N. Aillery
C. Marais
Orange
November 2, 2016

OpenID Connect User Questioning API 1.0
draft-user-questioning-api-04

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.

The Client can use the endpoint defined in this specification whether the End-User is currently using the Client or not.

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

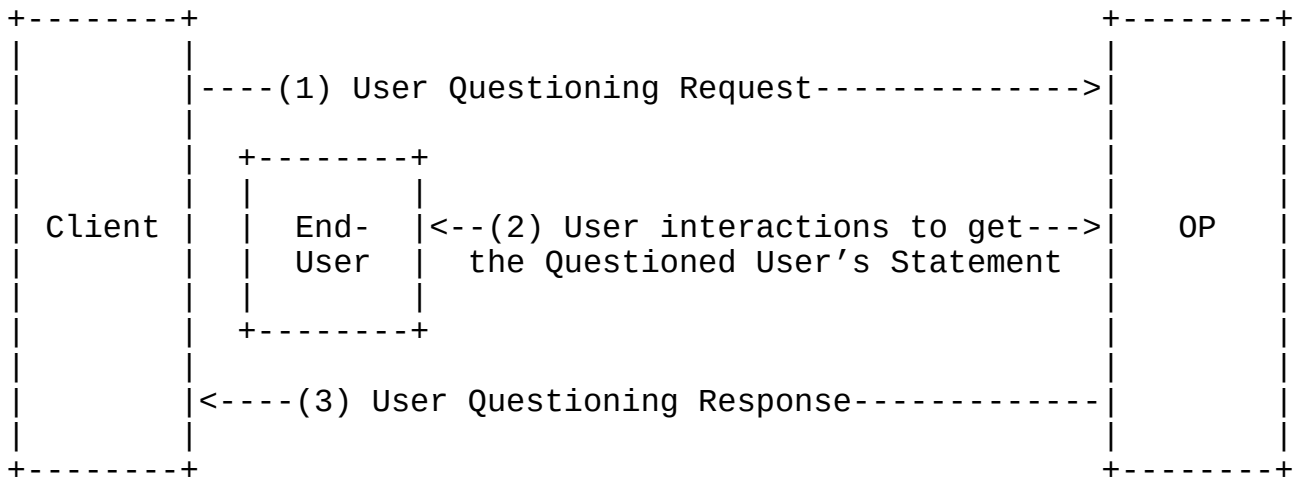
The User Questioning protocol, in abstract, follows the following steps.

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 End-User that may not be currently using the Client.

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. This identifier is created by the OP. The "question_id" value is a case sensitive string.

iss MANDATORY. Issuer Identifier for the Issuer of the response. "iss" is defined in chapter 2 of [OpenID.Core]. The issuer is the OP.

sub MANDATORY. Subject Identifier. "sub" is defined in chapter 2 of [OpenID.Core].

aud MANDATORY. Audience(s) that this User Statement Token is intended for. "aud" is defined in chapter 2 of [OpenID.Core].

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, ...). When present, this is the identifier actually used to reach the Questioned User. 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_amr MANDATORY. Authentication Method References. These are the authentication methods used by the OP to authenticate the Questioned User before he made his statement. "Authentication Method References" are defined in chapter 2 of [OpenID.Core], where they are named "amr". The "used_amr" value is an array of case sensitive strings.

used_acr MANDATORY. Authentication Context Class Reference. Authentication Context Class Reference value that identifies the Authentication Context Class used by the OP to authenticate the Questioned User before he made his statement. "Authentication Context Class Reference" is defined in chapter 2 of [OpenID.Core], where it is named "acr". The "used_acr" value is a case sensitive string.

User Statement Tokens MUST be signed by the OP using a digital signature as defined in JWS [JWS] providing authentication, integrity and non-repudiation. The signature MUST be a digital signature. Message Authentication Codes (MACs) are FORBIDDEN. User Statement Tokens CAN optionally be both signed and then encrypted using JWS [JWS] and JWE [JWE] respectively, thereby providing authentication, integrity, non-repudiation, and confidentiality, per Section 10.5. 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", "HS256", "HS384" and "HS512" 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 10.4.

The signature MUST be verified by the Client.

