoiMjQ4Mjg5NzYxMDAxIiwiYXVkIj

oiaHR0cDpcL1wvY2xpZW50LmV4YW1wbGUuY29tIiwiZXhwIjoxMzExMjgxOTcwfQ.eDesUD0

vzDH3T1G3liaTNOrfaeWYjuRCEPNXVtaazNQ"

}

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

### 5.1.2.2.  Error Response

If the Token Request is invalid or unauthorized, the Authorization Server constructs the response by returning the Token Error Response defined in [OpenID Connect Messages 1.0 (Sakimura, N., Recordon, D., Bradley, J., de Medeiros, B., Jones, M., and E. Jay, “OpenID Connect Messages 1.0,” October 2011.)](#OpenID.Messages) [OpenID.Messages] in the entity body of the HTTP response using the application/json media type with HTTP response code 400.

Following is a non-normative example:

HTTP/1.1 400 Bad Request

Content-Type: application/json

Cache-Control: no-store

Pragma: no-cache

{

"error":"invalid\_request"

}

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

### 5.2.  Refreshing an Access Token

To refresh an Access token that has expired, the client sends a POST request to the Token endpoint with the following parameters in the entity-body using the application/x-www-form-urlencoded format:

grant\_type

REQUIRED. A string "refresh\_token".

refresh\_token

REQUIRED. The refresh token that was issued in a previous Token endpoint request.

scope

OPTIONAL. The scope of the access request. If not omitted is treated as equal to the scope originally granted by the resource owner. If omitted or includes openid as one of the strings, an id\_token is returned in addition to the access\_token.

The Authorization Server MUST verify the validity of the refresh token.

If the client type is confidential or was issued client credentials (or assigned other authentication requirements), the client MUST authenticate with the authorization server as described in Section 3.2.1 of [OAuth 2.0 (Hammer-Lahav, E., Ed., Recordon, D., and D. Hardt, “OAuth 2.0 Authorization Protocol,” September 2011.)](#OAuth.2.0) [OAuth.2.0].

The client MAY authenticate by sending the client\_id and client\_secret using the HTTP Basic authentication scheme as defined in [RFC2617 (Franks, J., Hallam-Baker, P., Hostetler, J., Lawrence, S., Leach, P., Luotonen, A., and L. Stewart, “HTTP Authentication: Basic and Digest Access Authentication,” June 1999.)](#RFC2617) [RFC2617]. Alternatively, the client MAY send the following client credentials in the HTTP request-entity body:

client\_id

REQUIRED. The client identifier of the client.

secret\_type

OPTIONAL. Specifies the client authentication type which determines how the client\_secret parameter is interpreted. It can be one of "basic" or "JWT". The defaults value is "basic". If the value is "basic", the client is performing symmetric key authentication as specified in OAuth 2.0. If the value is "JWT", the client is performing asymmetric key authentication.

client\_secret

REQUIRED. The client secret. If the secret\_type is "basic", the value is the pre-shared secret that was issued to the client during client registration. If the "secret\_type" is "JWT", the value is the [JWS (Jones, M., Balfanz, D., Bradley, J., Goland, Y., Panzer, J., Sakimura, N., and P. Tarjan, “JSON Web Signatures,” April 2011.)](#JWS) [JWS] signature of a JSON object containing one of the following claims:

code

REQUIRED. The authorization code that was issued by the authorization server.

refresh\_token

REQUIRED. The refresh token that was issued.

Authorization servers MUST support both forms of client authentication at the Token Endpoint.

The Authorization Server MUST verify the validity of the refresh token.

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

### 5.2.1.  Positive Assertion

Upon successful verification of the refresh token, a positive assertion response returns the "application/json" media type and the response body is the Access Token Response of [Section 3.2.2 of OpenID Connect Messages 1.0 (Sakimura, N., Recordon, D., Bradley, J., de Medeiros, B., Jones, M., and E. Jay, “OpenID Connect Messages 1.0,” October 2011.)](#OpenID.Messages) [OpenID.Messages].

