OAuth Working Group
Internet-Draft
Intended status: Standards Track
Expires: May 7, 2020

T. Spencer
Curity AB

November 4, 2019

The OAuth 2.0 Authorization Framework: Claims
draft-spencer-oauth-claims-00

Abstract

This document extends the OAuth 2.0 framework to include a simple query language that can be used by clients to request certain claims from an authorization server. This mechanism can be used during the authorization request and refresh request. It also defines a response parameter of the token and introspection endpoints that indicates to the caller which claims were authorized by the resource owner. Lastly, it stipulates how this request parameter can be used during token exchange, and how clients may request that certain claims be placed in an access token intended for a particular resource server.

Status of This Memo

This Internet-Draft is submitted in full conformance with the provisions of BCP 78 and BCP 79.

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF). Note that other groups may also distribute working documents as Internet-Drafts. The list of current Internet-Drafts is at https://datatracker.ietf.org/drafts/current/.

Internet-Drafts are draft documents valid for a maximum of six months and may be updated, replaced, or obsoleted by other documents at any time. It is inappropriate to use Internet-Drafts as reference material or to cite them other than as "work in progress."

This Internet-Draft will expire on May 7, 2020.

Table of Contents

1. Introduction

As stated in Section 1.4 of [RFC6749], an access token represents the specific scope and duration of access. The requested scope is verified by the authorization server according to its policy, and the

perhaps-different scope is granted by the resource owner. The requested and granted scope may vary due to the authorization server's policy and/or the resource owner's limitation of the granted scope. The resulting scope is enforced by the resource server. The way in which the client indicates the intended scope of access is by the "scope" request parameter defined in Section 3.3 of [RFC6749]. This specification defines a more sophisticated instrument to achieve this same purpose.

At times, this existing mechanism is too limited. In some uses cases, for example, a client may need to request particular claims from an authorization server. It may also do this to request specific claim values. Furthermore, a client may need to indicate to the authorization server that certain claims are essential for its ability to operate. In such cases, the grant is of little use to the client if the resource owner does not comply. Another example of when the existing "scope" parameter is insufficient is when the client knows that some claim is required by a particular resource server. The extent of a client's knowledge is usually limited to knowing that a claim is needed in an access token; however, in some cases, it may also know that a claim should be restricted to access tokens issued to a particular resource server. In these situations, the existing mechanism for stipulating the scope of access is insufficient.

To accommodate these use cases and requirements, this specification defines a new request parameter that can be used when the client obtains an authorization grant, as described in Section 1.3 of [RFC6749] and Section 2.1 of [I-D.ietf-oauth-token-exchange]. For each request wherein these fix grant types -- authorization code, implicit, resource owner password credentials, client credentials, and token exchange -- are sought, this specification defines a new parameter called "claims". It can be used by a client with any of these to request that certain claims and/or particular claim values be authorized by the resource owner. The value of this parameter is a JavaScript Object Notation (JSON) object [RFC8259]. This can also be used to indicate to the authorization server that the client considers some or all of the claims to be required. The client can also use this object to indicate that certain claim values are preferred or essential to its ability to operate on behalf of the resource owner.

During a refresh request (as described in Section 1.5 of [RFC6749]), the "claims" parameter defined herein can also be used to alter the resulting scope of access. This can be used, for example, to lessen the scope by including a certain subset of claims that should be in the new access token. After such, a client may increase the scope in a subsequent refresh request by including additional claim names in

the JSON object value of the "claims" authorization request parameter. When it does so, the client cannot, however, expand the scope from that which was initially authorized by the resource owner.

This specification also stipulates how the authorized claim names are returned from an authorization request and the result of introspecting a token.

This specification is designed to be compatible with OpenID Connect [OpenID.Core] but does not require the authorization server to support that protocol.

1.1. Claims vis-a-vis Scope Tokens

As previously stated, claims relate to scope tokens. How exactly is beyond the extent of this specification. Instead, this document provides a framework in which these two constructs can be used together or independently. That said, however, there are at least three common ways that claims will be used:

1. Not at all (in which case this specification is irrelevant).

2. In lieu of scope tokens.

3. Together with scope tokens.

The first and second option are straightforward. The third, however, will require a specification to define the relation between the two in order to achieve interoperability. For instance, OpenID Connect core [OpenID.Core] specification relates claims to scope tokens by grouping certain claims into various scope tokens. This grouping of claims into various scope tokens is RECOMMENDED when simultaneously using claims and scope tokens to request authorization.

1.2. Notational 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].

Unless otherwise noted, all the protocol parameter names and values are case sensitive.

1.3. Terminology

This specification uses the terms "Access Token", "Authorization Code", "Authorization Endpoint", "Authorization Grant", "Authorization Server", "Client", "Grant Type", "Redirection URI",

"Refresh Token", "Resource Owner", "Resource Server", and "Token Endpoint" defined by [RFC6749]; "Claim", "Claim Name", and "Claim Value" defined by [RFC7519]; and the following defined herein:

Claims Sink
: The location or destination where the authorization server MAY include all requested claims that are authorized by the resource owner. An acccess token intended for an unspecified resource server or an access token the client intends to send to a particular resource server or an ID token (when the OpenID Connect profile of this specification is used) are examples of claims sinks.

