Web Authorization Protocol
Internet-Draft
Intended status: Standards Track
Expires: March 23, 2020

T. Lodderstedt
yes.com
J. Richer
Bespoke Engineering
B. Campbell
Ping Identity
September 20, 2019

# OAuth 2.0 Rich Authorization Requests
# draft-lodderstedt-oauth-rar-02

Abstract

This document specifies a new parameter "authorization_details" that is used to carry fine grained authorization data in the OAuth authorization request.

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 March 23, 2020.

Table of Contents



## 1. Introduction

The OAuth 2.0 authorization framework [RFC6749] defines the parameter "scope" that allows OAuth clients to specify the requested scope, i.e., the permission, of an access token. This mechanism is sufficient to implement static scenarios and coarse-grained authorization requests, such as "give me read access to the resource owner's profile" but it is not sufficient to specify fine-grained authorization requirements, such as "please let me make a payment with the amount of 45 Euros" or "please give me read access to folder A and write access to file X".

This draft introduces a new parameter "authorization_details" that allows clients to specify their fine-grained authorization requirements using the expressiveness of JSON data structures.

For example, a request for payment authorization can be represented using a JSON object like this:

```
[
 {
   "type": "payment_initiation",
   "instructedAmount":{
      "currency":"EUR",
      "amount":"123.50"
   },
   "debtorAccount":{
      "iban":"DE40100100103307118608"
   },
   "creditorName":"Merchant123",
   "creditorAccount":{
      "iban":"DE02100100109307118603"
   },
   "remittanceInformationUnstructured":"Ref Number Merchant"
 }
]
```

In addition to facilitating custom authorization requests, this draft also introduces a set of common data type fields for use across different APIs.

For a comprehensive discussion of the challenges arising from new use cases in the open banking and electronic signing spaces see [transaction-authorization].

## 1.1. Conventions and Terminology

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 specification uses the terms "access token", "refresh token", "authorization server", "resource server", "authorization endpoint", "authorization request", "authorization response", "token endpoint", "grant type", "access token request", "access token response", and "client" defined by The OAuth 2.0 Authorization Framework [RFC6749].

# 2. Request parameter "authorization_details"

The request parameter "authorization_details" contains a JSON array of JSON objects. Each JSON object contains the data to specify the authorization requirements for a certain type of resource. The type of resource or access requirement is determined by the "type" field.

This example shows the specification of authorization details for a payment initiation transaction:

```
[
   {
      "type": "payment_initiation",
      "actions": ["initiate", "status", "cancel"],
      "locations":[
        "https://example.com/payments"
      ],
      "instructedAmount":{
         "currency":"EUR",
         "amount":"123.50"
      },
      "debtorAccount":{
         "iban":"DE40100100103307118608"
      },
      "creditorName":"Merchant123",
      "creditorAccount":{
         "iban":"DE02100100109307118603"
      },
      "remittanceInformationUnstructured":"Ref Number Merchant"
   }
]
```

This example shows a combined request asking for access to account information and permission to initiate a payment:

```
[
   {
      "type": "account_information",
      "actions":
        ["list_accounts", "read_balances", "read_transactions"],
      "identifier": "abc-123565",
      "locations": [
        "https://example.com/accounts"
      ]
   },
   {
      "type": "payment_initiation",
      "actions": ["initiate", "status", "cancel"],
      "locations":[
        "https://example.com/payments"
      ],
      "instructedAmount":{
         "currency":"EUR",
         "amount":"123.50"
      },
      "debtorAccount":{
         "iban":"DE40100100103307118608"
      },
      "creditorName":"Merchant123",
      "creditorAccount":{
         "iban":"DE02100100109307118603"
      },
      "remittanceInformationUnstructured":"Ref Number Merchant"
   }
]
```

The JSON objects with "type" fields of "account_information" and "payment_initiation" represent the different authorization data to be used by the AS to ask for consent and MUST subsequently also be made available to the respective resource servers. The array MAY contain several elements of the same "type".

## 2.1. Authorization data elements types

This draft defines a set of common data elements that are designed to be usable across different types of APIs. These data elements MAY be combined in different ways depending on the needs of the API. Unless otherwise noted, all data elements are OPTIONAL.

type:
: The type of resource request as a string. This field MAY define which other elements are allowed in the request. This element is REQUIRED.

locations:
: An array of strings representing the location of the resource or resource server. This is typically composed of URIs.

actions:
: An array of strings representing the kinds of actions to be taken at the resource. The values of the strings are determined by the API being protected.

data:
: An array of strings representing the kinds of data being requested from the resource.

identifier:
: A string identifier indicating a specific resource available at the API.

An API MAY define its own extensions, subject to the "type" of the request. It is assumed that the full structure of each of the authorization data elements is tailored to the needs of a certain application, API, or resource type. The example structures shown above are based on certain kinds of APIs that can be found in the Open Banking space.

Note: Applications MUST ensure that their authorization data types do not collide. This is either achieved by using a namespace under the control of the entity defining the type name or by registering the type with the new "OAuth Authorization Data Type Registry" (see Section 8).