Following is a non-normative example of the refresh token request and response:

POST /token HTTP/1.1

Host: server.example.com

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

client\_id=s6BhdRkqt3

&client\_secret=some\_secret12345

&grant\_type=refresh\_token

&refresh\_token=8xLOxBtZp8

&scope=openid profile

HTTP/1.1 200 OK

Content-Type: application/json

Cache-Control: no-store

Pragma: no-cache

{

"access\_token": "TlBN45jURg",

"token\_type": "Bearer",

"refresh\_token": "9yNOxJtZa5",

"expires\_in": 3600,

"id\_token": "eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9.eyJpc3MiOiJodHRwOl

wvXC9zZXJ2ZXIuZXhhbXBsZS5jb20iLCJ1c2VyX2lkIjoiMjQ4Mjg5NzYxMDAxIiwiYXVkIj

oiaHR0cDpcL1wvY2xpZW50LmV4YW1wbGUuY29tIiwiZXhwIjoxMzExMjgxOTcwfQ.eDesUD0

vzDH3T1G3liaTNOrfaeWYjuRCEPNXVtaazNQ"

}

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

### 6.  UserInfo Endpoint

To obtain the additional attributes and tokens/assertions, the client makes a GET or POST request to the UserInfo Endpoint as in [OpenID Connect Messages 1.0 (Sakimura, N., Recordon, D., Bradley, J., de Medeiros, B., Jones, M., and E. Jay, “OpenID Connect Messages 1.0,” October 2011.)](#OpenID.Messages) [OpenID.Messages].

Authorization Servers MUST require the use of a transport-layer security mechanism. The authorization server MUST support TLS 1.2 as described in [RFC 5246 (Dierks, T. and E. Rescorla, “The Transport Layer Security (TLS) Protocol Version 1.2,” August 2008.)](#RFC5246) [RFC5246] and MAY support other transport-layer mechanisms with equivalent security.

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

Authorization servers MUST be able to accept Access Tokens via [Bearer Tokens (Jones, M., Hardt, D., and D. Recordon, “OAuth 2.0 Protocol: Bearer Tokens,” September 2011.)](#OAuth.2.0.Bearer) [OAuth.2.0.Bearer] specification.

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

### 6.1.  UserInfo Request

Client SHOULD send the UserInfo request defined in section 3.3 of the [OpenID Connect Messages 1.0 (Sakimura, N., Recordon, D., Bradley, J., de Medeiros, B., Jones, M., and E. Jay, “OpenID Connect Messages 1.0,” October 2011.)](#OpenID.Messages) [OpenID.Messages] either in HTTP GET or POST request.

The request parameters are the following:

access\_token

REQUIRED. The Access Token obtained from an OpenID Connect Authorization Request. This parameter MUST NOT be sent if the Access Token is sent in the HTTP Authorization header as described in Section 7.1 of [OAuth 2.0 (Hammer-Lahav, E., Ed., Recordon, D., and D. Hardt, “OAuth 2.0 Authorization Protocol,” September 2011.)](#OAuth.2.0) [OAuth.2.0]. Access tokens sent in the authorization header MUST be [Bearer Tokens (Jones, M., Hardt, D., and D. Recordon, “OAuth 2.0 Protocol: Bearer Tokens,” September 2011.)](#OAuth.2.0.Bearer) [OAuth.2.0.Bearer]. If the client is using the HTTP GET method, it SHOULD send the Access Token in the authorization header. The access\_token MAY alternatively be sent in the message body, as described in the [OAuth 2.0 (Hammer-Lahav, E., Ed., Recordon, D., and D. Hardt, “OAuth 2.0 Authorization Protocol,” September 2011.)](#OAuth.2.0) [OAuth.2.0] specification.

schema

REQUIRED. The schema in which the data is to be returned. The only defined value is openid. If the value of this parameter is omitted, or not openid, the response may be a proprietary schema to support backwards compatibility. A URL MAY be passed to define custom schemes not specified by short names. Custom scheme names and responses are out of scope for this specification.

id

This identifier is reserved for backwards compatibility. It MUST be ignored by the endpoint if the openid schema is used.

The following is a non-normative example. Line wraps are for display purpose only:

POST /userinfo HTTP/1.1

Host: server.example.com

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

access\_token=SlAV32hkKG

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

### 6.2.  UserInfo Response

Upon receipt of the UserInfo request, the UserInfo endpoint MUST return the JSON Serialization of the UserInfo response as in [OpenID Messages (Sakimura, N., Recordon, D., Bradley, J., de Medeiros, B., Jones, M., and E. Jay, “OpenID Connect Messages 1.0,” October 2011.)](#OpenID.Messages) [OpenID.Messages] in the HTTP response body. The content-type of the HTTP response MUST be set to application/json if the response body is a text JSON structure. If the JSON response is [JWS (Jones, M., Balfanz, D., Bradley, J., Goland, Y., Panzer, J., Sakimura, N., and P. Tarjan, “JSON Web Signatures,” April 2011.)](#JWS) [JWS] signed or [JWE (Jones, M., Bradley, J., and N. Sakimura, “JSON Web Encryption,” July 2011.)](#JWE) [JWE] encrypted, then the content-type MUST be set to application/jwt.