Claims Request Object
: Has the meaning ascribed to it in Section 3.

Claims Sink Query Object
: Has the meaning ascribed to it in Section 3.1.

Claim Value Query Object
: Has the meaning ascribed to it in Section 3.1.

Critical Claim
: Has the meaning ascribed to it in Section 3.2.

Essential Claim
: A claim specified by the client as being necessary to ensure a smooth authorization experience for a specific task requested by the resource owner.

Scope Token
: A case-sensitive string joined by spaces together with other such strings and included in the the "scope" request parameter of an authorization request (i.e., a "scope-token" as set forth in the ABNF of Section 3.3 of [RFC6749]).

Voluntary Claim
: A claim specified by the client as being useful but not essential for the specific task requested by the resource owner.

2. Protocol Flow

2.1. Authorization Request

When a client requests authorization from the resource owner indirectly via the authorization server, the protocol flow MAY include a query for certain claims. Based on the policy of the authorization server and the delegated access of the resource owner,

certain claims MAY be granted. Given an authorization grant, the authorization server informs the client as to which claims were actually issued (if different from those requested). This message exchange pattern is shown in Figure 1:

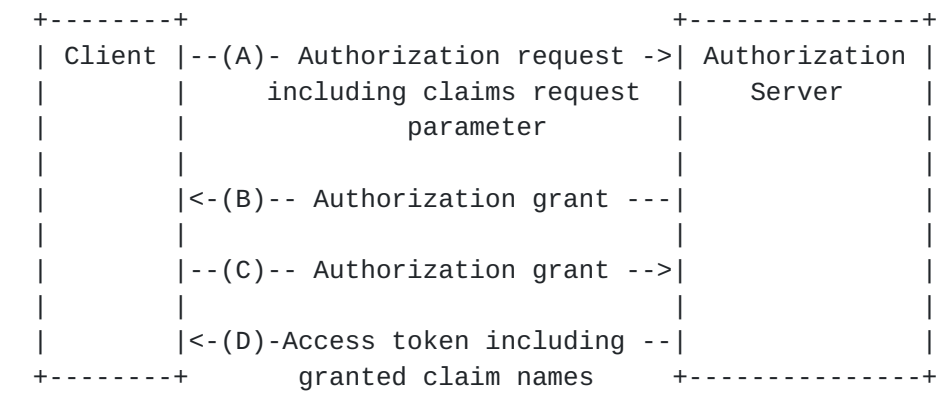

Figure 1: Protocol Flow that Includes Requested and Granted Claims

The steps in the flow illustrated in Figure 1 are generally the same as those described in Section 1.2 of [RFC6749] with a few important distinctions:

- During the authorization request (A), the client includes a claims request object as the corresponding value of the "claims" request parameter, as described in Section Section 3.1 below.

- After obtaining (B) and presenting the authorization grant (C), the response MAY include an access token and a possibly-empty list of claim names that were authorized (D). If the asserted claims embodied by the access token differ from those requested (A), then the authorization server MUST include a list of authorized claim names in the authorization response (D).

2.2. Refresh Request

As described in Section 1.5 of [RFC6749] and further explained in section 6 thereof, a client that was issued a refresh token MAY use this to narrow the scope of access for an access token. It does this by sending the "scope" request parameter in step (G) of Figure 2 of [RFC6749]. At times, the client might want to be more explicit about which claims should be included in the refreshed access token or about where those claims should be asserted.

To address this need, this document defines an additional input parameter that the client may send to the authorization server when it presents a refresh token to the token endpoint. This update to

the flow wherein a client refreshes an access token to narrow the scope of access to a particular set of claims is shown in Figure 2.

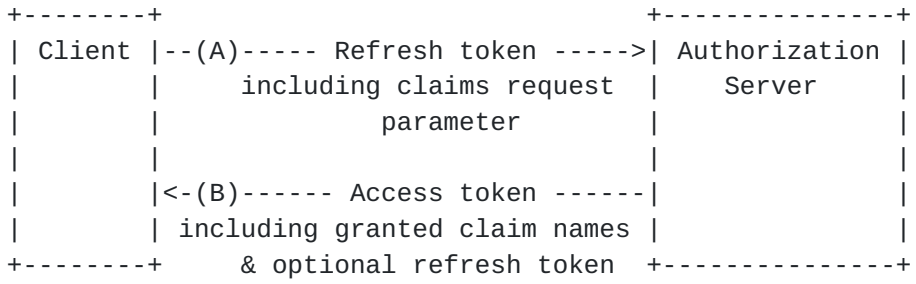

Figure 2: Narrowing the Scope of Access to a Particular Set of Claims

Downscoping in this flow is achieved when the client requests a subset of the claims authorized by the resource owner.

