Web Authorization Protocol
Internet-Draft
Intended status: Standards Track
Expires: 28 November 2026

K. McGuinness
Independent

27 May 2026

# OAuth 2.0 Insufficient Claims Challenge
# draft-mcguinness-oauth-insufficient-claims-00

## Abstract

This specification defines an OAuth 2.0 challenge mechanism by which an Authorization Server or Protected Resource signals that a credential presented by a Client is otherwise acceptable but lacks claims required to fulfill the request. A new error code, insufficient_claims, together with a required_claims parameter that enumerates the missing claims, lets the recipient signal what is needed. The same challenge is used in Token Endpoint error responses, in Bearer authentication challenges at Protected Resources, and (optionally) in OAuth 2.0 Protected Resource Metadata.

The challenge is intentionally decoupled from how the Client responds. For back-channel re-issuance grants (OAuth 2.0 Token Exchange and Refresh Token), a Client uses the requested_claims Token Endpoint request parameter defined here. For grants that may require end-user interaction (authorization_code, device_code, CIBA), a Client uses an applicable front-channel claims request mechanism, such as the OpenID Connect claims request parameter.

A motivating use case is just-in-time account provisioning by a Resource Authorization Server receiving an identity assertion under the Identity Assertion Authorization Grant.

## About This Document

This note is to be removed before publishing as an RFC.

Status information for this document may be found at https://datatracker.ietf.org/doc/draft-mcguinness-oauth-insufficient-claims/.

Discussion of this document takes place on the Web Authorization Protocol Working Group mailing list (mailto:oauth@ietf.org), which is archived at https://mailarchive.ietf.org/arch/browse/oauth/. Subscribe at https://www.ietf.org/mailman/listinfo/oauth/.

Source for this draft and an issue tracker can be found at https://github.com/mcguinness/draft-mcguinness-oauth-insufficient-claims.

## Status of This Memo



## Copyright Notice



## Table of Contents

## 1. Introduction

OAuth 2.0 Clients present credentials at two natural recipients: a Token Endpoint (an assertion or subject token used to obtain an Access Token) and a Protected Resource (an Access Token used to invoke an operation). Either recipient may require specific claims about the subject (an identifier, a directory attribute, a policy attribute) that the credential does not carry, even though the credential is cryptographically valid and structurally acceptable.

There is currently no interoperable way for either recipient to signal which claims are missing, nor for the Client to convey that requirement to whoever issued the credential. Coordination happens out of band per deployment pair, limiting composability across Authorization Servers, Clients, and resources.

This specification separates the challenge (universal across recipients and grants) from the Client's response (grant-specific). It defines:

1. An OAuth 2.0 error code, insufficient_claims, that a recipient returns when a presented credential is otherwise acceptable but lacks claims required to fulfill the request. Used in Token Endpoint error responses (Section 5.2 of [RFC6749]) and Bearer authentication challenges at Protected Resources (Section 3 of [RFC6750]).

2. A required_claims parameter (a JSON array of claim entries) carried in Token Endpoint error response bodies, in JSON response bodies accompanying Protected Resource Bearer challenges, and (optionally) in OAuth 2.0 Protected Resource Metadata ([RFC9728]).

3. A requested_claims Token Endpoint request parameter that a Client includes on a back-channel re-issuance request to obtain a credential carrying the indicated claims. Defined for use with the OAuth 2.0 Token Exchange ([RFC8693]) and Refresh Token (Section 6 of [RFC6749]) grants only.

4. A required_claims Protected Resource Metadata parameter ([RFC9728]) by which a Protected Resource advertises the claims it may require, letting Clients request them at Access Token issuance and reduce runtime challenges.

5. A requested_claims_parameter_supported Authorization Server Metadata parameter ([RFC8414]) by which an Authorization Server advertises that it recognizes the requested_claims request parameter at its Token Endpoint.

The challenge applies to credentials presented at the Token Endpoint under any grant (see Section 3.3) and to Access Tokens presented at Protected Resources (see Section 3.4). The Client’s response is grant-specific:

* For OAuth 2.0 Token Exchange and Refresh Token grants, the Client sends a back-channel Token Endpoint request including the requested_claims parameter (see Section 4.1).

* For grants that may require end-user interaction (authorization_code, device authorization, CIBA, and similar), the Client initiates a new authorization request and conveys its claim requirements via an applicable front-channel claims request mechanism, such as the OpenID Connect claims request parameter (Section 5.5 of [OpenID.Core]; see Section 7.5). The requested_claims parameter defined here is not used with these grants.

The mechanism is opt-in and degrades gracefully. A recipient opts in by returning insufficient_claims; a Client that does not recognize the error treats the response as it would any other failure. An Authorization Server that does not recognize the requested_claims request parameter ignores it per Section 3.2 of [RFC6749]; the Client may then receive a second insufficient_claims, which per Section 4.4 it treats as a terminal failure. The Authorization Server remains the policy authority for claim release and does not release any claim it would not otherwise release.

### 1.1. Why a New Error Code

OAuth defines distinct error codes for distinct failure modes: invalid_grant ([RFC6749]) when a token is rejected, insufficient_scope ([RFC6750]) when a valid token's scope is insufficient, and insufficient_user_authentication ([RFC9470]) when a valid token's authentication context is insufficient. The condition addressed here is parallel to insufficient_scope but applies to claim content rather than scope, and arises at both the Token Endpoint and the Protected Resource. A distinct error code lets the Client distinguish a recoverable claim-negotiation failure from a hard rejection of the credential.

### 1.2. Motivating Use Cases

#### 1.2.1. Just-in-Time Account Provisioning at a Resource Authorization Server

A common deployment pattern, formalized by the Identity Assertion Authorization Grant [I-D.ietf-oauth-identity-assertion-authz-grant], conveys an identity assertion (the ID-JAG, a signed JWT) from an Identity Provider (IdP) Authorization Server to a Resource Authorization Server (RAS) governing a downstream resource. The Client obtains the ID-JAG from the IdP AS via OAuth 2.0 Token Exchange ([RFC8693]) and presents the ID-JAG to the RAS via the JWT Bearer assertion grant ([RFC7523]).

If the subject has no account at the RAS, the RAS may perform just-in-time (JIT) provisioning from the claims; if an account exists, the RAS may update it. Either operation requires a sufficient set of identity claims, and the required set varies per RAS. The mechanism defined here lets the RAS enumerate the claims it needs in an insufficient_claims response, and the Client forward that requirement to the IdP AS on the follow-up Token Exchange.

#### 1.2.2. Resource-Side Claim Requirements

A Protected Resource may require subject claims at request time that were not provisioned into the Access Token at issuance: for example, an authorization service whose policy depends on attributes the Client did not request via scope, or a downstream resource that needs claims an upstream Authorization Server did not include. The mechanism defined here allows the Protected Resource to challenge the Client to obtain a richer Access Token before retrying.

#### 1.2.3. Multi-Resource Refresh Tokens

A Client holding a Refresh Token that issues Access Tokens for multiple audiences (Section 6 of [RFC6749]) may need different claim sets in the Access Tokens it obtains for each audience. The requested_claims parameter on the Refresh Token request lets the Client request the audience-appropriate claim set for the next Access Token without initiating a new authorization flow.