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_acr":"2",
    "used_amr":"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):

```
eyJhbGciOiJSUzI1NiIsInR5cCI6IkpXVCIsImtpZCI6ImsyYmRjIn0.eyJxdWVzdG
lvbl9pZCI6Ijk4NGRjYzdkM2Q0ZDRiMGY5ZjgwMjJlMzQ0ZjkiLCJpc3MiOiJodHRw
czovL3NlcnZlci5leGFtcGxlLmNvbSIsImF1ZCI6InM4VjN3bThJWE1XNlRib2F6WF
pYIiwic3ViIjoiZzExN1lCdENaTzNtQUtQUUtQOG8iLCJ1c2VyX2lkIjoiKzMzNjEy
MzQ1Njc4IiwidXNlcl9pZF90eXBlIjoibXNpc2RuIiwicXVlc3Rpb25fZGlzcGxheW
VkIjoiRG8geW91IGFsbG93IC4uLj8iLCJzdGF0ZW1lbnQiOiJZZXMiLCJzdGF0ZW1l
bnRfZGF0ZSI6MTMxMTI4Mjk3NSwidXNlZF9hY3IiOiIyIiwidXNlZF9hbXIiOiJDTE
lDS19PSyJ9.vj5DOJAZPE3-t0GmBMvaoKE8QB1VQYz94-nXbP5bUkwE9BYwixbU0sF
xhmabRbEZgrEmhJp1LFNw6OMPDlJsUHPgnqo_NbrxcmRYxpz07dKMLcOk8SbGcyL3e
RPG-wiT0PCbxOFw9qjrjFJHCTFdkGfTnuyf0cOia9oEInTN49dPt_WkL1JHgs9SkjA
UUlY9zcQc9KMjwjcXGN2Z6ypNHu0jZXdplm3CIaxc7vSa0eHZrimCg4PSwAhuFoe7S
bquLcVU56Bl4PvBBv6jGBtHW7O2Mo-ZhdoLeHqBOPWe707Wd0spbV1aekNdw8LB_ID
-TsOsWTQn7S8NfsFkk1dZXw
```

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. Managing user_id

The "user_id" is an identifier used to identify the Questioned User and to retrieve a reachability mean.

The "user_id" is determined from the User Questioning Request, either from the "user_id" attribute or from the Access Token. If the "user_id" is present in both the User Questioning Request and the Access Token, an error is raised. If the "user_id" is absent of both the User Questioning Request and the Access Token, an error is also raised.

If the "user_id" is a reachability identifier, it SHOULD be used by reachability mean. Otherwise, the OP must use the "user_id" to determine a suitable reachability identifier.

"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".

Other "user_id_type" can be specified for specific use cases.

5. Client registration

In order to use the User Questioning API, the Client MUST have the right to access the User Questioning API. 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".

If the the Client wants to be notified of the User Question Response (cf. Section 6.2), it MUST register its client_notification_endpoint. The client_notification_endpoint is a callback URL on the Client. If the client does not use the client notification, it MUST obtain the result by using the user questioning polling endpoint. Both mechanisms are mutual exclusive, i.e. a particular client_id can only be used with one of these mechanisms.

6. User Questioning flows

This document specifies two User Questioning flows:

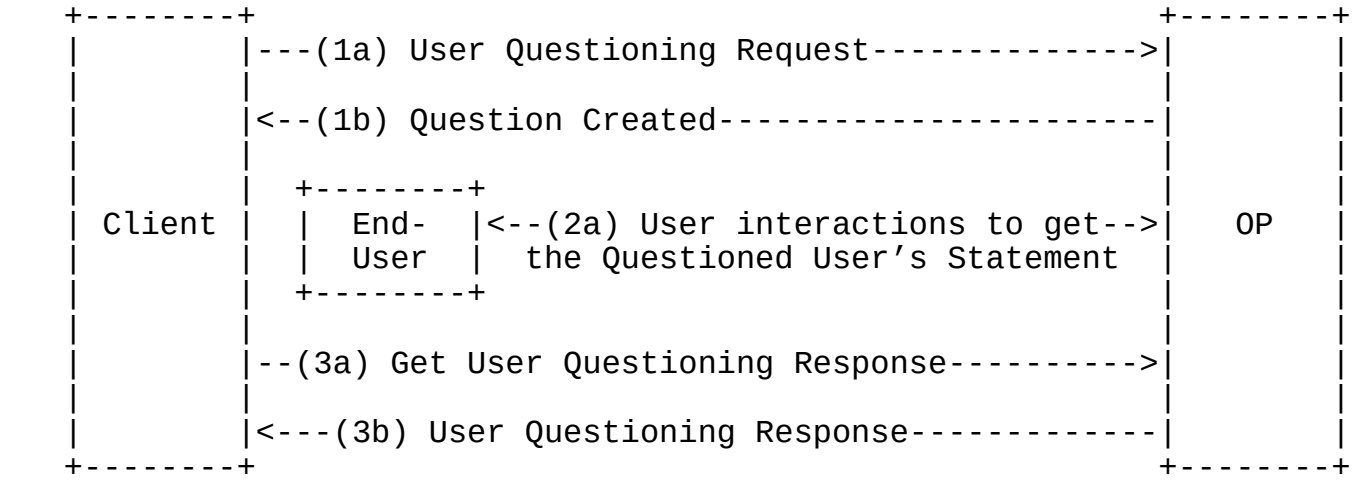
Pulled-By-Client flow: In this flow, after the User Questioning Request, the Client must call the OP in order to get the User Questioning Response. Refer to Section 6.1 for more details.