The client MAY request to change the claims sink where the authorized claims should be asserted using this flow. If the client does so, it is RECOMMENDED that the authorization server accept this change baring any policy to the contrary. The client MAY also send a claim value in the claim value query object(s) of the request. When it does, the authorization server SHOULD consider this request when asserting the claim but it MAY return an error if asserting a different claim value is against its policy or exceeds the authorization granted by the resource owner. If it is a critical claim, the requested claim value MUST be asserted or an error MUST result if the authorization server supports critical claims.

There are many corner cases that can arise when implementing this flow. Most stem from policy and configuration changes of the authorization server which may happen between the time an access token is issued and it is refreshed. Other complications arise when claims are used together with scope tokens. Both are beyond the scope of this specification and not addressed by this memo.

2.3. Token Introspection

[RFC7662] stipulates that the introspection endpoint of an authorization server must return a JSON [RFC8259] document representing the meta information surrounding the token, including its scope. This specification extends that object to include the claim names that the resource owner authorized the client for. This request/response interaction pattern is shown in Figure 3.

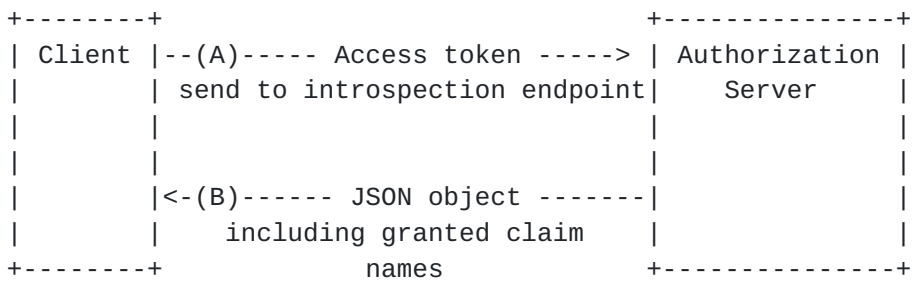

Figure 3: Introspecting an Access Token and Obtaining Granted Claims

Using this flow, a client will be informed about the authorized claim names in the same way it is informed about the scope of access by way of the "scope" response member.

2.4. Token Exchange

TBD

3. Claims Request Object

The "claims" request parameter value is a UTF-8 encoded JSON object ("Claims Request Object") specifying requested claims. Prior to transmission to the authorization server it is also form-URL-encoded as appropriate. The claims request object is not intended to be a mechanism that the client may use to instruct the authorization server to assert specific claims. Instead, it is a simple query language that a client can use to request certain claims or to specify that it would like the authorization server to obtain authorization from the resource owner for a claim, perhaps with a particular claim value. The claims request object provides a client with a more structured method of requesting the scope of access that the resource owner authorizes it for.

The top-level members of the claims request object SHOULD include at least one claims sink. The only specific claims sinks defined by this specification are "access_token", "*", and "?". Additionally, this specification also sets forth a mechanism by which a client may signal to the authorization server which claims it prefers to be included in an access token that it intends to be furnished to a particular resource server; this is done by using an absolute URI of the target service or resource as a claims sink. A claims request object MAY also contain the member "crit" to indicate parts of the claims request object that the authorization server MUST understand if the "crit" member itself is understood. Other members of a claims request object MAY be present; any that are not understood by the authorization server MUST be ignored.

An example of a claims request object that is sent to the authorization server as the value of the "claims" request parameter provided during an authorization request, refresh request or token exchange request is shown in Figure 4.

```
{
    "access_token" : {

    }
}
```

Figure 4: Example of a Claims Request Object

In this non-normative example, the "access_token" member is the claims sink. It is the location where the authorization server MAY include any of the requested claims that the resource owner authorizes. If the authorization server uses the requested claims from a particular claims sink to derive or determine alternative claims which it asserts, it is RECOMMENDED to consider the client's request to include those alternative claims in the same requested claims sink.

3.1. Requesting Particular Claim Names and Claim Values

Within the claims request object, a claims sink is associated with another JSON object ("Claims Sink Query Object"). This object contains properties that have the name of a claim which the client is requesting the authorization server to assert. The possible values associated with each of these is "null" or another JSON object ("Claim Value Query Object").

When the value is "null", it indicates that the claim with the associated claim name is a voluntary claim, and the client has no specific requirements on the claim value. Conversely, when the claim value query object is not "null" it is a JSON object with the following properties:

essential
: OPTIONAL. Indicates whether the claim being requested is an essential claim. If the value is "true", this indicates that the claim is an essential claim. If the value is "false" or if this property is not include, then the claim is a voluntary claim.

value
: OPTIONAL. Requests that the claim be returned with a particular value.

values
: OPTIONAL. Requests that the claim be returned with one of a set of values, with the values appearing in order of preference.

The properties "value" and "values" are mutually exclusive. If the client sends a claim value query object with both, the authorization server MUST return an error as described in Section Section 4 below.

By requesting essential claims, the client indicates to the authorization server (who indicates to the resource owner) that releasing these claims will ensure a smooth authorization for the specific task requested by that resource owner. If the claims are not available because the resource owner did not authorize their release or they are not present, the authorization server MUST NOT generate an error when claims are not returned.