#### 1.2.4. Other Assertion-Based Grants

Any assertion-based grant defined by [RFC7521], including JWT Bearer ([RFC7523]) and SAML 2.0 Bearer ([RFC7522]), shares the "incoming credential carrying claims" shape with Token Exchange. This specification’s error code applies uniformly to those grants; how the Client obtains a richer assertion is governed by the assertion provider’s protocol and is out of scope for this document.

## 2. Conventions and Definitions

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "NOT RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in BCP 14 [RFC2119] [RFC8174] when, and only when, they appear in all capitals, as shown here.

This document uses the following terms from [RFC6749]: Client, Authorization Server, Access Token, Token Endpoint, Protected Resource, and error response.

This document uses the following terms from [RFC8693]: Token Exchange, subject token, and requested token type.

This document uses the term "assertion" as defined in Section 1.2 of [RFC7521]: a package of information that allows identity and security information to be shared across security domains.

This document additionally uses the following terms:

Issuing Authorization Server: An OAuth 2.0 Authorization Server that issued, or can re-issue, the credential the Client uses in an OAuth 2.0 request. In the back-channel retry path (Section 4.1), the Issuing Authorization Server is the one the Client targets with requested_claims. In the Protected Resource case (Section 3.4), it is the Authorization Server that issued the Access Token presented at the resource.

Processing Authorization Server: The Authorization Server that

receives a Token Endpoint request from the Client and originates the insufficient_claims error response.

Claim: An attribute of the subject or the assertion, as defined in Section 4 of [RFC7519] for JWT-formatted tokens and generalized here to any token format that carries named claims.

## 3. The Insufficient Claims Challenge

The insufficient_claims error code (Section 3.1) and the required_claims response parameter (Section 3.2) define a common challenge that applies in two contexts:

* OAuth 2.0 Token Endpoint requests under any grant that presents a claim-bearing credential, including the Token Exchange grant ([RFC8693]), assertion-based grants ([RFC7521], [RFC7522], [RFC7523]), and the Refresh Token grant (Section 6 of [RFC6749]); see Section 3.3.

* Access Tokens presented at OAuth 2.0 Protected Resources; see Section 3.4.

The HTTP envelope and the carrier of the structured response differ per context, as described in Section 3.3 and Section 3.4. For Client behavior on receiving the error, see Section 4.

### 3.1. Error Code

The error code is:

insufficient_claims: The credential presented by the Client is acceptable but does not carry claims sufficient for the recipient to fulfill the request.

### 3.2. Response Parameter

A recipient SHOULD include a required_claims parameter enumerating the claims it requires.

In a JSON response body, the value of required_claims is a JSON array of claim entries. Each entry is either a JSON string or a JSON object.

A bare-string entry identifies a claim by name and indicates that the named claim is required; any value is acceptable. For example:

```
"required_claims": ["email", "given_name", "family_name"]
```

An object entry adds an optional value constraint and has the following members:

* name: REQUIRED. A JSON string identifying the claim name.

* value: OPTIONAL. A JSON value the claim MUST equal.

* values: OPTIONAL. A JSON array of values, one of which the claim MUST equal.

An entry MUST NOT include both value and values. An entry that includes neither is equivalent to the bare-string form. For example:

```
"required_claims": [
  "email",
  {"name": "email_verified", "value": true},
  {"name": "tenant_id", "values": ["t-123", "t-456"]}
]
```

A claim name identifies a claim as it appears in the credential (for example, the JSON member name in a JWT). Claim names use the scope-token syntax of Section 3.3 of [RFC6749]: visible ASCII characters excluding space (%x20), double-quote (%x22), and backslash (%x5C). Claim names are case-sensitive. The order of entries in the array is not significant.

This document does not define essential/voluntary semantics; all entries are required. Profiles needing richer expressivity SHOULD define their own parameter rather than overload required_claims (see Section 7.5).

The same JSON array shape is used wherever this document conveys a claim list in a JSON document, namely in Token Endpoint error response bodies (Section 5.2 of [RFC6749]), in Protected Resource error response bodies (Section 3.4), and in Protected Resource Metadata documents (Section 5.1). This aligns with the scopes_supported and claims_supported metadata conventions of [RFC8414] and [OpenID.Core].

When the same list is carried as the requested_claims parameter on an application/x-www-form-urlencoded Token Endpoint request (see Section 4.1), the JSON array value is serialized to a JSON string and percent-encoded per application/x-www-form-urlencoded rules. This follows the precedent of authorization_details ([RFC9396]). Characters that have special meaning in JSON or in form bodies, including [, ], ", and ,, MUST be percent-encoded.

The required_claims parameter is OPTIONAL. A recipient that cannot or does not wish to enumerate the missing claims MAY return the error code without it. In that case the Client has no machine-readable basis for retry and SHOULD treat the response as a terminal failure unless out-of-band information is available.

A recipient MUST NOT include in required_claims any claim name whose semantics are not defined by a registered claims registry (such as the JSON Web Token Claims registry), a registered profile (such as OpenID Connect Core [OpenID.Core]), or prior agreement with the population of Issuing Authorization Servers it expects to interoperate with.

Claim names in required_claims are interpreted in the context of the (issuer, subject, audience, Client) tuple of the request, in the same way as the scope parameter (Section 3.3 of [RFC6749]); see Section 7.3. The Authorization Server's claim release policy, the content and format of issued claims, and the meaning attached to a given claim name are all scoped to that tuple.

Entries forwarded from a required_claims field into a requested_claims parameter on a retry Token Endpoint request retain their meaning only when both of the following hold:

* The audience, subject, and Client are preserved between the original request and the retry.

* The Issuing and Processing Authorization Servers share an understanding of the claim names and constraint values involved.

A recipient MUST NOT include the same claim name in more than one entry of the array. A Client receiving duplicate entries for the same claim name in required_claims MUST treat the value as malformed and MUST NOT forward it as requested_claims. An Authorization Server receiving duplicate entries for the same claim name in requested_claims MUST treat the parameter as malformed.

## 3.3. Returned at the Token Endpoint

A Processing Authorization Server receiving a Token Endpoint request MUST first validate the presented credential per its own acceptance rules and the rules of the grant profile in use. If the credential is rejected for cryptographic, audience, issuer, type, or freshness reasons, the server MUST respond with the applicable error from Section 5.2 of [RFC6749] (typically invalid_grant) rather than the error defined here.

If the credential is acceptable but does not carry claims sufficient to fulfill the request, the Processing Authorization Server MAY respond with insufficient_claims. The error code is returned in the error parameter of the Token Endpoint error response, formatted per Section 5.2 of [RFC6749] with HTTP status code 400.

Example error response:

```
HTTP/1.1 400 Bad Request
Content-Type: application/json
Cache-Control: no-store

{
  "error": "insufficient_claims",
  "error_description": "The presented credential is missing required claims.",
  "required_claims": ["email", "given_name", "family_name"]
}
```

### 3.4. Returned at the Protected Resource

When a Protected Resource receives a request bearing an Access Token that is otherwise valid but does not carry claims sufficient to fulfill the request, the Protected Resource MAY return an authentication challenge containing the insufficient_claims error code.

