	N. Aillery
	C. Marais
	Orange
	October 5, 2016

OpenID Connect User Questioning API 1.0

draft-user-questioning-api-03

Abstract

This specification defines a specific endpoint used by a Client (i.e. Service Provider) in order to question an End-User and get his Statement (i.e. his answer).

Table of Contents

1. Introduction

This specification defines a specific endpoint used by a Client (i.e. Service Provider) in order to question an End-User and get his Statement (i.e. his answer).

This endpoint is specified as an OAuth 2.0-protected Resource Server accessible with an Access Token.

The way the Access Token has been obtained by the Client is out of scope of this specification.

Whether the End-User is currently using the Client or not is also out of scope of this specification.

The User Questioning API is an asynchronous API. There are 2 main ways to get the End-User's Statement: the first one requires some polling of the API and the second requires the Client to expose a callback endpoint.

1.1. Requirements Notation and Conventions

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

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

All uses of JSON Web Signature (JWS) [JWS] and JSON Web Encryption (JWE) [JWE] data structures in this specification utilize the JWS Compact Serialization or the JWE Compact Serialization; the JWS JSON Serialization and the JWE JSON Serialization are not used.

1.2. Terminology

This specification uses the terms "Access Token", "Resource Server", and "Client" defined by OAuth 2.0 [RFC6749], the terms "JSON Web Token (JWT)", "JWT Claims Set", and "Nested JWT" defined by JSON Web Token (JWT) [JWT], and the terms "Header Parameter" and "JOSE Header" defined by JSON Web Signature (JWS) [JWS]. This specification also defines the following terms:

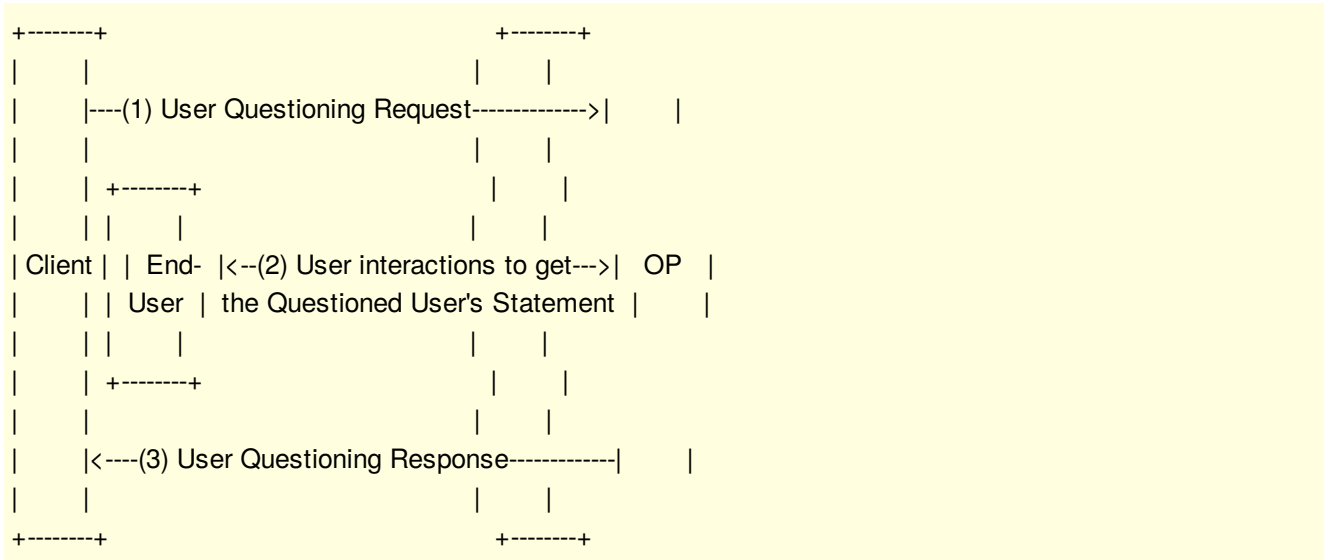
Questioned User
: End-User receiving the question and requested to give his statement.

2. Overview

The User Questioning protocol, in abstract, follows the following steps.

1. The Client sends a User Questioning Request to the OpenID Provider (OP).
2. The OP interacts with the Questioned User and obtains his Statement.
3. The OP responds to the Client with a User Questioning Response.

These steps are illustrated in the following diagram:



2.1. Example of use cases involving the User Questioning API

The following use cases are non-normative examples to illustrate the usage of the User Questioning API by a Client.

1. The Client can be a bank and the User Questioning API can be used to challenge the End-User when he wants to pay on Internet in order to secure the transaction. This is similar to 3D-Secure. The question could be: "Do you allow a payment of x euros to party y? (Yes)

(No)".