Other members of the claim value query object MAY be defined to provide additional information about the requested claims. Any members of the claims value query object that are not understood by the authorization server MUST be ignored.

A non-normative example of the two possible types of values for a claim value query object is shown in Figure 5.

```
{
    "access_token" : {
        "https://example.com/claim1" : null,
        "fname" : {
            "value" : "John"
        }
    }
}
```

Figure 5: Example of a Claim Value Query Object

In this example, there are two claim names which the client is requesting "https://example.com/claim1" and "fname". The values associated with these are claim value query objects. The former is a simple query where the client has no preference on a particular value. For this reason, the client specifies the value "null". In the later case, the client has more precise needs: it desires the authorization server to assert a claim value of "John" for the claim name "fname". In such situations the authorization server MAY issue a claim with the claim name "fname" but with some other claim value than "John". Both are voluntary claims.

An example of an essential claim is shown in the non-normative listing of Figure 6.

```
{
    "access_token" : {
        "consentId" : {
            "essential" : true
        }
    }
}
```

Figure 6: Example of querying for an Essential Claim

This query indicates that the client would like the authorization server to issue an access token with a scope that includes a claim with the claim name "consentId". To ensure a smooth authorization experience at the resource server where the client will present the resulting access token, the client has indicated that the "consentId" claim is required, making it an essential claim.

As described above, a client may also indicate that it wishes the authorization server to assert a claim having a claim value that the client has some preference for. A non-normative example of such a query is show in Figure 7.

```
{
    "access_token" : {
        "accountId" : {
            "values" : ["act-123", "act-456"],
            "essential" : true
        },
        "paymentId" : {
            "value" : "pid-123456",
            "essential" : true
        }
    }
}
```

Figure 7: Example of querying for an Essential Claim with Certain Values

In this example, the client is requesting that the authorization server assert two essential claims: one named "accountId" and another named "paymentId". In the former case, the client requests that the claim value be "act-123" or "act-456". In the later case, a claim named "paymentId" is requested by the client to have a claim value of

"pid-123456". Again, the authorization server MUST NOT return an error if the resource owner does not authorize both of these claims or if they are non-existent. This is merely a request for a certain scope of access.

Another example inspired by the Revised Directive on Payment Services (PSD2) is shown in the non-normative listing of Figure 8.

```
{
    "access_token" : {
        "instructedAmount" : {
            "value" : {
                "amount" : 123.50,
                "currency" : "EUR"
            },
            "essential" : true
        },
        "debtorAccount/iban" : {
            "value" : "DE40100100103307118608",
            "essential" : true
        },
        "creditorName" : {
            "value" : "Merchant123",
            "essential" : true
        },
        "creditorAccount/iban" : {
            "value" : "DE02100100109307118603",
            "essential" : true
        },
        "remittanceInformationUnstructured" : {
            "value" : "Ref Number Merchant",
            "essential" : true
        }
    }
}
```

Figure 8: PSD2-related Example

In this example, the client is requesting (but not forcing) the authorization server to obtain authorization from the resource owner for five essential claims: "instructedAmount", "debtorAccount/iban", "creditorName", "creditorAccount/iban", and "remittanceInformationUnstructured". The claim value query object associated with each of these claim names has a particular value the client strongly prefers. One interesting case is the value of the "instructedAmount" essential claim; the query for the value of this claim is a JSON object with two properties. The authorization server

might use this claims request object to obtain the resource owner's consent before granting them, for instance. It might also check these values against a data source before asserting them. Based on the resource owner's choice or the data source lookup results, the authorization server may not issue the claims at all or may do so with some other value. For example, the authorization server may actually find that the "instructedAmount" value requested exceeds its policy's allowed limit and only prompt the resource owner to authorize EUR100.

Another interesting example of how structured scope of access can be requested is shown in the listing of Figure 9.

```
{
    "access_token" : {
        "credentialID" : {
            "value" : "qes_eidas",
            "essential" : true
        },
        "documentDigests" : {
            "value" : {
                "hash":"sTOgwOm+474gFj0q0x1iSNspKqbcse4IeiqlDg/HW=",
                "label":"Mobile Subscription Contract"
            },
            "essential" : true
        },
        "hashAlgorithmOID" : {
            "value" : "2.16.840.1.101.3.4.2.1"
        }
    }
}
```

Figure 9: ESI-related Example

This example shows how a client may request claims defined by the Electronic Signatures and Infrastructures (ESI) Protocols for remote digital signature creation. Like the previous example, the claims request object for the "access_token" claims sink includes a claim value query object for the "documentDigests" claim that includes a JSON object with multiple properties.

These illustrative examples hopefully impress upon the reader the versatility of this query language and the authorization server's prerogative to assert any claims with any claim values it chooses in its sole discretion. If the client's needs are stronger than preferential, it MAY use the "crit" member of the claims request object which the authorization server MAY understand.