The challenge follows Section 3 of [RFC6750]. The Protected Resource MUST respond with HTTP status 403 Forbidden. The response MUST include a WWW-Authenticate header field with the error parameter set to insufficient_claims. The response MAY also include the resource_metadata auth-param ([RFC9728]) to point the Client at its Protected Resource Metadata document (see Section 5.1).

Structured details of the challenge, including the claims the Protected Resource requires, are conveyed in the JSON response body rather than as additional WWW-Authenticate auth-params. This follows the body-carrying pattern of the OAuth 2.0 Token Endpoint error response (Section 5.2 of [RFC6749]) and keeps the Bearer challenge header simple.

When the Protected Resource emits a response body alongside the challenge, the body's Content-Type MUST be application/json. The body SHOULD include a required_claims member (Section 3.2) enumerating the claims the Protected Resource requires. The response SHOULD include Cache-Control: no-store.

The 403 status code is consistent with insufficient_scope (Section 3.1 of [RFC6750]): the Access Token authenticates the Client and subject but does not authorize this request because of insufficient claim content. The 401 Unauthorized status code is not used because the failure is not authentication of the Client or the token.

Example challenge (line breaks in the WWW-Authenticate header are shown for readability):

```
HTTP/1.1 403 Forbidden
WWW-Authenticate: Bearer error="insufficient_claims",
                  resource_metadata="https://api.example.com/.well-known/oauth-protect
ed-resource"
Content-Type: application/json
Cache-Control: no-store

{
  "error": "insufficient_claims",
  "error_description": "The Access Token is missing required claims.",
  "required_claims": ["email", "department"]
}
```

A Protected Resource that cannot emit a JSON body MAY convey only the WWW-Authenticate header. The Client then has no machine-readable claim list to echo on retry, and SHOULD treat the response as a terminal failure unless out-of-band information (such as Protected Resource Metadata, Section 5.1) is available.

If the resource request was unauthenticated (no Access Token presented), the Protected Resource MUST use the existing error semantics of Section 3 of [RFC6750] (no error parameter or error="invalid_token"), not insufficient_claims. The error defined here is only applicable when the presented Access Token is otherwise acceptable.

## 4. Client Remediation

A Client that receives an insufficient_claims response or challenge, and that supports the mechanism defined in this document, obtains a new credential carrying the claims listed in the required_claims field. The Client then retries the original request with the new credential. The retry mechanism depends on how the original credential was obtained:

* When the original credential was obtained via OAuth 2.0 Token Exchange ([RFC8693]) or from a Refresh Token (Section 6 of [RFC6749]), the Client sends a back-channel Token Endpoint request including the requested_claims parameter (Section 4.1).

* When the original credential was obtained via an interactive authorization flow (the authorization_code grant, the device authorization grant, the CIBA grant, or any other grant that may involve end-user interaction), the Client initiates a new authorization request and conveys its claim requirements using an applicable front-channel claims request mechanism, such as the OpenID Connect claims request parameter (Section 5.5 of [OpenID.Core]); see Section 4.2.

* When the original credential was a pre-issued assertion presented under a JWT Bearer ([RFC7523]) or SAML Bearer ([RFC7522]) grant, the Client obtains a richer assertion from the assertion provider using whatever mechanism that provider supports, then presents the new assertion in a fresh assertion-grant request; see Section 7.2.

### 4.1. Back-Channel Token Endpoint Request Parameter

This document extends the Token Endpoint request (Section 3.2 of [RFC6749]) with the following parameter:

requested_claims: OPTIONAL. A JSON array of claim entries that the Client is requesting be included in the issued token. The array shape, including the bare-string and constraint-object entry forms, and the claim-name syntax match the required_claims parameter defined in Section 3.2. When included in a Token Endpoint request, the JSON array is serialized to a JSON string and percent-encoded as the value of the requested_claims form parameter. The requested_claims parameter MUST NOT appear more than once in a single Token Endpoint request. A Client SHOULD NOT include the same claim name in more than one entry of the array. An Authorization Server receiving duplicate entries for the same claim name MUST treat the parameter as malformed.

The requested_claims parameter is defined for use with the OAuth 2.0 Token Exchange grant ([RFC8693]) and the OAuth 2.0 Refresh Token grant (Section 6 of [RFC6749]). These grants are back-channel re-issuance flows in which the Client already holds a credential that the Authorization Server can use to determine eligibility for the requested claims, without a fresh end-user interaction. The parameter MUST NOT be used with grants that may require end-user interaction, including the authorization_code grant, the urn:ietf:params:oauth:grant-type:device_code grant, and the urn:openid:params:grant-type:ciba grant; see Section 4.2.

When sending a request that carries requested_claims, the Client MUST identify the same audience or resource as the original request that produced the insufficient_claims response or challenge. For Token Exchange ([RFC8693]), the Client uses the audience and/or resource

request parameters defined by that specification. For Refresh Token requests, the Client uses the resource parameter defined by Resource Indicators ([RFC8707]).

The Authorization Server's release policy is scoped to the indicated audience or resource. Changing the audience or resource may cause a different policy to apply; see Section 7.3 and Section 7.4.

A Client MUST ensure that the requested_claims value it sends is a well-formed JSON array of claim entries conforming to the syntax defined in Section 3.2, and MUST NOT forward malformed input received in a required_claims field from a recipient.

A Client SHOULD forward the value received in required_claims verbatim as the value of requested_claims, percent-encoded for the application/x-www-form-urlencoded body. A Client MAY include additional entries (bare claim names or constraint objects) in the array based on local knowledge of the target resource.

#### 4.1.1. Use with Token Exchange

Under the OAuth 2.0 Token Exchange grant ([RFC8693]), the Client sends a new request to the Issuing Authorization Server. The Client selects the subject_token and subject_token_type based on its established credential relationship with the Issuing Authorization Server. This is typically the same subject_token the Client originally exchanged at that Authorization Server.

The following example uses the ID-JAG token type from the Identity Assertion Authorization Grant ([I-D.ietf-oauth-identity-assertion-authz-grant]); any requested_token_type registered with IANA may be used. The audience value matches the Processing Authorization Server that issued the original insufficient_claims response.

```
POST /oauth2/token HTTP/1.1
Host: issuer.example.com
Content-Type: application/x-www-form-urlencoded

grant_type=urn%3Aietf%3Aparams%3Aoauth%3Agrant-type%3Atoken-exchange
&requested_token_type=urn%3Aietf%3Aparams%3Aoauth%3Atoken-type%3Aid-jag
&subject_token=eyJhbGciOi...
&subject_token_type=urn%3Aietf%3Aparams%3Aoauth%3Atoken-type%3Aid_token
&audience=https%3A%2F%2Fapi.example.com%2F
&requested_claims=%5B%22email%22%2C%22given_name%22%2C%22family_name%22%5D
```

The requested_claims value above is the URL-encoded form of the JSON array ["email","given_name","family_name"].

#### 4.1.2. Use with Refresh Token

A Client holding a Refresh Token (Section 6 of [RFC6749]) MAY include requested_claims on a Refresh Token request to obtain a re-issued Access Token (and, where applicable, ID Token) carrying the indicated claims. This is particularly useful for multi-resource Refresh Tokens, where a single Refresh Token issues Access Tokens for several audiences and the claim set required differs per audience.