2. The Client can be a bank and the User Questioning API can be used to challenge the End-User when he wants to add a new payee for a bank transfer. The question could be: "Do you allow party y to be added to your payees? (I accept) (I refuse)".
3. The Client can be a drive-in food market and the User Questioning API can be used to ask the End-User if he accepts the exchange of one missing product by another. The question could be: "Do you agree to get product x instead of product y? (I agree) (I disagree)".
4. The Client can be a ticketing platform and the User Questioning API can be used to prevent transactions by bots. The question could be: "Do you confirm that you are currently booking a ticket? (I confirm) (I deny)".
5. The Client can be an airline company and the User Questioning API can be used to be sure that the End-User is notified of a delay. The question could be: "Your flight is postponed. Can you confirm that you are aware? (I read this)".
6. The Client can be a survey company and the User Questioning API can be used to get the End-User's choice. The question could be: "Which is your favorite brand? (brand_A) (brand_B) (brand_C)".

2.2. When to use the User Questioning API

The User Questioning API should be used when the following contraints must be fulfilled:

1. The Client needs to have a real-time interaction with an offline End-User.
2. The Client needs to have the End-User's statement on a given question.
3. The Client needs to be sure that the responding End-User has been authenticated before responding.
4. The Client needs to have a non-repudiable proof of the End-User's statement.
5. The Client needs the End-User's statement to enforce a policy (i.e. take a decision).

3. User Statement Token

The User Statement Token is a security token that contains Claims about the statement made by the Questioned User. The User Statement Token is represented as a JSON Web Token (JWT) [JWT].

The following Claims are used within the User Statement Token in all Questioning API flows.

question_id
: MANDATORY. The question_id is a unique identifier of the Question. The question_id value is a case sensitive string.

iss
: MANDATORY. 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.

aud
: MANDATORY. Audience(s) that this User Statement 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.

sub
: MANDATORY. Subject Identifier. A locally unique and never reassigned identifier within the Issuer for the End-User, which is intended to be consumed by the Client, e.g. g117YBtCZO3mAKPQKP8o. It MUST NOT exceed 255 ASCII [RFC20] characters in length. The sub value is a case-sensitive string.

user_id
: OPTIONAL. The user_id is present in the User Statement Token only if the user_id and user_id_type

were present in the User Questioning Request. The user_id is a unique identifier allowing to identify the Questioned User (e.g. Mobile phone, sub, ...). The user_id value is a case sensitive string.

user_id_type
: OPTIONAL. The user_id_type is present in the User Statement Token only if the user_id and user_id_type were present in the User Questioning Request. The user_id_type indicates the type of the End-User's identifier used for User Questioning. The user_id_type value is a case sensitive string.

question_displayed
: MANDATORY. The question_displayed is the question displayed to the Questioned User. If the question_to_display has not been displayed as is, the question_displayed MUST be the exact message displayed to Questioned User. The question_displayed value is a case sensitive string.

statement
: MANDATORY. Statement made by the User Questioning. The statement SHALL be the exact statement made by the Questioned User. The statement value is a case sensitive string.

statement_date
: MANDATORY. Date indicating when the End-User gave his Statement on the Question. The statement_date value is a JSON number representing the number of seconds from 1970-01-01T0:0:0Z as measured in UTC.

used_qcr
: MANDATORY. Questioning Context Class Reference. Questioning Context Class Reference value that identifies the Authentication Context Class used by the OP to authenticate the Questioned User before he made his statement. The used_qcr value is a case sensitive string.

used_qmr
: MANDATORY. Questioning Method Reference. This is the authentication method used by the OP to authenticate the Questioned User before he made his statement. The used_qmr value is a case sensitive string.

User Statement Tokens MUST be signed using JWS [JWS] and optionally both signed and then encrypted using JWS [JWS] and JWE [JWE] respectively, thereby providing authentication, integrity, non-repudiation, and optionally, confidentiality, per Section 9.1. If the User Statement Token is encrypted, it MUST be signed then encrypted, with the result being a Nested JWT, as defined in [JWT]. User Statement Tokens MUST NOT use none as the alg value.

User Statement Tokens SHOULD NOT use the JWS or JWE x5u, x5c, jku, or jwk Header Parameter fields. Instead, references to keys used are communicated in advance using Discovery and Registration parameters, per Section 8.