3.2. Critical Members of a Claims Request Object

As described previously, the client can indicate to the authorization server that certain claims are preferential or essential to the smooth operation of the client. At times, however, the client's needs are stronger and require certain claims to be asserted. In such situations, the client would rather the authorization server return an error than grant access with different claims than those requested. This is not always possible for an authorization server, however, and a client MUST NOT assume that the authorization server can be controlled in this manner. To know if this interaction pattern in supported, the client must have a priori knowledge gained by some means not defined by this specification or by the presence of a "true" value in the authorization server's "critical_claims_supported" metadata (see section Section 9 below). An authorization server is RECOMMENDED to support this capability unless it cannot. When it does, the authorization server MUST issue any claim denoted as critical or it MUST return an error. The error must be "invalid_claims" as described below in Section 4.

A client indicates to the authorization server that it must understand certain claims and be able to assert them by including a list of JSON Pointers [RFC6901] associated with the "crit" member of the claims request object. Each such claim that the elements of this list point to is a "Critical Claim". The JSON Pointers in this list MUST refer to members of the claims request object and MUST NOT point to elements within the list itself. If any JSON Pointer refers to an element of the JSON Pointer list, the authorization server MUST return an error with a code of "invalid_request" if it supports critical claims. When the JSON Pointers are valid, if the authorization server does not understand any of the claims pointed to by any of the elements of this list, the authorization server MUST return an error of "invalid_claims". Likewise, if the authorization server is unable to assert a critical claim (and it supports critical claims), it MUST return the same error. If a critical claim is requested with a certain value (as in the following example), the authorization server MUST assert the claim with that exact claim value. If it's not able to (e.g., because the resource owner does not have an attribute with that particular value), the authorization server MUST return an error with a code of "invalid_claims" unless it does not support critical claims.

A non-normative example of a claims request object with a critical claim is shown in Figure 10.

```
{
    "crit" : [
 "/access_token/verified_claims/verification/trust_framework/value"
    ],
    "access_token" : {
        "verified_claims" : {
            "verification" : {
                "trust_framework" : {
                    "value" : "de_aml"
                }
            }
        }
    }
}
```

Figure 10: Example of a Request Containing a Critical Claim

In this example, the "value" member of the JSON object associated with "trust_framework" must be understood by the authorization server because it is pointed by the element of the critical claims list. The way in which the authorization server understands this particular query is beyond the scope of this specification. The only part of this example that is germane is the "crit" member of the claims request object which requires the authorization server to understand and assert a particular claim value (provided it understands the "crit" itself). If it cannot and if it supports critical claims, it must return an error.

It is not uncommon for a claim name to defined as a URI containing slashes ("/", %x2F). When such a claim is critical, the escaping described in Section 3 of [RFC6901] MUST be used, as in the non-normative listing of Figure 11.

```
{
    "crit" : ["/access_token/https:~1~1example.com~1claim1"],
    "access_token" : {
        "https://example.com/claim1" : null,
    }
}
```

Figure 11: Example of Escaping the JSON Pointer used to Refer to a Critical Claim with a Name Containing Slashes

3.3. Special Claims Sinks

A client may know that it needs a particular claim; however, it may not be aware which claims sink the claim should be included in. The client may prefer to leave this determination to the authorization server. In such cases, the client MAY use the claims sink "?" (%x3F) as mentioned in Section Section 3 above. This special claim sink may result in the claim being issued in the access token or any other claims sink that the authorization server deems appropriate. A non-normative example of a claims request object indicating that a particular claim be asserted in any claims sink is shown in Figure 12.

```
{
    "?" : {
        "https://example.com/claim1" : null,
    }
}
```

Figure 12: Example Requesting a Claim to be Asserted in Any Claims Sink

Similarly, there are situations where the client wishes claims to be asserted in all claims sinks the authorization server supports. Rather than requiring the the client to repeat its requirement for each claims sink, it MAY use the special claims sink "*" (%x2A). This claims sink indicates to the authorization server that the client prefers all claims included in the claims request object to be asserted in each claim sink that the authorization server supports. The two claims request objects shown in Figure Figure 13 and Figure 14 are equivalent (if the authorization server only supports the two claims sinks shown in the latter).

```
{
    "*" : {
        "https://exmaple.com/claim1" : null,
    }
}
```

Figure 13: Example Requesting a Claim to be Asserted in All Claims Sinks

```
{
    "access_token" : {
        "https://exmaple.com/claim1" : null,
    },
    "my-good-claims-sink" : {
        "https://exmaple.com/claim1" : null,
    }
}
```

Figure 14: Equivalent Example of Requesting a Claim to be Asserted in All Claims Sinks

The use of either the claims sink "?" and "*" with any other claims sink in the same claims request object is undefined. The authorization server SHOULD return an error or apply some other logic not defined by this specification. The client SHOULD NOT make such queries unless it has some knowledge gained a priori about the authorization server's support of such a query.

4. Obtaining Authorization

As stated in Section 4 of [RFC6749], a request for an access token requires the client to obtain authorization from the resource owner. As described there, this can be done using various grant types. To make a request for certain claims, the "claims" request parameter defined herein is used when requesting an authorization code, implicit, resource owner password credentials, or client credentials grant type. The "claims" request parameter MAY also be used with additional grant type that use the extension mechanism defined in Section 4.5 of [RFC6749] if so profiled by some other specification.