Example request:

```
POST /oauth2/token HTTP/1.1
Host: as.example.com
Content-Type: application/x-www-form-urlencoded

grant_type=refresh_token
&refresh_token=8xLOxBtZp8...
&resource=https%3A%2F%2Fapi.example.com%2F
&requested_claims=%5B%22email%22%2C%22department%22%5D
```

The requested_claims value above is the URL-encoded form of the JSON array ["email","department"].

The Authorization Server applies its normal Refresh Token policy, including any policy that would have applied had the Client requested those claims at the original authorization.

An Authorization Server that would require fresh end-user consent or re-authentication to release a requested claim has two options. It MAY decline that claim and issue a token without it. Alternatively, it MAY respond with an OpenID Connect error such as consent_required or interaction_required (Section 3.1.2.6 of [OpenID.Core]), prompting the Client to initiate a new authorization request (see Section 4.2).

### 4.2. Interactive Grants

For grants that may require end-user interaction (the authorization_code grant, the device authorization grant, the CIBA grant, and similar), a Client responding to insufficient_claims SHOULD initiate a new authorization request and convey its claim requirements using an applicable front-channel claims request mechanism, such as the OpenID Connect claims request parameter (Section 5.5 of [OpenID.Core]); see Section 7.5.

This separation lets the Authorization Server apply consent, user authentication, or interaction policies appropriate to the new claim release, rather than requiring those policies to be expressed as a back-channel side effect.

### 4.3. Authorization Server Behavior

An Authorization Server that supports the requested_claims request parameter on Token Exchange or Refresh Token requests SHOULD include each requested claim in the issued token, subject to its own policy. In particular, the Authorization Server:

* MUST NOT release a claim it would not otherwise release under its configured policy for the subject, the audience, and the requesting Client. Receipt of requested_claims is not authorization to bypass consent, privacy, or release controls.

* MUST NOT treat requested_claims as a complete enumeration of the claims that should be issued. The Authorization Server MAY include additional claims it would normally include.

* MAY decline to include any specific requested claim. Declining to include a claim is not, by itself, a reason to fail the request.

* SHOULD NOT fail the request because of an unrecognized claim name in requested_claims.

* When an entry includes a value or values constraint, the Authorization Server MUST NOT include the claim in the issued token with a value that does not satisfy the constraint. The Authorization Server MAY decline to include the claim entirely if it cannot satisfy the constraint.

* An Authorization Server that does not support constraint entries MUST NOT ignore the value or values members and treat the entry as the bare-string form. It MUST either decline the constrained claim or reject the request with invalid_request.

Issuance of a token in response to a requested_claims request is not an assertion by the Authorization Server that all requested claims were honored. A Client that needs to verify a specific claim is present in the re-issued token before retrying inspects the issued token using the mechanisms applicable to its format.

### 4.4. No Loop Guarantee

This specification does not oblige an Issuing Authorization Server to satisfy a requested_claims request, nor a recipient (Processing Authorization Server or Protected Resource) to accept a re-issued credential whose claims still fall short. For the purposes of this section, the "logical exchange" is the workflow that produced the original insufficient_claims response or challenge, whether returned from a Token Endpoint or a Protected Resource. A Client SHOULD issue

at most one retry per logical exchange, and MUST treat any subsequent insufficient_claims for the same logical exchange as a terminal failure, even if it enumerates a different required_claims value.

## 5. Discovery

This section defines two static-metadata mechanisms that complement the runtime challenge. Protected Resources advertise the claims they may require so that Clients can request them at issuance time. Authorization Servers advertise support for the requested_claims request parameter so that Clients can use it proactively. Both mechanisms are optional and degrade gracefully when absent.

### 5.1. Protected Resource Metadata

A Protected Resource MAY advertise the set of claims it may require in its OAuth 2.0 Protected Resource Metadata document ([RFC9728]) using the following metadata parameter:

required_claims: OPTIONAL. A JSON array of claim entries with the shape defined in Section 3.2, enumerating claims the Protected Resource may require in Access Tokens. Entries MAY be bare claim names or constraint objects, though deployments commonly advertise bare names only.

Example metadata fragment:

```
{
  "resource": "https://api.example.com/",
  "authorization_servers": ["https://as.example.com/"],
  "scopes_supported": ["read", "write"],
  "required_claims": ["email", "given_name", "family_name"]
}
```

The advertised set is advisory and represents a maximal or typical set of claims the resource may require across its operations. A Client that obtains an Access Token carrying these claims will, in the common case, avoid an insufficient_claims challenge.

The Protected Resource MAY still return insufficient_claims for operations whose requirements depend on request path, parameters, subject state, or policy. The Protected Resource is not obliged to require every advertised claim for every operation.

Clients SHOULD treat the advertised list as a hint for Access Token acquisition. Clients SHOULD NOT depend on it as a complete or stable contract.

As with required_claims in error responses and challenges, and requested_claims in Token Endpoint requests, claim names in the metadata are interpreted in the context of the (issuer, subject, audience, Client) tuple of any Access Token that will be presented at the resource; see Section 7.3. For Authorization Server-side advertising of requested_claims support, see Section 5.2.

### 5.2. Authorization Server Metadata

An Authorization Server that supports the requested_claims request parameter defined in Section 4.1 SHOULD advertise that support in its OAuth 2.0 Authorization Server Metadata ([RFC8414]) using the following metadata parameter:

requested_claims_parameter_supported: OPTIONAL. Boolean value indicating whether the Authorization Server supports the requested_claims Token Endpoint request parameter defined in this document. If omitted, the default value is false.

A Client MAY consult this metadata to determine whether to send requested_claims on a Token Exchange or Refresh Token request. This includes proactively including the parameter on a first attempt where the Client expects specific claims will be needed.

A Client MUST NOT rely on the absence or false value of this metadata to predict an Authorization Server’s behavior on requested_claims. An Authorization Server that does not advertise support MAY still honor the parameter, consistent with Section 3.2 of [RFC6749].

This metadata parameter advertises support for the Token Endpoint request parameter only. It does not advertise support for the insufficient_claims error code in error responses. Recipients return that error code per their own policy regardless of metadata.