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

```
{
   "question_id":"984dcc7d3d4d4b0f9f8022e344f9",
   "iss":"https://server.example.com",
   "aud":"s8V3wm8IXMW6TboazXZX",
   "sub":"g117YBtCZO3mAKPQKP8o",
   "user_id"="+33612345678",
   "user_id_type"="msisdn",
   "question_displayed":"Do you allow ...?",
   "statement":"Yes",
   "statement_date":"1311282975",
   "used_qcr":"2",
   "used_qmr":"CLICK_OK"
}
```

Signing the User Statement Token with the RS256 algorithm results in this Object value (with line wraps within values for display purposes only):

```
eyJhbGciOiJSUzI1NiIsImtpZCI6ImsyYmRjIn0.ew0KICJpc3MiOiAiczZCaGRSa3
F0MyIsDQogImF1ZCI6ICJodHRwczovL3NlcnZlci5leGFtcGxlLmNvbSIsDQogInJl
Y2tuYW1lIjogbnVsbCwN    TODO    sIjogeyJlc3NlbnRpYWwiOiB0cnVlfS
JoZFJrcXQzIiwNCiAicmVkaXJlY3RfdXJpIjogImh0dHBzOi8vY2xpZW50LmV4YW1w
bGUub3JnL2NiIiwNCiAi            lbmlkIiwNCiAic3RhdGUiOiAiYWYwaW
Y2tuYW1lIjogbnVsbCwN    TODO    sIjogeyJlc3NlbnRpYWwiOiB0cnVlfS
bGUub3JnL2NiIiwNCiAi            lbmlkIiwNCiAic3RhdGUiOiAiYWYwaW
ogICAgICJnaXZlbl9uYW1lIjogeyJlc3NlbnRpYWwiOiB0cnVlfSwNCiAgICAgIm5p
Y2tuYW1lIjogbnVsbCwN    TODO    sIjogeyJlc3NlbnRpYWwiOiB0cnVlfS
wNCiAgICAgImVtYWlsX3ZlcmlmaWVkIjogeyJlc3NlbnRpYWwiOiB0cnVlfSwNCiAg
ICAgInBpY3R1cmUiOiBudWxsDQogICAgfSwNCiAgICJpZF90b2tlbiI6IA0KICAgIH
bGUub3JnL2NiIiwNCiAi            lbmlkIiwNCiAic3RhdGUiOiAiYWYwaW
Y2tuYW1lIjogbnVsbCwN    TODO    sIjogeyJlc3NlbnRpYWwiOiB0cnVlfS
bGUub3JnL2NiIiwNCiAi            lbmlkIiwNCiAic3RhdGUiOiAiYWYwaW
F6zTHm8CHERFMGQPhos-EJcaH4Hh-sMgk8ePrGhw_trPYs8KQxsn6R9Emo_wHwajyF
Y2tuYW1lIjogbnVsbCwN    TODO    sIjogeyJlc3NlbnRpYWwiOiB0cnVlfS
0GxFbuPbj96tVuj11pTnmFCUR6IEOXKYr7iGOCRB3btfJhM0_AKQUfqKnRlrRscc8K
ol-cSLWoYE9l5QqhoIlmzjT_cMnNIznW9E7CDyWXTsO70xnB4SkG6pXfLSjLLlxmPG
iyon_-Te111V8uE83IlzCYIb_NMXvtTIVc1jpspnTSD7xMbpL-2QgwUsAlMGzw
```

The following RSA public key, represented in JWK format, can be used to validate the Object signature in this example (with line wraps within values for display purposes only):

```
{
"kty":"RSA",
"kid":"k2bdc",
"n":"y9Lqv4fCp6Ei-u2-ZCKq83YvbFEk6JMs_pSj76eMkddWRuWX2aBKGHAtKlE5P
7_vn__PCKZWePt3vGkB6ePgzAFu08NmKemwE5bQI0e6kIChtt_6KzT5OaaXDF
I6qCLJmk51Cc4VYFaxgqevMncYrzaW_50mZ1yGSFIQzLYP8bijAHGVjdEFgZa
ZEN9lsn_GdWLaJpHrB3ROlS50E45wxrlg9xMncVb8qDPuXZarvghLL0HzOuYR
adBJVoWZowDNTpKpk2RklZ7QaBO7XDv3uR7s_sf2g-bAjSYxYUGsqkNA9b3xV
W53am_UZZ3tZbFTIh557JICWKHlWj5uzeJXaw",
"e":"AQAB"
}
```

4. Client registration

In order to use the User Questioning API, the Client should:

Have the right to access the User Questioning API:

MANDATORY. The right for a Client to access the User Questioning API is the right to use the User Questioning API's scope. This scope value is question.

Register its Client Notification Endpoint:

OPTIONAL. If the Client wants to be notified of the User Question Response (cf. Section 5.2), it MUST register its client_notification_endpoint. The client_notification_endpoint is a callback URL on the Client.

5. User Questioning flows

This document specifies two User Questioning flows:

Pulled-By-Client flow:

In this flow, after the User Questioning Request, the Client must call the OP in order to get the User Questioning Response. Refer to Section 5.1 for more details.