The following example shows how an implementation could utilize the namespace "https://scheme.example.org/" to ensure collision resistant element names.

```
{
   "type":"https://scheme.example.org/files",
   "locations":[
      "https://example.com/files"
   ],
   "permissions":[
      {
         "path":"/myfiles/A",
         "access":[
            "read"
         ]
      },
      {
         "path":"/myfiles/A/X",
         "access":[
            "read",
            "write"
         ]
      }
   ]
}
```

### 2.2. Using "authorization_details"

The request parameter can be used anywhere where the "scope" parameter is used, examples include:

- Authorization requests as specified in [RFC6749],
- Access token requests as specified in [RFC6749],
- Request objects as specified in [I-D.ietf-oauth-jwsreq],
- Device Authorization Request as specified in [RFC8628].

Parameter encoding is determined by the respective context.

In the context of an authorization request according to [RFC6749], the parameter is encoded using the "application/x-www-form-urlencoded" format as shown in the following example (JSON string trimmed for brevity):

```
GET /authorize?response_type=code&client_id=s6BhdRkqt3
    &state=af0ifjsldkj
    &redirect_uri=https%3A%2F%2Fclient.example.org%2Fcb
    &code_challenge_method=S256
    &code_challenge=K2-ltc83acc4h0c9w6ESC_rEMTJ3bww-uCHaoeK1t8U
    &authorization_details=%5B%7B%22type%22%3A%22https%3A%2F%2F
    www.someorg.com%2Fpayment_initiation%22%2C%22actions%22%3A%5
    B%22initiate%22%2C%22status%22%2C%22cancel%22%5D%2C%22locat
    ions%22%3A%5B%22https%3A%2F%2Fexample.com%2Fpayments%22%5D%
    2C%22instructedAmount%22%3A%7B%22currency%22%3A%22EUR%22%2C
    %22amount%22%3A%22123.50%22%7D%2C%22debtorAccount%22%3A%7B%
    22iban%22%3A%22DE40100100103307118608%22%7D%2C%22creditorNa
    me%22%3A%22Merchant123%22%2C%22creditorAccount%22%3A%7B%22i
    ban%22%3A%22DE02100100109307118603%22%7D%2C%22remittanceInf
    ormationUnstructured%22%3A%22Ref%20Number%20Merchant%22%7D%
    5D%0A%20%20%20 HTTP/1.1
Host: server.example.com
```

In the context of a request object as specified in [I-D.ietf-oauth-jwsreq], "authorization_details" is added as another top level JSON element.

```
{
   "iss":"s6BhdRkqt3",
   "aud":"https://server.example.com",
   "response_type":"code",
   "client_id":"s6BhdRkqt3",
   "redirect_uri":"https://client.example.com/cb",
   "state":"af0ifjsldkj",
   "code_challenge_method":"S256",
   "code_challenge":"K2-ltc83acc4h0c9w6ESC_rEMTJ3bww-uCHaoeK1t8U",
   "authorization_details":[
     {
        "type": "https://www.someorg.com/payment_initiation",
        "actions": ["initiate", "status", "cancel"],
        "locations":[
          "https://example.com/payments"
        ],
        "instructedAmount":{
           "currency":"EUR",
           "amount":"123.50"
        },
        "debtorAccount":{
           "iban":"DE40100100103307118608"
        },
        "creditorName":"Merchant123",
        "creditorAccount":{
           "iban":"DE02100100109307118603"
        },
        "remittanceInformationUnstructured":"Ref Number Merchant"
     }
  ]
}
```

Note: Authorization request URIs containing authorization details in a request parameter or a request object can become very long. Implementers SHOULD therefore consider using the "request_uri" parameter as defined in [I-D.ietf-oauth-jwsreq], potentially in combination with the pushed request object mechanism as defined in [I-D.lodderstedt-oauth-par] to pass authorization details in a reliable and secure manner.

## 2.3. Authorization Request Processing

Based on the data provided in the "authorization_details" parameter the AS will ask the user for consent to the requested access permissions.

Note: The AS is supposed to merge the authorization requirements given in the "scope" parameter and the "authorization_details" parameter if both are present in the authorization request.

The AS MUST refuse to process any unknown authorization data type. If the "authorization_details" contains any unknown authorization data type, the AS MUST abort processing and respond with an error "invalid_scope" to the client.

If the resource owner grants the client the requested access, the AS will issue tokens to the client that are associated with the respective "authorization_details".

The AS MUST make the "authorization_details" available to the respective resource servers. The AS MAY add the "authorization_details" element to access tokens in JWT format and to Token Introspection responses.

The AS MUST take into consideration the privacy implications when sharing authorization details with the resource servers. The AS SHOULD share this data with the resource servers on a "need to know" basis.

### 2.4. Token Response

In addition to the token response parameters as defined in [RFC6749], the authorization server MUST also return the authorization details as granted by the resource owner and assigned to the respective access token.