Following is a non-normative example of such response:

HTTP/1.1 200 OK

Content-Type: application/json

{

"name": "Jane Doe"

"given\_name": "Jane",

"family\_name": "Doe",

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

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

}

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

### 6.2.1.  UserInfo Error Response

When some error condition arises, the UserInfo endpoint returns the JSON serialized Error Response defined in section 3.3.3 of [OpenID Connect Messages 1.0 (Sakimura, N., Recordon, D., Bradley, J., de Medeiros, B., Jones, M., and E. Jay, “OpenID Connect Messages 1.0,” October 2011.)](#OpenID.Messages) [OpenID.Messages] in the entity body of the HTTP response using the application/json media type with HTTP response code 400.

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

HTTP/1.1 400 Bad Request

Content-Type: application/json

{

"error":"invalid\_request"

}

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

### 7.  Check ID Endpoint

The Check ID endpoint validates ID Tokens and returns the claims of an issued ID Token in JSON text format. This endpoint is used by clients that are not able to or do not wish to process ID Tokens. Clients MAY treat ID Tokens as opaque values and use the Check ID endpoint to validate and retrieve the claims associated with the ID Token in plain text JSON format.

Authorization Servers MUST require the use of a transport-layer security mechanism. The authorization server MUST support TLS 1.2 as described in [RFC 5246 (Dierks, T. and E. Rescorla, “The Transport Layer Security (TLS) Protocol Version 1.2,” August 2008.)](#RFC5246) [RFC5246] and MAY support other transport-layer mechanisms with equivalent security.

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

Clients MAY use the HTTP "GET" or "POST" method 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 as defined in [OpenID Connect Messages 1.0 (Sakimura, N., Recordon, D., Bradley, J., de Medeiros, B., Jones, M., and E. Jay, “OpenID Connect Messages 1.0,” October 2011.)](#OpenID.Messages) [OpenID.Messages]. If using the HTTP "POST" method, the request parameters are added to the HTTP request entity-body using "application/x-www-form-urlencoded" format.

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

### 7.1.  Check ID Requests

The client prepares a Check ID Endpoint Request to the Check ID Endpoint with the request parameters using the HTTP "GET" or "POST" method. The scheme used in the Authorization URL MUST be HTTPS.

The request parameters are as follows:

id\_token

REQUIRED. The ID Token obtained from an OpenID Connect authorization request.

The Following is a non-normative example of a Check ID request:

POST /check\_id HTTP/1.1

Host: server.example.com

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

id\_token=eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9.eyJpc3MiOiJodHRwOlwvXC9zZX

J2ZXIuZXhhbXBsZS5jb20iLCJ1c2VyX2lkIjoiMjQ4Mjg5NzYxMDAxIiwiYXVkIjoiaHR0cD

pcL1wvY2xpZW50LmV4YW1wbGUuY29tIiwiZXhwIjoxMzExMjgxOTcwfQ.eDesUD0vzDH3T1G

3liaTNOrfaeWYjuRCEPNXVtaazNQ

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

### 7.2.  Check ID Response

The Check ID Endpoint MUST return the JSON serialized claims associated with the ID Token as described in Check ID Response section of [OpenID Messages (Sakimura, N., Recordon, D., Bradley, J., de Medeiros, B., Jones, M., and E. Jay, “OpenID Connect Messages 1.0,” October 2011.)](#OpenID.Messages) [OpenID.Messages] in the HTTP response body. The content-type of the HTTP response MUST be set to application/json if the response body is a text JSON structure. If the JSON response is [JWS (Jones, M., Balfanz, D., Bradley, J., Goland, Y., Panzer, J., Sakimura, N., and P. Tarjan, “JSON Web Signatures,” April 2011.)](#JWS) [JWS] signed or [JWE (Jones, M., Bradley, J., and N. Sakimura, “JSON Web Encryption,” July 2011.)](#JWE) [JWE] encrypted, then the content-type MUST be set to application/jwt.

The following is a non-normative example of a response from the Check ID endpoint:

HTTP/1.1 200 OK

Content-Type: application/json

{

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

"user\_id": "248289761001",

"aud": "http://client.example.com",

"exp": 1311281970

}

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

### 7.2.1.  Check ID Error Response

When some error condition arises, the UserInfo endpoint returns the JSON serialized Error Response defined in section 3.4.3 of [OpenID Connect Messages 1.0 (Sakimura, N., Recordon, D., Bradley, J., de Medeiros, B., Jones, M., and E. Jay, “OpenID Connect Messages 1.0,” October 2011.)](#OpenID.Messages) [OpenID.Messages] in the entity body of the HTTP response using the application/json media type with HTTP response code 400.