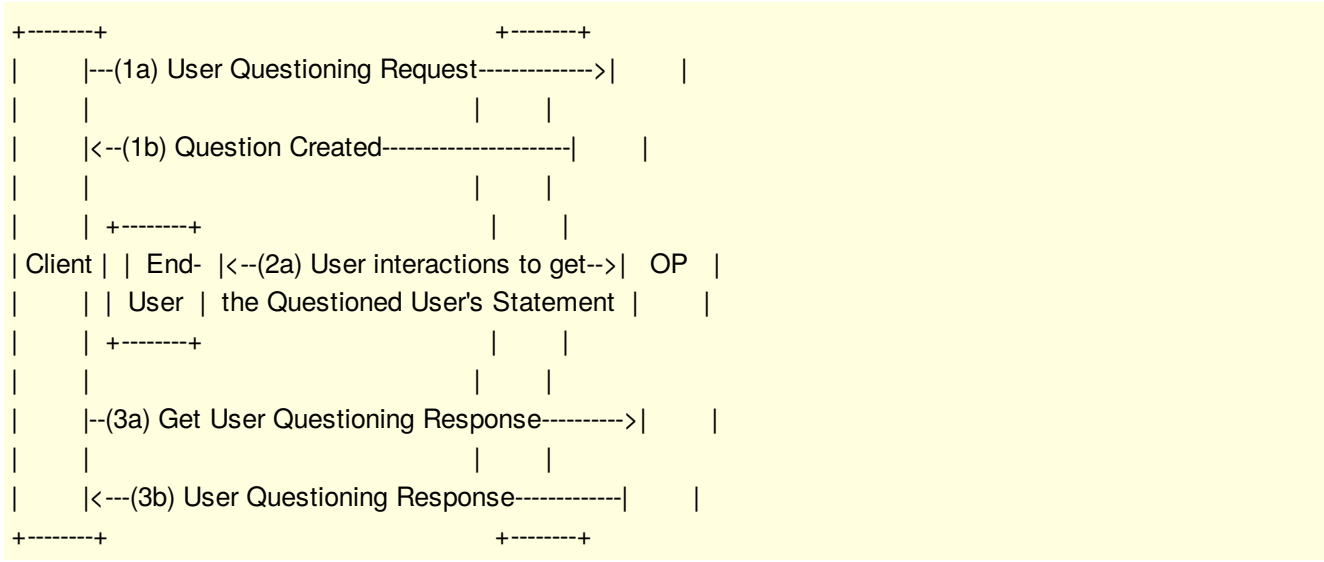
Pushed-To-Client flow:

In this flow, after the User Questioning Request, the OP calls the Client to deliver the User Questioning Response. Refer to Section 5.2 for more details.

The flow a Client will use is configured at registration. Refer to Section 4 for more details.

5.1. Pulled-By-Client Flow

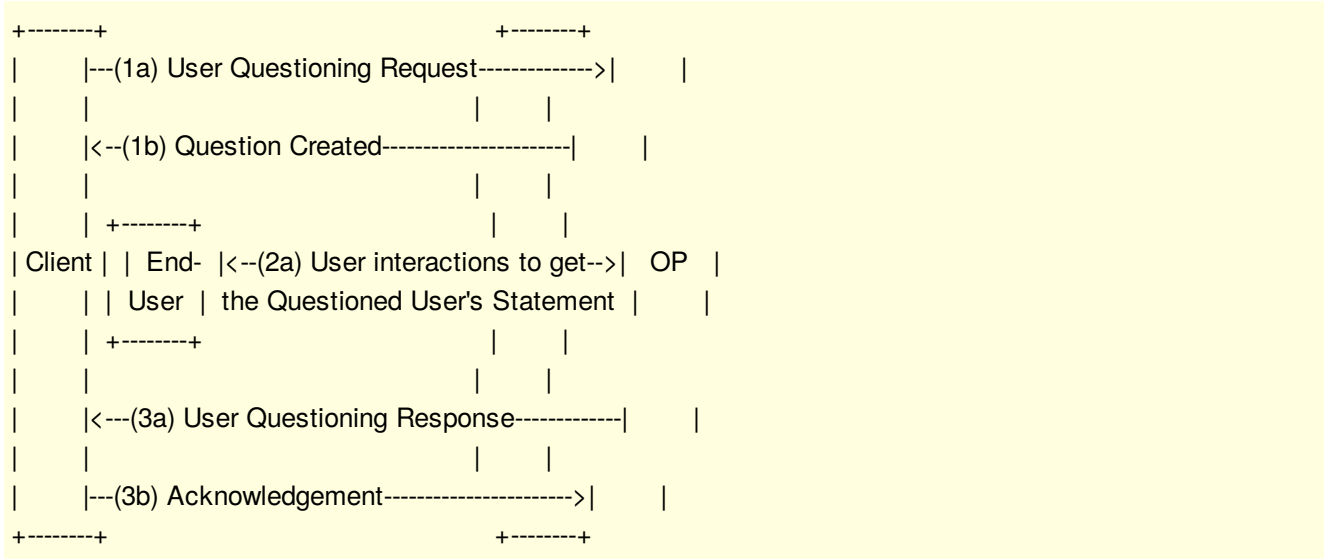
In this flow, the Client will poll the OP to get the User Statement Token.