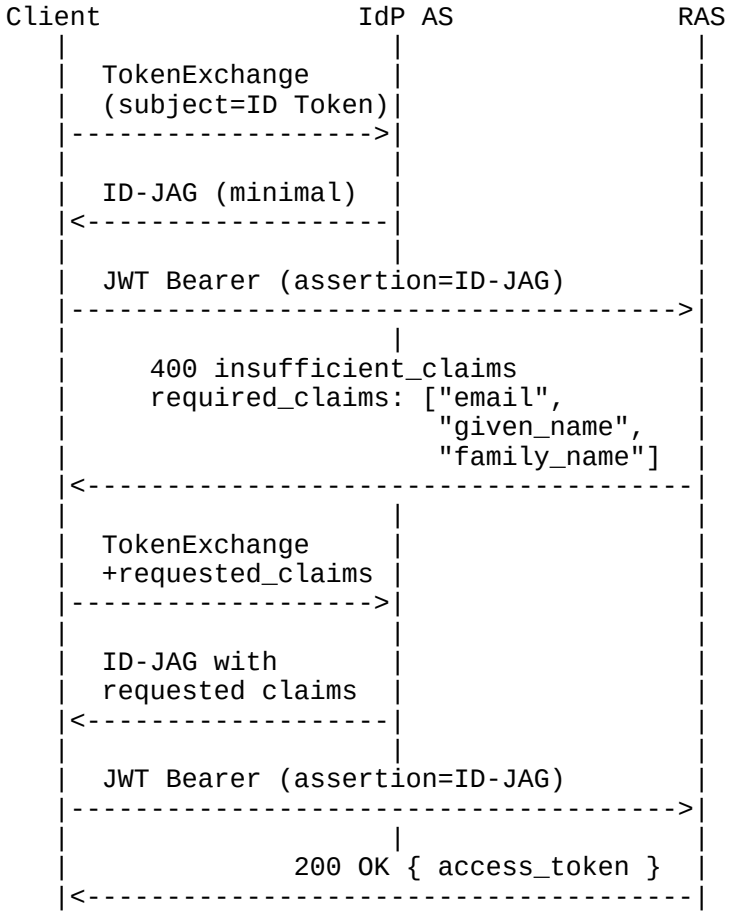
For Protected Resource-side advertising of required claims, see Section 5.1.

## 6. End-to-End Examples

The two diagrams below illustrate the principal flows. For a full worked example with concrete actors, decoded JWT payloads, and complete HTTP messages, see Appendix A.

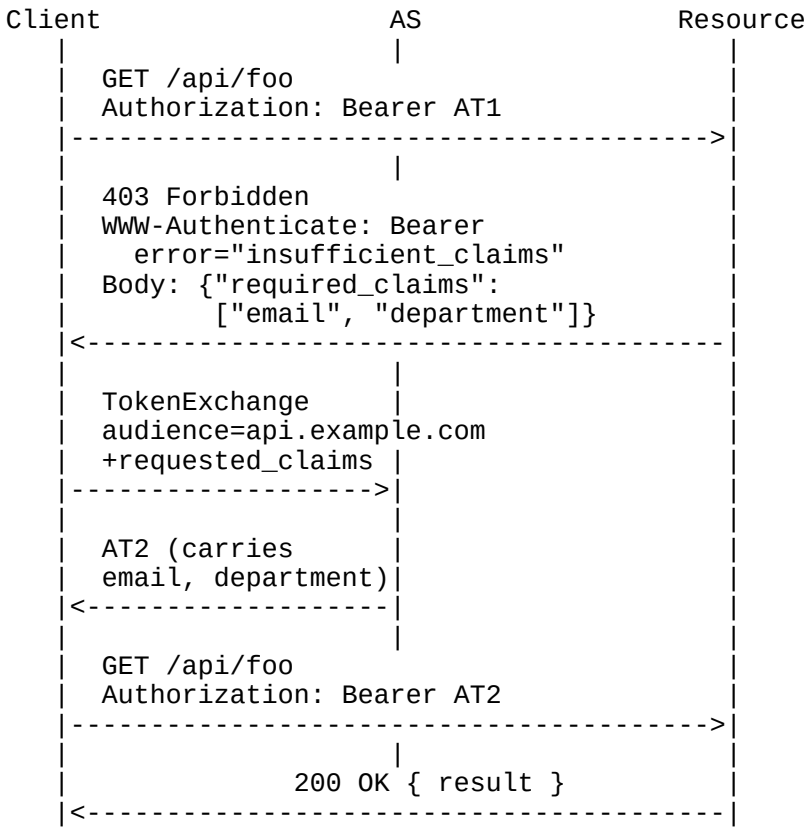
### 6.1. Token Exchange Retry

The following non-normative example illustrates the Token Endpoint flow using the Identity Assertion Authorization Grant profile ([I-D.ietf-oauth-identity-assertion-authz-grant]), where the Processing Authorization Server is the Resource Authorization Server (RAS) and the Issuing Authorization Server is the Identity Provider (IdP) Authorization Server.




### 6.2. Protected Resource Challenge

The following non-normative example illustrates a Protected Resource returning insufficient_claims and the Client obtaining a richer Access Token via Token Exchange before retrying.




## 7. Relationship to Other Specifications

### 7.1. Identity Assertion Authorization Grant

[I-D.ietf-oauth-identity-assertion-authz-grant] defines the urn:ietf:params:oauth:token-type:id-jag token type and the role of the Resource Authorization Server in a cross-domain identity assertion flow. The Client mints an ID-JAG via OAuth 2.0 Token Exchange ([RFC8693]) at the IdP AS and presents the ID-JAG to the RAS via the JWT Bearer assertion grant ([RFC7523]). With the mechanism defined here, specific claim requirements at the RAS become a deployment-time decision negotiated at runtime via the required_claims/requested_claims pair, with the Issuing Authorization Server retaining policy authority over release.

### 7.2. Assertion-Based Grants

[RFC7521] defines the framework for assertion-based OAuth 2.0 grants. [RFC7522] and [RFC7523] define specific profiles for SAML 2.0 and JWT assertions respectively. The insufficient_claims error code defined here applies uniformly to Token Endpoint requests using any of these grants.

The requested_claims request parameter (Section 4.1) is defined for Token Exchange and Refresh Token grants and is not used with JWT Bearer or SAML Bearer requests directly. The assertion presented in those grants is obtained out of band, and obtaining a richer assertion is governed by the assertion provider's protocol.

Where the assertion provider is itself an OAuth 2.0 Authorization Server, a Client receiving insufficient_claims would typically obtain a richer assertion via Token Exchange or via an interactive flow using the OpenID Connect claims request parameter. The Client then presents the new assertion in a fresh JWT Bearer or SAML Bearer request.

### 7.3. OAuth 2.0 scope Parameter

The scope parameter (Section 3.3 of [RFC6749]) is a coarse, Authorization Server-specific authorization request signal. In some deployment profiles, including OpenID Connect, specific scope values (for example, profile, email) imply the issuance of specific claims. This document does not redefine that mapping.

The mechanism defined here is finer-grained than scope: a recipient names individual claims directly in required_claims, independent of any scope-to-claim mapping the issuing Authorization Server may apply. The Client can forward a precise requirement as requested_claims without knowing that mapping.

Both required_claims and requested_claims share with scope the property that values are interpreted in the context of the (issuer, subject, audience, Client) tuple of the request. A claim name has no globally registered semantics that override an Authorization Server's local release policy. The same name may carry different content, format, or release rules at different Authorization Servers, for different subjects, audiences, or Clients.

A Client forwarding claim names between a recipient (in required_claims) and an Issuing Authorization Server (in requested_claims) is relying on those two parties having a shared understanding of the listed names, typically through registered claims or profile alignment.

requested_claims complements scope rather than replacing it. A Client retrying at the Token Endpoint MAY include both parameters in the same request. The Authorization Server applies its scope semantics as it otherwise would, and additionally takes requested_claims into account.