4.1. Authorization Code Grant

4.1.1. Authorization Request

When a client seeks to obtain authorization using the authorization code grant type defined in Section 4.1 of [RFC6749], the client MAY include the following additional query component that it sends to the authorization endpoint URI:

claims
: OPTIONAL. A claims request object as described in Section 3.

The value of this parameter must use the "application/x-www-form-urlencoded" format defined in Appendix B of [RFC6749].

A non-normative example of a request to the authorization endpoint with a URL-encoded value of the claims parameter is shown in Figure 15 (with extra line breaks for display purposes only):

```
GET /authorize?client_id=s6BhdRkqt3&response_type=code&
    claims=%7B%0A%20%20%22access_token%22%20%3A%20%7B%20%0A
        %20%20%20%20%22https%3A%2F%2Fexample.com%2Fclaim1%22%20%3A
        %20null%2C%0A%20%20%20%20%22fname%22%20%3A%20%7B%0A%20%20
        %20%20%20%20%22value%22%20%3A%20%22John%22%0A%20%20%20%20
        %7D%0A%20%20%7D%0A%7D
Host: server.example.com
```

Figure 15: Example of Using the Claims Request Parameter on the Authorize Endpoint

4.1.2. Error Response

If the authorization server understands the "claims" request parameter, it MUST redirect the user-agent of the resource owner to the client's redirection endpoint as described in Section 4.1.2.1 of [RFC6749] with one of the following "error" values:

claims_not_supported
: The authorization server understands but does not support the "claims" request parameter, and the client SHOULD NOT use it when requesting authorization.

invalid_request
: The authorization server MAY use this less-descriptive error code to indicate that the claims request parameter value is not accepted, e.g., because it is syntactically incorrect. It is, however, RECOMMENDED that the authorization server return "claims_not_supported" or "invalid_claims" as appropriate.

invalid_claims
: When a client makes a request for a critical claim, and the authorization server cannot assert such a claim because it is invalid, unknown, or malformed, this error results. If the request includes only claim names in the claims request object which are disallowed according to the authorization server's policy, this error (or the less-descriptive alternative, "invalid_request") MUST result.

4.1.3. Access Token Response

In a non-error case, the authorization server MAY include details about the claims that the client is authorized for. This is done by augmenting the response defined in Section 4.1.4 of [RFC6749]. In particular, the authorization server MAY include the following response member in the JSON object returned from the token endpoint:

claims
: OPTIONAL, if identical to the claims requested by the client; otherwise, REQUIRED. The space-separated claim names granted by the resource owner which denote the scope of the access token that is returned in the same response.

4.2. Implicit Flow

4.2.1. Authorization Request

When a client seeks to obtain authorization using the implicit grant type defined in Section 4.2 of [RFC6749], the client MAY include the following additional query component that it sends to the authorization endpoint URI:

claims
: OPTIONAL. A claims request object as described in Section 3.

The value of this parameter must use the "application/x-www-form-urlencoded" format defined in Appendix B of [RFC6749].

4.2.2. Access Token Response

In a non-error case, the authorization server MAY include details about the claims that the client is authorized for. This is done by augmenting the response defined in Section 4.2.2 of [RFC6749]. In particular, the authorization server MAY include the following response parameter included on the fragment component of the redirection URI:

claims
: OPTIONAL, if identical to the claims requested by the client; otherwise, REQUIRED. The space-separated claim names granted by the resource owner which denote the scope of the access token that is returned in the same response.

4.2.3. Error Response

If the authorization server understands the "claims" request parameter, it MUST redirect the user-agent of the resource owner to the client's redirection URI as described in Section 4.2.2.1 of [RFC6749] with one of the following "error" values:

claims_not_supported
: The authorization server understands but does not support the "claims" request parameter, and the client SHOULD NOT use it when requesting authorization.

invalid_request
: The authorization server MAY use this less-descriptive error code to indicate that the claims request parameter value is not accepted, e.g., because it is syntactically. incorrect. It is, however, RECOMMENDED that the authorization server return "claims_not_supported" or "invalid_claims" as appropriate.

invalid_claims
: When a client makes a request for a critical claim, and the authorization server cannot assert such a claim because it is invalid, unknown, or malformed, this error results. If the request includes only claim names in the claims request object which are disallowed according to the authorization server's policy, this error (or the less-descriptive alternative, "invalid_request") MUST result.

4.3. Resource Owner Password Credentials Grant

4.3.1. Access Token Request

When a client seeks to obtain authorization using the resource owner password credentials grant type defined in Section 4.3 of [RFC6749], the client MAY include the following additional parameter using the "application/x-www-form-urlencoded" format per Appendix B of [RFC6749] with a character encoding of UTF-8 in the HTTP request entity-body:

claims
: OPTIONAL. A claims request object as described in Section 3.