About the "(2a) User interactions to get the Questioned User's Statement" step, note that the way the OP obtains the Questioned User's Statement is out of the scope of this specification.

5.2. Pushed-To-Client Flow

In this flow, the OP will send the User Statement Token on the client_notification_endpoint registered by the Client.



About the "(2a) User interactions to get the Questioned User's Statement" step, note that the way the OP obtains the Questioned User's Statement is out of the scope of this specification.

6. Endpoints

6.1. User Questioning Request Endpoint

6.1.1. Request with User Questioning Request

The Client sends the User Questioning Request using an HTTP POST Request.

The Client MUST have a valid Access Token for the scope question.

The HTTP POST request contains the following header:

Authorization
: MANDATORY. The Authorization value is an Access Token obtained from an OAuth Authorization request and used as a [RFC6750] Bearer Token. The Authorization value is a case sensitive string.

The User Questioning Request MUST contain a JSON structure in the body of the HTTP POST, with the following attributes:

client_notification_token
: MANDATORY if the Client uses the Pushed-To-Client Flow described in Section 5.2. IGNORED otherwise. The client_notification_token is an opaque token issued by the Client. The client_notification_token is Bearer Token used by the Client to authorize the OP when the OP sends the User Questioning Response to the Client Notification Endpoint. The client_notification_token value is a case sensitive string.

user_id
: FORBIDDEN if the Access Token is tied with an End-User, MANDATORY if the Access Token is not tied with an End-User. The user_id is a unique identifier allowing to identify the Questioned User (e.g. Mobile phone, sub, ...). The user_id value is a case sensitive string.

user_id_type
: FORBIDDEN if the Access Token is tied with an End-User, MANDATORY if the Access Token is not tied with an End-User. The user_id_type indicates the type of the End-User's identifier used for User Questioning. The user_id_type value is a case sensitive string.

question_to_display
: MANDATORY. The question_to_display is the question to be displayed to the Questioned User. The question_to_display SHALL be displayed with no modification to Questioned User. If some modifications occur, it MUST be due to restrictions imposed by the Questioning Method. The question_to_display value is a case sensitive string.

statements_to_display
: MANDATORY. The statements_to_display is the list of possible statements to be displayed to the Questioned User. The statements_to_display SHALL be displayed with no modification to Questioned User. The statements_to_display value is a JSON array of case sensitive strings.

wished_qcr
: MANDATORY. Questioning Context Class Reference. The wished_qcr is Questioning Context Class Reference value that identifies the Authentication Context Class to be used by the OP to authenticate the Questioned User before he made his statement. The wished_qcr value is a case sensitive string.

wished_qmr
: OPTIONAL. Questioning Method Reference. The wished_qmr is the authentication method to be used by the OP to authenticate the Questioned User before he made his statement. The wished_qmr value is a case sensitive string.

The parameters are included in the entity-body of the HTTP POST using the "application/json" media type as defined by [RFC4627]. The parameters are serialized into a JavaScript Object Notation (JSON) structure by

adding each parameter at the highest structure level. Parameter names and string values are included as JSON strings. Numerical values are included as JSON numbers. The order of parameters does not matter and can vary.

The following is a non-normative example of the JSON structure.

```
POST /questions HTTP/1.1
Host: server.client.com
Accept: application/json
Authorization: Bearer SIAV32hkKG
Content-Type: application/json
Content-Length: xxx

{
 "client_notification_token":"vO4n2DAeRrcWE0VAlo4I",
 "user_id":"+33612345678",
 "user_id_type":"msisdn",
 "question_to_display":"Do you allow ...?",
 "statements_to_display":["Yes","No"],
 "wished_qcr":"3",
 "wished_qmr":"PIN_OK"
}
```

6.1.1.1. user_id_type and user_id

The user_id is an identifier used to identify the Questioned User and to retrieve a reachability mean. If the user_id is a reachability identifier, it SHOULD be used by reachability mean.

user_id_type member can take the following values:

msisdn
: If the user_id_type value is msisdn, the user_id value is the mobile phone number corresponding to the Questioned User. E.164 [E.164] is RECOMMENDED as the format of this Claim, for example, +1 (425) 555-1212 or +5626872400.

email
: If the user_id_type value is email, the user_id value is the email adress corresponding to the Questioned User. Its value MUST conform to the RFC 5322 [RFC5322] addr-spec syntax.

sub
: If the user_id_type value is sub, the user_id value is the sub corresponding to the Questioned User for the requesting client_id.