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

HTTP/1.1 400 Bad Request

Content-Type: application/json

{

"error":"invalid\_id\_token"

}

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

### 8.  Session Management Endpoints

The Session Management endpoints are specified in the [OpenID Connect Session Management (Sakimura, N., Bradley, J., Jones, M., de Medeiros, B., Mortimore, C., and E. Jay, “OpenID Connect Session Management 1.0,” September 2011.)](#OpenID.Session) [OpenID.Session] specification.

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

### 9.  Discovery and Registration

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

However, if installations choose to support unanticipated interactions between Relying Parties and OpenID Providers that do not have pre-configured relationships, they SHOULD accomplish this by implementing the facilities defined in the [OpenID Connect Discovery 1.0 (Sakimura, N., Bradley, J., Jones, M., and E. Jay, “OpenID Connect Discovery 1.0,” September 2011.)](#OpenID.Discovery) [OpenID.Discovery] and [OpenID Connect Dynamic Client Registration 1.0 (Sakimura, N., Bradley, J., Ed., and M. Jones, “OpenID Connect Dynamic Client Registration 1.0,” September 2011.)](#OpenID.Registration) [OpenID.Registration] specifications.

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

### 10.  Security Considerations

This specification references the security considerations defined in [OpenID Connect Messages 1.0 (Sakimura, N., Recordon, D., Bradley, J., de Medeiros, B., Jones, M., and E. Jay, “OpenID Connect Messages 1.0,” October 2011.)](#OpenID.Messages) [OpenID.Messages].

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

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

### 10.1.  Implicit Grant Flow Threats

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

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

### 11.  IANA Considerations

This document makes no request of IANA.

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

### 12. Normative References

|  |  |
| --- | --- |
| **[JWE]** | Jones, M., Bradley, J., and N. Sakimura, “[JSON Web Encryption](http://tools.ietf.org/html/draft-jones-json-web-encryption),” July 2011. |
| **[JWS]** | Jones, M., Balfanz, D., Bradley, J., Goland, Y., Panzer, J., Sakimura, N., and P. Tarjan, “[JSON Web Signatures](http://tools.ietf.org/html/draft-jones-json-web-signature),” April 2011. |
| **[JWT]** | Jones, M., Balfanz, D., Bradley, J., Goland, Y., Panzer, J., Sakimura, N., and P. Tarjan, “[JSON Web Token](http://tools.ietf.org/html/draft-jones-json-web-token),” July 2011. |
| **[OAuth.2.0]** | Hammer-Lahav, E., Ed., Recordon, D., and D. Hardt, “[OAuth 2.0 Authorization Protocol](http://tools.ietf.org/html/draft-ietf-oauth-v2),” September 2011. |
| **[OAuth.2.0.Bearer]** | Jones, M., Hardt, D., and D. Recordon, “[OAuth 2.0 Protocol: Bearer Tokens](http://tools.ietf.org/html/draft-ietf-oauth-v2-bearer),” September 2011. |
| **[OpenID.Discovery]** | Sakimura, N., Bradley, J., Jones, M., and E. Jay, “[OpenID Connect Discovery 1.0](http://openid.net/specs/openid-connect-discovery-1_0.html),” September 2011. |
| **[OpenID.Messages]** | Sakimura, N., Recordon, D., Bradley, J., de Medeiros, B., Jones, M., and E. Jay, “[OpenID Connect Messages 1.0](http://openid.net/specs/openid-connect-messages-1_0.html),” October 2011. |
| **[OpenID.Registration]** | Sakimura, N., Bradley, J., Ed., and M. Jones, “[OpenID Connect Dynamic Client Registration 1.0](http://openid.net/specs/openid-connect-registration-1_0.html),” September 2011. |
| **[OpenID.Session]** | Sakimura, N., Bradley, J., Jones, M., de Medeiros, B., Mortimore, C., and E. Jay, “[OpenID Connect Session Management 1.0](http://openid.net/specs/openid-connect-session-1_0.html),” September 2011. |
| **[RESPONSE.TYPES]** | de Medeiros, B., [Scurtescu, M.](mailto:mscurtescu@google.com), and [P. Tarjan](mailto:pt@fb.com), “[OAuth 2.0 Multiple Response Type Encoding Practices](http://openid.net/specs/oauth-v2-multiple-response-types-1_0.html),” October 2011. |
| **[RFC2119]** | [Bradner, S.](mailto:sob@harvard.edu), “[Key words for use in RFCs to Indicate Requirement Levels](http://tools.ietf.org/html/rfc2119),” BCP 14, RFC 2119, March 1997 ([TXT](http://www.rfc-editor.org/rfc/rfc2119.txt), [HTML](http://xml.resource.org/public/rfc/html/rfc2119.html), [XML](http://xml.resource.org/public/rfc/xml/rfc2119.xml)). |
| **[RFC2616]** | [Fielding, R.](mailto:fielding@ics.uci.edu), [Gettys, J.](mailto:jg@w3.org), [Mogul, J.](mailto:mogul@wrl.dec.com), [Frystyk, H.](mailto:frystyk@w3.org), [Masinter, L.](mailto:masinter@parc.xerox.com), [Leach, P.](mailto:paulle@microsoft.com), and [T. Berners-Lee](mailto:timbl@w3.org), “[Hypertext Transfer Protocol -- HTTP/1.1](http://tools.ietf.org/html/rfc2616),” RFC 2616, June 1999 ([TXT](http://www.rfc-editor.org/rfc/rfc2616.txt), [PS](http://www.rfc-editor.org/rfc/rfc2616.ps), [PDF](http://www.rfc-editor.org/rfc/rfc2616.pdf), [HTML](http://xml.resource.org/public/rfc/html/rfc2616.html), [XML](http://xml.resource.org/public/rfc/xml/rfc2616.xml)). |
| **[RFC2617]** | [Franks, J.](mailto:john@math.nwu.edu), [Hallam-Baker, P.](mailto:pbaker@verisign.com), [Hostetler, J.](mailto:jeff@AbiSource.com), [Lawrence, S.](mailto:lawrence@agranat.com), [Leach, P.](mailto:paulle@microsoft.com), Luotonen, A., and [L. Stewart](mailto:stewart@OpenMarket.com), “[HTTP Authentication: Basic and Digest Access Authentication](http://tools.ietf.org/html/rfc2617),” RFC 2617, June 1999 ([TXT](http://www.rfc-editor.org/rfc/rfc2617.txt), [HTML](http://xml.resource.org/public/rfc/html/rfc2617.html), [XML](http://xml.resource.org/public/rfc/xml/rfc2617.xml)). |
| **[RFC5246]** | Dierks, T. and E. Rescorla, “[The Transport Layer Security (TLS) Protocol Version 1.2](http://tools.ietf.org/html/rfc5246),” RFC 5246, August 2008 ([TXT](http://www.rfc-editor.org/rfc/rfc5246.txt)). |