4.3.2. Access Token Response

In a non-error case, the authorization server MAY include details about the claims that the client is authorized for. This is done by augmenting the response defined in Section 4.3.3 of [RFC6749]. In

particular, the authorization server MAY include the following response member in the JSON object returned from the token endpoint:

claims
: OPTIONAL, if identical to the claims requested by the client; otherwise, REQUIRED. The space-separated claim names the client is authorized for which denote the scope of the access token that is returned in the same response.

If the request is invalid due to the value of the "claims" parameter, the authorization server returns an error with one of the following error codes:

claims_not_supported
: The authorization server understands but does not support the "claims" request parameter, and the client SHOULD NOT use it when requesting an access token.

invalid_request
: The authorization server MAY use this less-descriptive error code to indicate that the claims request parameter value is not accepted, accepted, e.g., because it is syntactically incorrect. It is, however, RECOMMENDED that the authorization server return "claims_not_supported" or "invalid_claims" as appropriate.

invalid_claims
: When a client makes a request for a critical claim, and the authorization server cannot assert such a claim because it is invalid, unknown, or malformed, this error results. If the request includes only claim names in the claims request object which are disallowed according to the authorization server's policy, this error (or the less-descriptive alternative, "invalid_request") MUST result.

4.4. Client Credentials Grant

4.4.1. Access Token Request

When a client seeks to obtain authorization using the client credentials grant type defined in Section 4.4 of [RFC6749], the client MAY include the following additional parameter using the "application/x-www-form-urlencoded" format per Appendix B of [RFC6749] with a character encoding of UTF-8 in the HTTP request entity-body:

claims
: OPTIONAL. A claims request object as described in Section 3.

4.4.2. Access Token Response

In a non-error case, the authorization server MAY include details about the claims that the client is authorized for. This is done by augmenting the response defined in Section 4.4.3 of [RFC6749]. In particular, the authorization server MAY include the following response member in the JSON object returned from the token endpoint:

claims
: OPTIONAL, if identical to the claims requested by the client; otherwise, REQUIRED. The space-separated claim names the client is authorized for which denote the scope of the access token that is returned in the same response.

If the request is invalid due to the value of the "claims" parameter, the authorization server returns an error with one of the following error codes:

claims_not_supported
: The authorization server understands but does not support the "claims" request parameter, and the client SHOULD NOT use it when requesting an access token.

invalid_request
: The authorization server MAY use this less-descriptive error code to indicate that the claims request parameter value is not accepted, e.g., because it is syntactically incorrect. It is, however, RECOMMENDED that the authorization server return "claims_not_supported" or "invalid_claims" as appropriate.

invalid_claims
: When a client makes a request for a critical claim, and the authorization server cannot assert such a claim because it is invalid, unknown, or malformed, this error results. If the request includes only claim names in the claims request object which are disallowed according to the authorization server's policy, this error (or the less-descriptive alternative, "invalid_request") MUST result.

5. Token Refresh

As defined in Section 6 of [RFC6749], a client may be provided with a refresh token. When it is, it can present this to the token endpoint of the authorization server in a refresh request. This specification extends the request and response of this flow as described in the following subsections.

5.1. Token Refresh Request

When performing a token refresh request, the client MAY send the following parameter using the "application/x-www-form-urlencoded" format per Appendix B of [RFC6749] with a character encoding of UTF-8 in the HTTP request entity-body:

claims
: OPTIONAL. A claims request object as described in Section 3.

If the client includes a claims request object in the request, it SHOULD ensure that the claim names in the claims value query object(s) are ones that were authorized by the resource owner. It can do this by remembering what was originally requested and/or from the authorization server's response to its authorization request which will include the list of claim names if they differ from those originally requested.

5.2. Access Refresh Response

In a non-error case, the authorization server MAY include details about the claims that the client is authorized for. This is done by augmenting the response defined in Section 5.1 of[RFC6749]. In particular, the authorization server MAY include the following response member in the JSON object returned from the token endpoint:

claims
: OPTIONAL, if identical to the claims requested by the client; otherwise, REQUIRED. The space-separated claim names the client is authorized for which denote the scope of the access token that is returned in the same response.

If the request is invalid due to the value of the value of the "claims" parameter, the authorization server returns an error with one of the following error codes:

claims_not_supported
: The authorization server understands but does not support the "claims" request parameter, and the client SHOULD NOT use it when requesting an access token.

invalid_request
: The authorization server MAY use this less-descriptive error code to indicate that the claims request parameter value is not accepted, e.g., because it is syntactically incorrect. It is, however, RECOMMENDED that the authorization server return "claims_not_supported" or "invalid_claims" as appropriate.

invalid_claims
: When a client makes a request for a critical claim, and the authorization server cannot assert such a claim because it is invalid, unknown, or malformed, this error results. If the request includes only claim names in the claims request object which are disallowed according to the authorization server's policy, this error (or the less-descriptive alternative, "invalid_request") MUST result.

6. Token Exchange

TBD

7. Token Introspection

This specification defines an additional top-level member in the JSON [RFC8259] object of the authorization server's introspection endpoint response as stipulated in Section 2.2 of [RFC7662].

claims
: OPTIONAL. The space-separated claim names granted by the resource owner which denote the scope of the access token.