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 6.2 for more details.

The flow a Client will use is configured at registration. Refer to Section 5 for more details.

6.1. Pulled-By-Client Flow

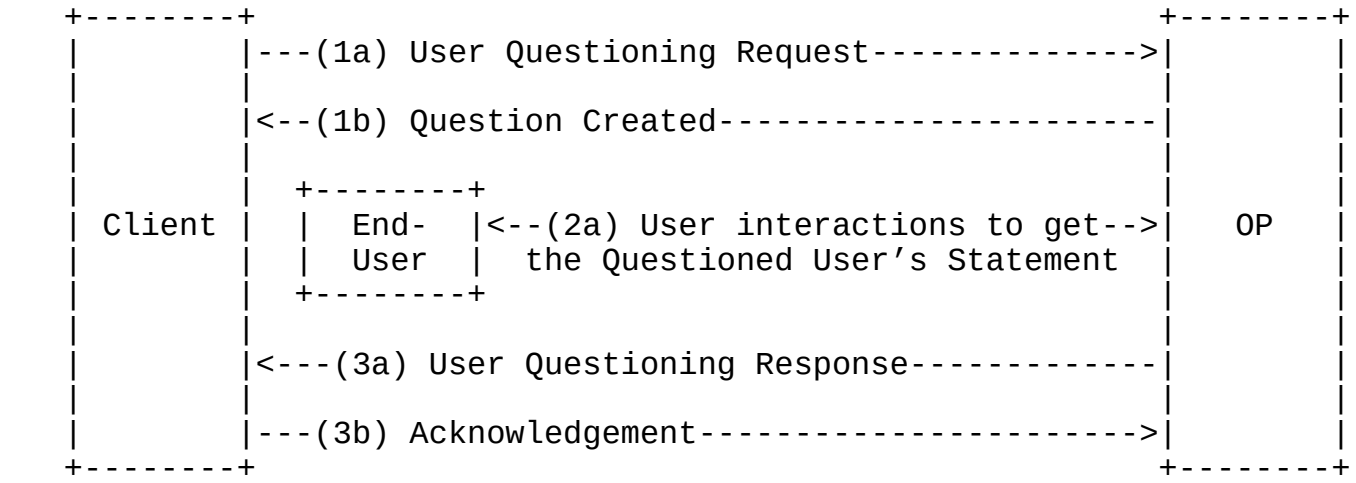
In this flow, the Client will poll the OP to get the User Statement Token.



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.

6.2. Pushed-To-Client Flow

In this flow, the OP will send the User Statement Token to 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.

7. Endpoints

7.1. User Questioning Request Endpoint

Communication with the User Questioning Request Endpoint MUST utilize TLS. See Section 16.17 in [OpenID.Core] for more information on using TLS.

7.1.1. 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 MUST contain 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 6.2. FORBIDDEN 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 amongst the values defined in Section 4.

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_amr OPTIONAL. Authentication Method Reference. The "wished_amr" is the authentication methods to be used by the OP to authenticate the Questioned User before he made his statement. "Authentication Method References" are defined in chapter 2 of [OpenID.Core], where they are named "amr". The "wished_amr" value is an array of case sensitive strings.