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

### Appendix A.  Footnotes

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

### A.1.  Example JWT Values

The JWT values used in the non-normative examples of this specification are only place holders for actual JWT values. The examples use "jwt\_header.jwt\_part2.jwt\_part3" as the place holder value. For an example of an actual JWT, refer to the [JWT (Jones, M., Balfanz, D., Bradley, J., Goland, Y., Panzer, J., Sakimura, N., and P. Tarjan, “JSON Web Token,” July 2011.)](#JWT) [JWT] specification.

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

### Appendix B.  Acknowledgements

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

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

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

George Fletcher (gffletch@aol.com), AOL

Hideki Nara (hideki.nara@gmail.com), Takt Communications

John Bradley (jbradely@mac.com), Protiviti Government Services

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

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

Ryo Itou (ritou@yahoo-corp.jp), Yahoo! Japan

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

### Appendix C.  Notices

Copyright (c) 2011 The OpenID Foundation.

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

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

### Appendix D.  Document History

[[ To be removed from the final specification ]]

-06

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

-05

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

-04

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

-03

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

-02

* Incorporates feedback from Nat Sakimura.

-01

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

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

### Authors' Addresses

|  |  |
| --- | --- |
|  | Nat Sakimura (editor) |
|  | Nomura Research Institute, Ltd. |
| **Email:** | [n-sakimura@nri.co.jp](mailto:n-sakimura@nri.co.jp) |
|  |  |
|  | John Bradley |
|  | Protiviti Government Services |
| **Email:** | [jbradley@mac.com](mailto:jbradley@mac.com) |
|  |  |
|  | Breno de Medeiros |
|  | Google |
| **Email:** | [breno@google.com](mailto:breno@google.com) |
|  |  |
|  | Michael B. Jones |
|  | Microsoft Corporation |
| **Email:** | [mbj@microsoft.com](mailto:mbj@microsoft.com) |
|  |  |
|  | Edmund Jay |
|  | MGI1 |
| **Email:** | [ejay@mgi1.com](mailto:ejay@mgi1.com) |