### 7.4. OAuth 2.0 Resource Indicators

[RFC8707] allows a Client to specify the resource (the resource parameter) at which the requested token will be used. The Authorization Server may use that signal, together with the audience parameter, to determine the audience of the issued token and to apply audience-specific claim release policy. Two consequences follow:

1. Claims releasable to one audience may not be releasable to another. A retry under Section 4.1 that changes the audience or resource value from the original request may obtain a token under different policy. The Authorization Server may include or omit different claims, and the recipient that returned insufficient_claims may not accept the re-issued token. For this reason Section 4.1 requires the Client to preserve the original audience and, where applicable, resource on retry.

2. A Processing Authorization Server or Protected Resource that returns insufficient_claims does so for a specific audience. A required_claims value that is meaningful for one audience may be meaningless, or trigger different release policy, for another.

### 7.5. OpenID Connect claims Request Parameter

OpenID Connect Core 1.0 ([OpenID.Core], Section 5.5) defines a claims request parameter at the OIDC Authorization Endpoint and Token Endpoint, carrying a JSON object that requests individual claims for the ID Token and UserInfo responses with optional essential/voluntary semantics and value constraints.

This document defers to the OIDC claims parameter for any response to insufficient_claims that requires a new authorization request, including the authorization_code grant, the device authorization grant, the CIBA grant, and similar interactive flows. In these cases:

* The Client SHOULD initiate a new authorization request to the Authorization Server and include a claims parameter reflecting the entries enumerated in the required_claims field of the insufficient_claims response or challenge. Mapping constraint entries to OIDC claims syntax is deployment-specific and only possible where the Authorization Server supports the corresponding OIDC claim request semantics.

* The Authorization Server applies its normal interactive-flow policy: it MAY prompt the end user for consent, perform additional authentication, or otherwise involve the user before releasing newly requested claims. This is the natural place to handle such policy, which the back-channel requested_claims parameter (Section 4.1) cannot accommodate.

* On completion of the new authorization request, the Client obtains an Access Token (and, where applicable, ID Token) carrying the requested claims, and retries the original request.

The requested_claims parameter defined in this document is not a substitute for the OIDC claims parameter for interactive grants; the two parameters target different stages of the OAuth 2.0 flow and different deployment patterns:

* OIDC claims is presented at an Authorization or backchannel-authentication request, where end-user consent and authentication can be evaluated. It supports essential/voluntary claims and value constraints.

* requested_claims is presented at the Token Endpoint as part of a back-channel re-issuance request (Token Exchange or Refresh Token), where no end-user interaction occurs. It is a hint to the Authorization Server about which claims should be included in the issued token, with optional value or values constraints.

An Authorization Server MAY support both claims (on its OIDC endpoints) and requested_claims (on Token Exchange and Refresh Token requests at its Token Endpoint). This document does not define an interaction between them.

### 7.6. OAuth 2.0 Bearer Token Usage and Step-Up Authentication

[RFC6750] defines the Bearer authentication scheme used at the Protected Resource and the insufficient_scope error returned when the Access Token’s scope is insufficient. [RFC9470] defines insufficient_user_authentication returned when the authentication context behind the Access Token is insufficient. The mechanism in Section 3.4 of this document follows the same challenge pattern but

addresses claim content rather than scope or authentication context.

### 7.7. OAuth 2.0 Rich Authorization Requests

[RFC9396] carries structured request objects (authorization_details) as JSON. The claim-entry syntax in this document is intentionally much narrower: it can name claims and, when needed, request equality against one value or one of a set of values. Profiles needing richer authorization semantics, schema references, or non-claim constraints SHOULD define their own parameter rather than overload required_claims.

## 8. Security Considerations

### 8.1. Disclosure of Claim Requirements

The required_claims parameter, whether in an error response body or in Protected Resource Metadata, discloses to the Client the set of claims the recipient intends to consume. This is generally low-sensitivity information, comparable to the OAuth scope parameter, but operators SHOULD confirm that disclosing the list to any Client capable of reaching the Token Endpoint or Protected Resource is acceptable.

When required_claims entries include value or values constraints, those constraints disclose more deployment-specific policy than bare claim names: they reveal specific tenant identifiers, role names, or other policy attributes that the recipient evaluates. Operators SHOULD consider whether disclosing these values to any Client capable of reaching the endpoint is acceptable. Where the constraint values themselves are sensitive, deployments SHOULD use bare claim names and validate values out of band.

### 8.2. Disclosure of Issued Claims

When an Authorization Server includes additional claims in a re-issued token in response to requested_claims, those claims may be readable by the Client (for example, a JWT-formatted token can be parsed by anyone holding the token). Authorization Servers SHOULD apply the same release policy as for any other token issued to the same subject, audience, and Client.

In deployments where the intended consumer of the issued token is the recipient only and the Client is treated as a transport, the security guidance from [I-D.ietf-oauth-identity-assertion-authz-grant] on audience scoping and (where supported) encryption of the token applies.

The Authorization Server is the policy authority for release. Receipt of a requested_claims parameter from a Client MUST NOT be treated as user consent, subject release authorization, or any other form of override of the Authorization Server's configured policy. An Authorization Server that finds that satisfying requested_claims would violate policy MUST decline the affected claims and MAY decline the request.

### 8.3. Untrusted Input

A recipient constructing a required_claims field, and an Authorization Server consuming a requested_claims parameter, MUST treat the value as untrusted input until validated. Implementations MUST validate that each entry in the array conforms to the syntax constraints stated in this document, including claim-name syntax, mutual exclusion of value and values, and the absence of duplicate claim names. Implementations MUST NOT pass the value into log formatters, database queries, or claim release rules without proper escaping or parameterization.

### 8.4. Replay and Caching

The insufficient_claims error response (at the Token Endpoint or Protected Resource) and the required_claims parameter it carries have no authentication state, no nonce, and no expiration, and MUST NOT be used to make any access decision. They serve only to guide the Client's next request. Responses carrying required_claims SHOULD be returned with Cache-Control: no-store.

### 8.5. Denial of Service

A recipient returning insufficient_claims invites the Client to perform an additional Token Endpoint request against an Authorization Server. Implementations on all sides SHOULD apply standard rate limiting to protect their endpoints. As noted in Section 4.4 and Section 3.4, Clients SHOULD NOT retry indefinitely.

### 8.6. Trust Between Recipient and Issuing Authorization Server

This specification does not establish trust between a recipient (Processing Authorization Server or Protected Resource) and the Issuing Authorization Server. The Client mediates between them and can add, remove, or modify the claim list it received in required_claims before forwarding it as requested_claims. For this reason requested_claims is advisory.

The Issuing Authorization Server SHOULD evaluate the request against the Client's identity, the requested audience, and its local release policy. The Issuing Authorization Server MUST NOT infer a recipient requirement from requested_claims alone. It MUST NOT treat the parameter's presence as authorization to release any claim.

## 9. Privacy Considerations

The mechanism in this document is designed to reduce, not increase, claim disclosure relative to a static policy. Without required_claims, an Issuing Authorization Server that wishes to interoperate with multiple recipients must either include claims speculatively (releasing data that may not be needed) or omit them and break the downstream operation. With required_claims, an Authorization Server can release claims only when a specific exchange requires them, subject to its own policy.

Operators of recipients (Processing Authorization Servers and Protected Resources) SHOULD request the minimum set of claims necessary, and SHOULD NOT enumerate claims they do not consume.

Operators of Issuing Authorization Servers SHOULD apply consent, contractual, or regulatory release controls before honoring any specific entry in requested_claims.