wished_acr MANDATORY. Authentication Context Class Reference. The "wished_acr" is Authentication 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. "Authentication Context Class Reference" is defined in chapter 2 of [OpenID.Core], where it is named "acr". The "wished_acr" 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 SlAV32hkKG
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_acr":"3",
        "wished_amr":"PIN_OK"
}
```

7.1.2. Successful Response

A successful response to the User Questioning Request is a HTTP 200 OK response.

The HTTP 200 OK response contains the following headers:

Location OPTIONAL. The "Location" value is a unique polling URL for the Question. The "Location" value is a URL on the User Questioning Polling Endpoint. When the Client is configured to use the Pushed-to-Client flow, this URL MUST be ignored by the Client. When the Client is configured to use the Pull-by-Client flow, if present, this URL MUST be used by the Client. If absent, the Client MUST use the "question_id" contained in the header "Question_id" and use it as a parameter to the "user_questioning_polling_endpoint" retrieved using [OpenID.Discovery].

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/1.1 200 OK
Location: https://server.example.com/questions_polling/984dcc7d3d4d4b0f9f802
2e344f9
Question_id: 984dcc7d3d4d4b0f9f8022e344f9
```

7.1.3. 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 8.

The "error_info" JSON structure can contain the following error codes, defined in Section 8.1:

- 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 6.1) and Pushed-To-Client flow (cf. Section 6.2).

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

{
        "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"
}
```

7.2. 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.

Communication with the User Questioning Polling Endpoint MUST utilize TLS. See Section 16.17 in [OpenID.Core] for more information on using TLS.

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

7.2.1. Polling Request

The Client get the User Questioning Response using HTTP GET.

The HTTP GET request MUST contain 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
```

7.2.2. 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
```

7.2.3. 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 Client SHOULD check that the "question_id" in the User Questioning Response is the same as the "question_id" in the "user_statement_token".

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":"eyJhbGciOiJSUzI1NiIsInR5cCI6IkpXVCIsImt
        pZCI6ImsyYmRjIn0.eyJxdWVzdGlvbl9pZCI6Ijk4NGRjYzdkM2Q0ZDRiMGY5Zj
        gwMjJlMzQ0ZjkiLCJpc3MiOiJodHRwczovL3NlcnZlci5leGFtcGxlLmNvbSIsI
        mF1ZCI6InM4VjN3bThJWE1XNlRib2F6WFpYIiwic3ViIjoiZzExN1lCdENaTzNt
        QUtQUUtQOG8iLCJ1c2VyX2lkIjoiKzMzNjEyMzQ1Njc4IiwidXNlcl9pZF90eXB
        lIjoibXNpc2RuIiwicXVlc3Rpb25fZGlzcGxheWVkIjoiRG8geW91IGFsbG93IC
        4uLj8iLCJzdGF0ZW1lbnQiOiJZZXMiLCJzdGF0ZW1lbnRfZGF0ZSI6MTMxMTI4M
        jk3NSwidXNlZF9hY3IiOiIyIiwidXNlZF9hbXIiOiJDTElDS19PSyJ9.vj5DOJA
        ZPE3-t0GmBMvaoKE8QB1VQYz94-nXbP5bUkwE9BYwixbU0sFxhmabRbEZgrEmhJ
        p1LFNw6OMPDlJsUHPgnqo_NbrxcmRYxpz07dKMLcOk8SbGcyL3eRPG-wiT0PCbx
        OFw9qjrjFJHCTFdkGfTnuyf0cOia9oEInTN49dPt_WkL1JHgs9SkjAUUlY9zcQc
        9KMjwjcXGN2Z6ypNHu0jZXdplm3CIaxc7vSa0eHZrimCg4PSwAhuFoe7SbquLcV
        U56Bl4PvBBv6jGBtHW7O2Mo-ZhdoLeHqBOPWe707Wd0spbV1aekNdw8LB_ID-Ts
        OsWTQn7S8NfsFkk1dZXw"
}
```

7.2.4. 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 8.

The "error_info" JSON structure can contain the following error codes, defined in Section 8.1:

- 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 6.1) and Pushed-To-Client flow (cf. Section 6.2).

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

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

7.3. 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.

Communication with the Client Notification Endpoint MUST utilize TLS. See Section 16.17 in [OpenID.Core] for more information on using TLS.

7.3.1. Successful Response with User Questioning Response

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

The HTTP POST request MUST contain 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 Client SHOULD check that the "question_id" in the User Questioning Response is the same as the "question_id" in the "user_statement_token".