6.1.2. Successful Response

A successful response to the User Questioning Request is a HTTP 201 Created response.

The HTTP 201 Created response contains the following headers:

Location
: MANDATORY. The Location value is a unique polling URL for the Question. The Location value is a URL on the User Questioning Polling Endpoint.

Question_id
: MANDATORY. The Question_id contains the question_id value. The Question_id value is a case sensitive string.

The following is a non-normative example of the JSON structure.

```
HTTP/1.1 201 Created
Location: https://server.example.com/questions_polling/984dcc7d3d4d4b0f9f8022e344f9
question_id: 984dcc7d3d4d4b0f9f8022e344f9
```

6.1.3. Error Response

The Client receives the error in a HTTP 400 Bad Request.

The response body contains an error_info JSON structure as defined in Section 7.

The error_info JSON structure can contain the following error codes, defined in Section 7.1:

- invalid_request
- no_suitable_method
- unknown_user
- unreachable_user

The following is a non-normative example of error in a HTTP 400 Bad Request. This example can occur in the Pulled-By-Client flow (cf. Section 5.1) and Pushed-To-Client flow (cf. Section 5.2).

```
HTTP/1.1 400 Bad Request
Content-Type: application/json
Content-Length: xxx

{
   "error_info":
   {
      "error_code":"invalid_request",
      "error_description":"user_id is FORBIDDEN
     if the Access Token is tied with an End-User",
      "error_uri":"https://server.example.com/errors/invalid_request"
   }
}
```

6.2. User Questioning Polling Endpoint

In order to detect that User Questioning Response is ready, the Client MUST request the User Questioning Polling Endpoint. If User Questioning Response is ready, it will be returned in the response. Else, the OP responds with HTTP 304 Not Modified.

A Client configured with a client_notification_endpoint MUST NOT sent a request to the User Questioning Polling Endpoint.

6.2.1. Polling Request

The Client get the User Questioning Response using HTTP GET.

The HTTP GET request contains the following headers:

Authorization
: MANDATORY. The Authorization value is an Access Token obtained from an OAuth Authorization request and used as a [RFC6750] Bearer Token. The Authorization value is a case sensitive string.

Client_timeout
: The Client_timeout indicates how much time, in seconds, the Client will wait for a HTTP response. The Client_timeout value is a number.

The following is a non-normative example.

```
GET /questions_polling/984dcc7d3d4d4b0f9f8022e344f9 HTTP/1.1
Host: server.example.com
Accept: application/json
Authorization: Bearer SlAV32hkKG
Client_timeout: 10
```

6.2.2. Pending Response

If the User Questioning Response is not ready, the Client will get a HTTP 304 Not Modified.

The following is a non-normative example.

```
HTTP/1.1 304 Not Modified
```

6.2.3. Successful Response with User Questioning Response

The Client receives the successful User Questioning Response in a HTTP 200 OK response.

The successful User Questioning Response is a JSON object, with the following attributes.

question_id
: MANDATORY. The question_id is a unique identifier of the Question. The question_id value is a case sensitive string.

user_statement_token
: MANDATORY. The user_statement_token is a User Statement Token as described in Section 3

The parameters are included in the entity-body of the HTTP 200 OK using the "application/json" media type as defined by [RFC4627]. The parameters are serialized into a JavaScript Object Notation (JSON) structure by adding each parameter at the highest structure level. Parameter names and string values are included as JSON strings. Numerical values are included as JSON numbers. The order of parameters does not matter and can vary.

The following is a non-normative example.

```
HTTP/1.1 200 OK
Content-Type: application/json
Content-Length: xxx

{
   "question_id":"984dcc7d3d4d4b0f9f8022e344f9",
   "user_statement_token":"eyJhbGciOiJSUzI1NiIsImtpZCI6IjFlOWdkazci
 fQ.ewogImlzci8vc2VydmVyLmV4YW1wbGUuY29tIiwKICJzdWIiOiAiMjQ4Mjg5
 NzYxMDAxIiwKICJhdWQiOiAiczZCaGRSa3F0MyIsCiAibm9uY2UiOiAibi0wUzZ
 fV3pBMk1qIiwKICJleHAiO                ImlhdCI6IDEzMTEyODA5Nz
 AKfQ.ggW8hZ1EuVLuxNuuI       TODO       6jgdqrOOF4daGU96Sr_P6q
 Jp6IcmD3HP99Obi1PRs-cw                L7F09JdijmBqkvPeB2T9CJ
 NqeGpe-gccMg4vfKjkM8FcGvnzZUN4_KSP0aAp1tOJ1zZwgjxqGByKHiOtX7Tpd
 QyHE5lcMiKPXfEIQILVq0pc_E2DzL7emopWoaoZTF_m0_N0YzFC6g6EJbOEoRoS
 K5hoDalrcvRYLSrQAZZKflyuVCyixEoV9GfNQC3_osjzw2PAithfubEEBLuVVk4
 XUVrWOLrLl0nx7RkKU8NXNHq-rvKMzqg"
}
```

6.2.4. Error Response

The Client receives the error in a HTTP 400 Bad Request.