## 10. IANA Considerations

### 10.1. OAuth Extensions Error Registry

IANA is requested to add the following entry to the "OAuth Extensions Error" registry established by Section 11.4 of [RFC6749].

Name: insufficient_claims

Usage Location: Token Endpoint Error Response, Resource Access Error Response

Protocol Extension: This document

Change Controller: IETF

Specification Document(s): This document

Description: Indicates that the credential presented by the Client is acceptable but does not carry claims sufficient for the recipient to fulfill the request. Returned in OAuth 2.0 Token Endpoint error responses and in OAuth 2.0 Bearer authentication challenges at Protected Resources.

### 10.2. OAuth Parameters Registry

IANA is requested to add the following entries to the "OAuth Parameters" registry established by Section 11.2 of [RFC6749].

#### 10.2.1. required_claims

Name: required_claims

Parameter Usage Location: token response

Change Controller: IETF

Specification Document(s): This document

Notes: The required_claims parameter appears in the JSON error response body of OAuth 2.0 Token Endpoint error responses (Section 5.2 of [RFC6749]) and in the JSON error response body of Protected Resources returning insufficient_claims (Section 3.4).

#### 10.2.2. requested_claims

Name: requested_claims

Parameter Usage Location: token request

Change Controller: IETF

Specification Document(s): This document

### 10.3. OAuth Protected Resource Metadata Registry

IANA is requested to add the following entry to the "OAuth Protected Resource Metadata" registry established by Section 7.1 of [RFC9728].

Metadata Name: required_claims

Metadata Description: A JSON array of claim entries (bare claim names or constraint objects) enumerating claims the Protected Resource may require in Access Tokens, advisory and not necessarily a complete or stable contract; see Section 5.1.

Change Controller: IETF

Specification Document(s): This document

### 10.4. OAuth Authorization Server Metadata Registry

IANA is requested to add the following entry to the "OAuth Authorization Server Metadata" registry established by Section 7.1 of [RFC8414].

Metadata Name: requested_claims_parameter_supported

Metadata Description: Boolean value indicating whether the Authorization Server supports the requested_claims Token Endpoint request parameter defined in this document; see Section 5.2.

Change Controller: IETF

Specification Document(s): This document

## 11. References

### 11.1. 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/rfc/rfc2119>.

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

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

[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/rfc/rfc7519>.

[RFC7521] Campbell, B., Mortimore, C., Jones, M., and Y. Goland, "Assertion Framework for OAuth 2.0 Client Authentication and Authorization Grants", RFC 7521, DOI 10.17487/RFC7521, May 2015, <https://www.rfc-editor.org/rfc/rfc7521>.

[RFC8174] Leiba, B., "Ambiguity of Uppercase vs Lowercase in RFC 2119 Key Words", BCP 14, RFC 8174, DOI 10.17487/RFC8174, May 2017, <https://www.rfc-editor.org/rfc/rfc8174>.

[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/rfc/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/rfc/rfc8414>.

[RFC8693] Jones, M., Nadalin, A., Campbell, B., Ed., Bradley, J., and C. Mortimore, "OAuth 2.0 Token Exchange", RFC 8693, DOI 10.17487/RFC8693, January 2020, <https://www.rfc-editor.org/rfc/rfc8693>.

[RFC8707] Campbell, B., Bradley, J., and H. Tschofenig, "Resource Indicators for OAuth 2.0", RFC 8707, DOI 10.17487/RFC8707, February 2020, <https://www.rfc-editor.org/rfc/rfc8707>.

[RFC9728] Jones, M.B., Hunt, P., and A. Parecki, "OAuth 2.0 Protected Resource Metadata", RFC 9728, DOI 10.17487/RFC9728, April 2025, <https://www.rfc-editor.org/rfc/rfc9728>.

## 11.2. Informative References

[I-D.ietf-oauth-identity-assertion-authz-grant] Parecki, A., McGuinness, K., and B. Campbell, "Identity Assertion JWT Authorization Grant", Work in Progress, Internet-Draft, draft-ietf-oauth-identity-assertion-authz-grant-04, 21 May 2026, <https://datatracker.ietf.org/doc/html/draft-ietf-oauth-identity-assertion-authz-grant-04>.

[OpenID.Core] Sakimura, N., Bradley, J., Jones, M., de Medeiros, B., and C. Mortimore, "OpenID Connect Core 1.0 incorporating errata set 2", December 2023, <https://openid.net/specs/openid-connect-core-1_0.html>.

[RFC7522] Campbell, B., Mortimore, C., and M. Jones, "Security Assertion Markup Language (SAML) 2.0 Profile for OAuth 2.0 Client Authentication and Authorization Grants", RFC 7522, DOI 10.17487/RFC7522, May 2015, <https://www.rfc-editor.org/rfc/rfc7522>.

[RFC7523] Jones, M., Campbell, B., and C. Mortimore, "JSON Web Token (JWT) Profile for OAuth 2.0 Client Authentication and Authorization Grants", RFC 7523, DOI 10.17487/RFC7523, May 2015, <https://www.rfc-editor.org/rfc/rfc7523>.

[RFC9396] Lodderstedt, T., Richer, J., and B. Campbell, "OAuth 2.0 Rich Authorization Requests", RFC 9396, DOI 10.17487/RFC9396, May 2023, <https://www.rfc-editor.org/rfc/rfc9396>.

[RFC9470] Bertocci, V. and B. Campbell, "OAuth 2.0 Step Up Authentication Challenge Protocol", RFC 9470, DOI 10.17487/RFC9470, September 2023, <https://www.rfc-editor.org/rfc/rfc9470>.

## Appendix A. Worked End-to-End Example

This appendix is non-normative. It walks through a complete just-in-time (JIT) provisioning flow using the Identity Assertion Authorization Grant ([I-D.ietf-oauth-identity-assertion-authz-grant]) profile, the Token Exchange retry path of Section 4.1, and the Protected Resource Metadata advertisement of Section 5.1. HTTP messages are shown in full; JWT contents are shown as decoded payloads for readability.

### A.1. Actors

* *End user:* Alice, an employee of Example Corp.

* *Client:* Acme Tools (client_id=acme-tools), an enterprise SaaS application that uses Alice’s IdP session to access a downstream API on her behalf.

* *Identity Provider Authorization Server (IdP AS):* https://idp.example.com/. This is the Issuing Authorization Server in the terminology of Section 4.1.

* *Resource Authorization Server (RAS):* https://ras.example.com/. This is the Processing Authorization Server.

* *Downstream resource:* https://api.example.com/.

The RAS performs JIT account provisioning when the subject is unknown to it. Its policy requires email, given_name, and family_name to provision a new account.