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":"eyJhbGciOiJSUzI1NiIsInR5cCI6IkpXVCIsImt
        pZCI6ImsyYmRjIn0.eyJxdWVzdGlvbl9pZCI6Ijk4NGRjYzdkM2Q0ZDRiMGY5Zj
        gwMjJlMzQ0ZjkiLCJpc3MiOiJodHRwczovL3NlcnZlci5leGFtcGxlLmNvbSIsI
        mF1ZCI6InM4VjN3bThJWE1XNlRib2F6WFpYIiwic3ViIjoiZzExN1lCdENaTzNt
        QUtQUUtQOG8iLCJ1c2VyX2lkIjoiKzMzNjEyMzQ1Njc4IiwidXNlcl9pZF90eXB
        lIjoibXNpc2RuIiwicXVlc3Rpb25fZGlzcGxheWVkIjoiRG8geW91IGFsbG93IC
        4uLj8iLCJzdGF0ZW1lbnQiOiJZZXMiLCJzdGF0ZW1lbnRfZGF0ZSI6MTMxMTI4M
        jk3NSwidXNlZF9hY3IiOiIyIiwidXNlZF9hbXIiOiJDTElDS19PSyJ9.vj5DOJA
        ZPE3-t0GmBMvaoKE8QB1VQYz94-nXbP5bUkwE9BYwixbU0sFxhmabRbEZgrEmhJ
        p1LFNw6OMPDlJsUHPgnqo_NbrxcmRYxpz07dKMLcOk8SbGcyL3eRPG-wiT0PCbx
        OFw9qjrjFJHCTFdkGfTnuyf0cOia9oEInTN49dPt_WkL1JHgs9SkjAUUlY9zcQc
        9KMjwjcXGN2Z6ypNHu0jZXdplm3CIaxc7vSa0eHZrimCg4PSwAhuFoe7SbquLcV
        U56Bl4PvBBv6jGBtHW7O2Mo-ZhdoLeHqBOPWe707Wd0spbV1aekNdw8LB_ID-Ts
        OsWTQn7S8NfsFkk1dZXw"
}
```

7.3.2. Error Response

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

The HTTP POST request MUST contain 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 8.

The "error_info" JSON structure can contain the following error codes, defined in Section 8.1:

- 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"
    }
}
```

7.3.3. 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
```

8. Errors

"error_info" is a JSON object which contains the following properties:

error_code REQUIRED. Code representing the error. See Section 8.1 for possible values.

error_description OPTIONAL. Human-readable ASCII encoded text description of the error. Human-readable ASCII [RFC20] text providing additional information, used to assist the client developer in understanding the error that occurred. Values for the "error_description" parameter MUST NOT include characters outside the set %x20-21 / %x23-5B / %x5D-7E.

error_uri OPTIONAL. A URI identifying a human-readable web page with information about the error, used to provide the client developer with additional information about the error. Values for the "error_uri" parameter MUST conform to the URI-reference syntax

and thus MUST NOT include characters outside the set %x21 / %x23-5B / %x5D-7E.

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"
}
```

8.1. 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. The request is missing a required parameter, includes an unsupported parameter value (other than grant type), repeats a parameter, includes multiple credentials, utilizes more than one mechanism for authenticating the client, or is otherwise malformed.

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.

9. Implementation Considerations

9.1. Implementation of questioning methods

In order to interact with the Questioned User, the OP has to implement at least one questioning method. A questioning method has three purposes: authenticating the Questioned User, displaying the question and saving the Question User’s statement.

The authentication method and the associated level of assurance are refered by the "AMR" and "ACR" attributes.

There is no specific attribute to describe how the question was displayed and how the statement was saved. However, the OP is responsible to display the question as it is. If the question has be modified, the displayed question MUST be returned to the Client as it was displayed, in the "displayed_question" attribute of the User Statement Token. In the same logic, the Questioned User’s statement MUST be returned to the Client as it was displayed, in the "statement" attribute of the User Statement Token.

The fact that a User Questing Request and an Authentication Request (as defined in [OpenID.Core]) are handled using the same "AMR" does not mean that the same mechanism instance is used. It only means that the authentication method is the same. For instance, if the "AMR" value is "SMS_URL" is used, the URL contained in the SMS can refer to two different components. It can also refer to a unique component able to handle both User Questioning Requests and Authentication Requests.

If an OP can not display a question or the statements, it MUST return an error. To prevent these errors, it can inform the Clients of its

limitation and limit the possible questions or statements. For instance, if the mechanism used for User Questioning can only display a question of limited size with only two predefined statements, the Client SHOULD send a User Questioning Request with a "question_to_display" limited in size and the two predefined statements as "statements_to_display" values.