The response body contains an error_info JSON structure as defined in Section 7.

The error_info JSON structure can contain the following error codes, defined in Section 7.1:

- duplicate_requests
- forbidden
- high_rate_client
- high_rate_question
- invalid_question_id
- invalid_request
- timeout
- user_refused_to_answer

The following is a non-normative example of error in a HTTP 400 Bad Request. This example can occur in the Pulled-By-Client flow (cf. Section 5.1) and Pushed-To-Client flow (cf. Section 5.2).

```
HTTP/1.1 400 Bad Request
Content-Type: application/json
Content-Length: xxx

{
   "error_info":
   {
      "error_code":"duplicate_requests",
      "error_description":"A newer request was sent for this question_id",
      "error_uri":"https://server.example.com/errors/invalid_request"
   }
}
```

6.3. Client Notification Endpoint

The OP sends the User Questioning Response to the Client using HTTP POST. The User Questioning Response is a JSON object. The User Questioning Response contains a status attribute that indicates if the User Questioning Response is successful or not.

6.3.1. Successful Response with User Questioning Response

The OP sends the User Questioning Response to the Client using HTTP POST.

The HTTP POST request contains the following header:

Authorization
: MANDATORY. The Authorization value is the client_notification_token specified in the User Questioning Request and used as a [RFC6750] Bearer Token. The Authorization value is a case sensitive string.

A successful User Questioning Response is a JSON object, with the following attributes.

question_id
: MANDATORY. The question_id is a unique identifier of the Question. The question_id value is a case sensitive string.

status
: MANDATORY. Status of the User Questioning Response. When the User Questioning Response is successful, the value of status is ok.

user_statement_token
: MANDATORY. The user_statement_token is a User Statement Token as described in Section 3.

The following is a non-normative example.

```
POST /notification HTTP/1.1
Host: client.example.com
Content-Type: application/json
Authorization: Bearer vO4n2DAeRrcWE0VAlo4I
Content-Length: xxx

{
   "question_id":"984dcc7d3d4d4b0f9f8022e344f9",
   "status":"ok",
   "user_statement_token":"eyJhbGciOiJSUzI1NiIsImtpZCI6IjFlOWdkazci
fQ.ewogImlzci8vc2VydmVyLmV4YW1wbGUuY29tIiwKICJzdWIiOiAiMjQ4Mjg5
NzYxMDAxIiwKICJhdWQiOiAiczZCaGRSa3F0MyIsCiAibm9uY2UiOiAibi0wUzZ
fV3pBMk1qIiwKICJleHAiO                    ImlhdCI6IDEzMTEyODA5Nz
AKfQ.ggW8hZ1EuVLuxNuuI       TODO        6jgdqrOOF4daGU96Sr_P6q
Jp6IcmD3HP99Obi1PRs-cw                   L7F09JdijmBqkvPeB2T9CJ
NqeGpe-gccMg4vfKjkM8FcGvnzZUN4_KSP0aAp1tOJ1zZwgjxqGByKHiOtX7Tpd
QyHE5lcMiKPXfEIQILVq0pc_E2DzL7emopWoaoZTF_m0_N0YzFC6g6EJbOEoRoS
K5hoDalrcvRYLSrQAZZKflyuVCyixEoV9GfNQC3_osjzw2PAithfubEEBLuVVk4
XUVrWOLrLl0nx7RkKU8NXNHq-rvKMzqg"
}
```

6.3.2. Error Response

The OP sends the erroneous User Questioning Response to the Client using HTTP POST.

The HTTP POST request contains the following header:

Authorization
: MANDATORY. The Authorization value is the client_notification_token specified in the User Questioning Request and used as a [RFC6750] Bearer Token. The Authorization value is a case sensitive string.

An erroneous User Questioning Response is a JSON object, with the following attributes.

question_id
: MANDATORY. The question_id is a unique identifier of the Question. The question_id value is a case sensitive string.

status
: MANDATORY. Status of the User Questioning Response. An erroneous User Questioning Response contains a status attribute with the value error.

error_info
: MANDATORY. An erroneous User Questioning Response contains a error_info JSON structure as defined in Section 7.