8. Requesting Claims for a Particular Protected Resource

TBD

9. Authorization Server Metadata

An authorization server that supports the "claims" request parameter SHOULD declare this fact by including the following property in the authorization server metadata response [RFC8414]:

claims_parameter_supported
: OPTIONAL. A boolean value indicating that the authorization server supports the "claims" request parameter or not. A value of "true" indicates that it is supported. A value of "false", a "null" value, or the absence of the property means that the "claims" request parameter is not supported by the authorization server.

claims_supported
: RECOMMENDED. JSON array containing a list of the claim names of the Claims that the authorization server MAY be able to supply values for. Note that for privacy or other reasons, this might not be an exhaustive list.

critical_claims_supported

OPTIONAL. A boolean value indicating that the authorization server supports the possibility for the client to indicate that certain parts of a claims request object MUST be understood by the authorization server. A value of "false", a "null" value, or the absence of this member means that the authorization server MAY not support this interaction pattern, and the client MUST NOT assume that it does.

If the authorization server returns a value of "false" for "claims_parameter_supported" and true for "critical_claims_supported", the interpretation by the client is undefined. It is RECOMMENDED that the client assume that the authorization server is misconfigured and that it not attempt to request claims in a manner defined by this specification.

A non-normative example of an authorization server metadata response which indicates that the "claims" request parameter and critical claims are supported by the server is shown in Figure 16.

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

{
    "issuer" :
        "https://server.example.com",
    "authorization_endpoint" :
        "https://server.example.com/authorize",
    "token_endpoint" :
        "https://server.example.com/token",
    "token_endpoint_auth_methods_supported" :
        ["client_secret_basic", "private_key_jwt"],
    "token_endpoint_auth_signing_alg_values_supported" :
        ["RS256", "ES256"],
    "userinfo_endpoint" :
        "https://server.example.com/userinfo",
    "jwks_uri" :
        "https://server.example.com/jwks.json",
    "registration_endpoint" :
        "https://server.example.com/register",
    "scopes_supported" :
        ["openid", "profile", "email", "address", "phone",
         "offline_access"],
    "response_types_supported" :
        ["code", "code token"],
    "service_documentation" :
        "http://server.example.com/service_documentation.html",
    "ui_locales_supported" :
        ["en-US", "en-GB", "en-CA", "fr-FR", "fr-CA"],
    "claims_parameter_supported" : true,
    "critical_claims_supported" : true,
    "claims_supported" : ["sub", "http://example.com/monkey" ]
}
```

Figure 16: Example of Metadata of an Authorization Server that Supports the Claims Request Parameter and Critical Claims

Note the last three members in particular.

10. Security Considerations

TBD

11. Privacy Considerations

TBD

12. IANA Considerations

TBD

13. References

13.1. Normative References

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

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

[RFC6901] Bryan, P., Ed., Zyp, K., and M. Nottingham, Ed., "JavaScript Object Notation (JSON) Pointer", RFC 6901, DOI 10.17487/RFC6901, April 2013, <https://www.rfc-editor.org/info/rfc6901>.

[RFC7519] Jones, M., Bradley, J., and N. Sakimura, "JSON Web Token (JWT)", RFC 7519, DOI 10.17487/RFC7519, May 2015, <https://www.rfc-editor.org/info/rfc7519>.

[RFC7662] Richer, J., Ed., "OAuth 2.0 Token Introspection", RFC 7662, DOI 10.17487/RFC7662, October 2015, <https://www.rfc-editor.org/info/rfc7662>.

[RFC8259] Bray, T., Ed., "The JavaScript Object Notation (JSON) Data Interchange Format", STD 90, RFC 8259, DOI 10.17487/RFC8259, December 2017, <https://www.rfc-editor.org/info/rfc8259>.

[RFC8414] Jones, M., Sakimura, N., and J. Bradley, "OAuth 2.0 Authorization Server Metadata", RFC 8414, DOI 10.17487/RFC8414, June 2018, <https://www.rfc-editor.org/info/rfc8414>.

13.2. Informative References

[I-D.ietf-oauth-token-exchange] Jones, M., Nadalin, A., Campbell, B., Bradley, J., and C. Mortimore, "OAuth 2.0 Token Exchange", draft-ietf-oauth-token-exchange-19 (work in progress), July 2019.

[OpenID.Core]
Sakimura, N., Bradley, J., Jones, M., de Medeiros, B., and C. Mortimore, "OpenID Connect Core 1.0", OpenID Foundation Standards, February 2014, <http://openid.net/specs/openid-connect-core-1_0.html>.

Appendix A. Acknowledgements

The following individuals contributed ideas, feedback, and wording to this specification:

Mark Dobrinic, Jacob Ideskog

Appendix B. Document History

[[to be removed by the RFC editor before publication as an RFC]]

-00

o Initial draft.

Author's Address

Travis Spencer
Curity AB

Email: travis@curity.io