### A.2. Step 1: Client obtains an ID-JAG from the IdP AS

The Client has authenticated Alice via OpenID Connect at the IdP and holds her ID Token. The Client sends a Token Exchange request to the IdP AS for an ID-JAG bound to the RAS's audience. In this first attempt, the Client does not yet know which claims the RAS requires and sends no requested_claims.

```
POST /oauth2/token HTTP/1.1
Host: idp.example.com
Authorization: Basic YWNtZS10b29sczpzM2NyZXQ=
Content-Type: application/x-www-form-urlencoded

grant_type=urn%3Aietf%3Aparams%3Aoauth%3Agrant-type%3Atoken-exchange
&requested_token_type=urn%3Aietf%3Aparams%3Aoauth%3Atoken-type%3Aid-jag
&subject_token=eyJhbGciOiJSUzI1NiIs...
&subject_token_type=urn%3Aietf%3Aparams%3Aoauth%3Atoken-type%3Aid_token
&audience=https%3A%2F%2Fras.example.com%2F
```

The IdP AS validates the ID Token, applies its default release policy for this Client and audience, and issues an ID-JAG with minimal content. Decoded payload:

```
{
  "iss": "https://idp.example.com/",
  "sub": "alice-uuid-12345",
  "aud": "https://ras.example.com/",
  "client_id": "acme-tools",
  "exp": 1748190000,
  "iat": 1748189700
}
```

The Token Exchange response:

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

{
  "issued_token_type": "urn:ietf:params:oauth:token-type:id-jag",
  "access_token": "eyJhbGciOiJSUzI1NiIs...",
  "token_type": "N_A",
  "expires_in": 300
}
```

The access_token field carries the ID-JAG by convention of [RFC8693].

### A.3. Step 2: Client presents the ID-JAG to the RAS

The Client presents the ID-JAG to the RAS using the JWT Bearer assertion grant ([RFC7523]). The ID-JAG carries the audience identifier of the RAS in its aud claim, so no separate audience form parameter is needed.

```
POST /oauth2/token HTTP/1.1
Host: ras.example.com
Authorization: Basic YWNtZS10b29sczpzM2NyZXQ=
Content-Type: application/x-www-form-urlencoded

grant_type=urn%3Aietf%3Aparams%3Aoauth%3Agrant-type%3Ajwt-bearer
&assertion=eyJhbGciOiJSUzI1NiIs...
```

### A.4. Step 3: RAS returns insufficient_claims

The RAS validates the ID-JAG (signature, issuer, audience, freshness) and accepts it. It then attempts to resolve sub=alice-uuid-12345 to a local account; no match. To JIT-provision Alice, it needs email, given_name, and family_name. The ID-JAG carries none of these, so the RAS challenges the Client:

```
HTTP/1.1 400 Bad Request
Content-Type: application/json
Cache-Control: no-store

{
  "error": "insufficient_claims",
  "error_description":
    "Cannot provision user; missing required claims.",
  "required_claims": ["email", "given_name", "family_name"]
}
```

### A.5. Step 4: Client retries the Token Exchange with requested_claims

The Client extracts the required_claims JSON array from the response body, percent-encodes the array as the requested_claims form value, and sends a new Token Exchange request to the IdP AS with the same audience and the original ID Token as the subject token:

```
POST /oauth2/token HTTP/1.1
Host: idp.example.com
Authorization: Basic YWNtZS10b29sczpzM2NyZXQ=
Content-Type: application/x-www-form-urlencoded

grant_type=urn%3Aietf%3Aparams%3Aoauth%3Agrant-type%3Atoken-exchange
&requested_token_type=urn%3Aietf%3Aparams%3Aoauth%3Atoken-type%3Aid-jag
&subject_token=eyJhbGciOiJSUzI1NiIs...
&subject_token_type=urn%3Aietf%3Aparams%3Aoauth%3Atoken-type%3Aid_token
&audience=https%3A%2F%2Fras.example.com%2F
&requested_claims=%5B%22email%22%2C%22given_name%22%2C%22family_name%22%5D
```

The requested_claims value above is the URL-encoded form of the JSON array ["email","given_name","family_name"] returned in the RAS's required_claims field, forwarded verbatim.

### A.6. Step 5: IdP AS issues an enriched ID-JAG

The IdP AS checks that Alice has consented to share these claims with Acme Tools (or that the deployment's policy otherwise permits release). It then issues an ID-JAG that includes the requested claims. Decoded payload:

```
{
  "iss": "https://idp.example.com/",
  "sub": "alice-uuid-12345",
  "aud": "https://ras.example.com/",
  "client_id": "acme-tools",
  "exp": 1748190600,
  "iat": 1748190300,
  "email": "alice@example.com",
  "given_name": "Alice",
  "family_name": "Carter"
}
```

### A.7. Step 6: Client re-presents the enriched ID-JAG to the RAS

The Client repeats the JWT Bearer request from Step 2, this time using the enriched ID-JAG returned in Step 5.

```
POST /oauth2/token HTTP/1.1
Host: ras.example.com
Authorization: Basic YWNtZS10b29sczpzM2NyZXQ=
Content-Type: application/x-www-form-urlencoded

grant_type=urn%3Aietf%3Aparams%3Aoauth%3Agrant-type%3Ajwt-bearer
&assertion=eyJhbGciOiJSUzI1NiIs...
```

### A.8. Step 7: RAS provisions Alice’s account and issues an Access Token

The RAS:

1. Validates the enriched ID-JAG.

2. Confirms no existing account for sub=alice-uuid-12345.

3. Creates a local account using email, given_name, and family_name.

4. Issues an Access Token for the downstream API.

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

{
  "access_token": "eyJhbGciOiJSUzI1NiIs...",
  "token_type": "Bearer",
  "expires_in": 3600
}
```

### A.9. Step 8: Client calls the downstream API

```
GET /v1/projects HTTP/1.1
Host: api.example.com
Authorization: Bearer eyJhbGciOiJSUzI1NiIs...
```

The downstream API validates the Access Token, finds Alice’s now-provisioned account, and returns her project list.

### A.10. Avoiding the Round Trip with Protected Resource Metadata

If the RAS publishes its claim requirements via Protected Resource Metadata (Section 5.1), the Client can include requested_claims on its very first Token Exchange request and skip Steps 3 and 4 entirely.

Example PRM document at https://ras.example.com/.well-known/oauth-protected-resource:

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

{
  "resource": "https://ras.example.com/",
  "authorization_servers": ["https://idp.example.com/"],
  "required_claims": ["email", "given_name", "family_name"]
}
```

A Client that fetches this metadata at startup includes requested_claims=%5B%22email%22%2C%22given_name%22%2C%22family_name%22%5D (the URL-encoded form of ["email","given_name","family_name"]) in Step 1, and the IdP AS issues the enriched ID-JAG on the first attempt. The remainder of the flow proceeds directly to Step 7.

### A.11. Authorization Server Metadata Discovery

A Client that consults the IdP AS's metadata can also confirm support for the requested_claims parameter before sending it, avoiding wasted requests against an Authorization Server that does not recognize the parameter. Example IdP AS metadata fragment at https://idp.example.com/.well-known/oauth-authorization-server:

```
{
  "issuer": "https://idp.example.com/",
  "token_endpoint": "https://idp.example.com/oauth2/token",
  "grant_types_supported": [
    "authorization_code",
    "refresh_token",
    "urn:ietf:params:oauth:grant-type:token-exchange"
  ],
  "requested_claims_parameter_supported": true
}
```

## Acknowledgments

The author thanks the OAuth working group participants who raised the underlying interoperability gap in the Identity Assertion Authorization Grant draft.

## Author's Address

Karl McGuinness
Independent
Email: public@karlmcguinness.com