The error_info JSON structure can contain the following error codes, defined in Section 7.1:

- timeout
- user_refused_to_answer

The following is a non-normative example.

```
POST /notification HTTP/1.1
Host: client.example.com
Content-Type: application/json
Authorization: Bearer vO4n2DAeRrcWE0VAlo4I
Content-Length: xxx

{
   "question_id":"984dcc7d3d4d4b0f9f8022e344f9",
   "status":"error",
   "error_info":
   {
      "error_code":"unknown_user",
      "error_description":"The user is unknown",
      "error_uri":"https://server.example.com/errors/unknown_user"
   }
}
```

6.3.3. Acknowledgement

The acknowledgement MUST be a HTTP 200 OK. This HTTP response SHOULD be ignored by the OP.

The following is a non-normative example.

```
HTTP/1.1 200 OK
```

7. Errors

error_info is a JSON object which contains the following properties:

error_code
: REQUIRED. Code representing the error. See Section 6 for possible values.

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

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

The parameters are serialized into a JavaScript Object Notation (JSON) structure by adding each parameter at the highest structure level. Parameter names and string values are included as JSON strings. Numerical values are included as JSON numbers. The order of parameters does not matter and can vary.

The following is a non-normative example.

```
{
 "error_code":"unknown_user",
 "error_description":"The user is unknown",
 "error_uri":"https://server.example.com/errors/unknown_user"
}
```

7.1. Error Codes

The error_info JSON structure can contain the following error codes:

duplicate_requests
: The Client sent simultaneous requests to the User Questioning Polling Endpoint for the same question_id. This error is responded to oldest requests. The last request is processed normally.

forbidden
: The Client sent a request to the User Questioning Polling Endpoint whereas it is configured with a client_notification_endpoint.

high_rate_client
: The Client sent requests at a too high rate, amongst all question_id. Information about the allowed and recommended rates can be included in the error_description.

high_rate_question
: The Client sent requests at a too high rate for a given question_id. Information about the allowed and recommended rates can be included in the error_description.

invalid_question_id
: The Client sent a request to the User Questioning Polling Endpoint for a question_id that does not exist or is not valid for the requesting Client.

invalid_request
: The User Questioning Request is not valid.

no_suitable_method
: There is no Questioning Method suitable with the User Questioning Request.

timeout
: The Questioned User did not answer in the allowed period of time.

unknown_user
: The Questioned User mentioned in the user_id attribute of the User Questioning Request is unknown.

unreachable_user
: The Questioned User mentioned in User Questioning Request (either in the Access Token or in the user_id attribute) is unreachable.

user_refused_to_answer
: The Questioned User refused to give a statement to the question.

8. Signatures and Encryption

cf. OpenID Connect - chapter 10

9. Security Considerations

TBD

9.1. Signing and Encryption Order

cf. OpenID Connect - chapter 16.14

10. Privacy Considerations

TBD

11. IANA Considerations

This document makes no requests of IANA.

12. Normative References

[E.164] International Telecommunication Union, "E.164: The international public telecommunication numbering plan", 2010.

[JWE]	Jones, M., Rescorla, E. and J. Hildebrand, "JSON Web Encryption (JWE)", RFC 7516, DOI 10.17487/RFC7516, May 2015.
[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)", Internet-Draft draft-ietf-oauth-json-web-token, May 2013.
[OpenID.Core]	Sakimura, N., Bradley, J., Jones, M., de Medeiros, B., Mortimore, C. and E. Jay, "OpenID Connect Standard 1.0", December 2013.
[OpenID.Discovery]	Sakimura, N., Bradley, J., Jones, M. and E. Jay, "OpenID Connect Discovery 1.0", September 2014.
[OpenID.Registration]	Sakimura, N., Bradley, J. and M. Jones, "OpenID Connect Dynamic Client Registration 1.0", December 2013.
[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.
[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., "Internet Message Format", RFC 5322, DOI 10.17487/RFC5322, October 2008.
[RFC6749]	Hardt, D., "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.

Appendix A. Acknowledgements

The following have contributed to the development of this specification.

Appendix B. Notices

disclaim any) warranties (express, implied, or otherwise), including implied warranties of merchantability, non-infringement, fitness for a particular purpose, or title, related to this specification, and the entire risk as to implementing this specification is assumed by the implementer. The OpenID Intellectual Property Rights policy requires contributors to offer a patent promise not to assert certain patent claims against other contributors and against implementers. The OpenID Foundation invites any interested party to bring to its attention any copyrights, patents, patent applications, or other proprietary rights that may cover technology that may be required to practice this specification.

Appendix C. Document History

[[To be removed from the final specification]]

-01

- Initial draft
- Added OIDF Standard Notice

Authors' Addresses

Nicolas Aillery
Orange
EMail: nicolas.aillery@orange.com

Charles Marais
Orange
EMail: charles.marais@orange.com