10. Security Considerations

10.1. Using wished_amr

In the User Questioning Request, the "wished_amr" value is a list of authentication methods. The OP SHOULD choose some of these authentication methods in order to achieve the "wished_acr". The OP CAN also decide to either use other authentication methods or achieve another Authentication Context Class Reference as the one specified in "wished_acr". The used authentication methods MUST be stated in "used_amr" and the achieved Authentication Context Class Reference MUST be stated in "used_acr" of the USer Statement Token.

10.2. User Statement Token Validation

In order to validate the User Statement Token, the Client MUST verify the signature, using Section 10.4.

Then, the Client MUST verify the "used_amr", "used_acr" and "statement_date". The Client is responsible to use the "statement_date" to prevent replay. The Client is responsible to use the "used_amr" and "used_acr" match with its security policy.

Then, the Client MUST verify the "displayed_question" value and the "statement" value, regarding the "question_to_display" value and the "possible_statements" value sent in the User Questioning Request. The Client is responsible of its decision to accept the "statement" and the "displayed_question". The Client is also responsible of any decision based on the "statement" value and the "displayed_question" value.

10.3. Non Repudation

Non-repudiation is an important feature of User Statement Token. To enable this feature, the asymetric signatures are the only allowed signatures for User Statement Tokens. This way, the Client can store the User Statement Token and use it as a proof of the statement made by Questioned User. This proof engages the signer’s responsibility, i.e the OP’s reponsibility.

10.4. Signatures and Encryption

Signatures and Encryption should respect the requirements defined in chapter 10 of [OpenID.Core] with one restriction: symetric signatures are FORBIDDEN.

10.5. Signing and Encryption Order

Signing and Encryption Order should respect the requirements defined in chapter 16.14 of [OpenID.Core].

11. Privacy Considerations

11.1. Personal Information

The User Statement Token contains the Questioned User’s statement on a question. This information is personal. If the "user_id" is present in the User Questioning Request, it will also be present in the User Statement Token, therefore Questioned User is directly identifiable. In order to protect the Questioned User’s privacy, Client SHOULD only store encrypted User Statement Tokens. The encryption can be computed either by the OP or by the Client, using Section 10.4.

12. Discovery Considerations

The OpenID Connect User Questioning API can be discovered within the OpenID Provider Metadata thanks to [OpenID.Discovery]. The scope "question" MUST be present in the "scopes_supported" array. The URL of User Questioning Request Endpoint MUST be configured as the value of "uq_request_endpoint". The URL of User Questioning Polling Endpoint MUST be configured as the value of "uq_polling_endpoint".

13. IANA Considerations

This document makes no requests of IANA.

14. Normative References

[E.164] International Telecommunication Union, "E.164: The international public telecommunication numbering plan", 2010, <http://www.itu.int/rec/T-REC-E.164-201011-I/en>.

[JWE] Jones, M., Rescorla, E., and J. Hildebrand, "JSON Web Encryption (JWE)", RFC 7516, DOI 10.17487/RFC7516, May 2015, <http://tools.ietf.org/html/rfc7516>.

[JWS] Jones, M., Bradley, J., and N. Sakimura, "JSON Web Signature (JWS)", RFC 7515, DOI 10.17487/RFC7515, May 2015, <http://tools.ietf.org/html/rfc7515>.

[JWT] Jones, M., Bradley, J., and N. Sakimura, "JSON Web Token (JWT)", draft-ietf-oauth-json-web-token (work in progress), 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, <http://www.rfc-editor.org/info/rfc20>.

[RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, DOI 10.17487/RFC2119, March 1997, <http://www.rfc-editor.org/info/rfc2119>.

[RFC4627] Crockford, D., "The application/json Media Type for JavaScript Object Notation (JSON)", RFC 4627, DOI 10.17487/RFC4627, July 2006, <http://www.rfc-editor.org/info/rfc4627>.

[RFC5322] Resnick, P., Ed., "Internet Message Format", RFC 5322, DOI 10.17487/RFC5322, October 2008, <http://www.rfc-editor.org/info/rfc5322>.

[RFC6749] Hardt, D., Ed., "The OAuth 2.0 Authorization Framework", RFC 6749, DOI 10.17487/RFC6749, October 2012, <http://www.rfc-editor.org/info/rfc6749>.

[RFC6750] Jones, M. and D. Hardt, "The OAuth 2.0 Authorization Framework: Bearer Token Usage", RFC 6750, DOI 10.17487/RFC6750, October 2012, <http://www.rfc-editor.org/info/rfc6750>.

Appendix A. Acknowledgements

The following have contributed to the development of this specification.

Appendix B. Notices

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