This is shown in the following example:

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

{
  "access_token":"2YotnFZFEjr1zCsicMWpAA",
  "token_type":"example",
  "expires_in":3600,
  "refresh_token":"tGzv3JOkF0XG5Qx2TlKWIA",
  "authorization_details":[
    {
       "type": "https://www.someorg.com/payment_initiation",
       "actions": ["initiate", "status", "cancel"],
       "locations":[
         "https://example.com/payments"
       ],
       "instructedAmount":{
          "currency":"EUR",
          "amount":"123.50"
       },
       "debtorAccount":{
          "iban":"DE40100100103307118608"
       },
       "creditorName":"Merchant123",
       "creditorAccount":{
          "iban":"DE02100100109307118603"
       },
       "remittanceInformationUnstructured":"Ref Number Merchant"
    }
  ]
}
```

## 2.5. Relationship to "resource" parameter

[I-D.ietf-oauth-resource-indicators] defines the request parameter "resource" indicating to the AS the resource(s) where the client intends to use the access tokens issued based on a certain grant.

This mechanism is a way to audience-restrict access tokens and to allow the AS to create resource specific access tokens.

This draft can be used in conjunction with [I-D.ietf-oauth-resource-indicators] in the same way as the "scope" parameter. The AS is supposed to narrow down the authorization details and respective permissions to the needs of the particular resource when minting an access token.

This depends, however, on the AS to know what authorization details are relevant for what RS. The parameter introduced in this specification can also be combined with the concept of resource indicators to make this relationship explicit. This enables the AS to narrow down the privileges of an access token to specific permissions for individual operations on specific resources (see [I-D.ietf-oauth-security-topics], section-3.3).

The "locations" and the "identifier" elements together allow the AS to determine the resource a client wants to access as shown in following example:

```
[
   {
      "type": "https://scheme.example.org/storage":
      "locations":["https://storage.example.com"],
      "identifier":"/shared/group1",
      "actions":[
         "read"
      ]
   }
}
```

The AS MUST respect those values when deciding whether a certain element is placed into a (structured) access token or token introspection response.

## 3. Metadata

The AS advertises support for "authorization_details" using the metadata parameter "authorization_details_supported" of type boolean.

The authorization data types supported can be determined using the metadata parameter "authorization_data_types_supported", which is an JSON array.

Clients announce the authorization data types they use in the new dynamic client registration parameter "authorization_data_types".

The registration of new authorization data types with the AS is out of scope of this draft.

## 4. Implementation Considerations

The scheme and processing will vary significantly among different authorization data types. Any implementation of this draft is therefore supposed to allow the customization of the user consent and the handling of access token data.

One option would be to have a mechanism allowing the registration of extension modules, each of them responsible for rendering the respective user consent and any transformation needed to provide the data needed to the resource server by way of structured access tokens or token introspection responses.

## 5. Security Considerations

Authorization details are sent through the user agent in case of an OAuth authorization request, which makes them vulnerable to modifications by the user. In order to ensure their integrity, the client SHOULD send authorization details in a signed request object as defined in [I-D.ietf-oauth-jwsreq] or use the "request_uri" authorization request parameter as defined in [I-D.ietf-oauth-jwsreq] to pass the URI of the request object to the authorization server.

## 6. Privacy Considerations

Implementers MUST design and use authorization details in a privacy preserving manner.

Any sensitive personal data included in authorization details MUST be prevented from leaking, e.g., through referrer headers. Implementation options include encrypted request objects as defined in [I-D.ietf-oauth-jwsreq] or transmission of authorization details via end-to-end encrypted connections between client and authorization server by utilizing the "request_uri" authorization request parameter as defined in [I-D.ietf-oauth-jwsreq].

Even if the request data are encrypted, an attacker could use the authorization server to learn the user data by injecting the encrypted request data into an authorization request on a device under his control and use the authorization server's user consent screens to show the (decrypted) user data in the clear. Implementations MUST consider this attacker vector and implement appropriate counter measures, e.g. by only showing portions of the data or, if possible, determing whether the assumed user context is still the same (after user authentication).

## 7. Acknowledgements

We would would like to thank Daniel Fett, Sebastian Ebling, Dave Tonge, Mike Jones, Nat Sakimura, and Rob Otto for their valuable feedback during the preparation of this draft.

We would also like to thank Daniel Fett, Dave Tonge and Aaron Parecki for their valuable feedback to this draft.

## 8. IANA Considerations

TBD

- "authorization_details" as JWT claim
- "authorization_details_supported" and "authorization_data_types_supported" as metadata parameters
- "authorization_data_types" as dynamic client registration parameter
- establish authorization data type registry

## 9. References

### 9.1. 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>.

[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/info/rfc8174>.

[RFC8628] Denniss, W., Bradley, J., Jones, M., and H. Tschofenig, "OAuth 2.0 Device Authorization Grant", RFC 8628, DOI 10.17487/RFC8628, August 2019, <https://www.rfc-editor.org/info/rfc8628>.

### 9.2. Informative References

[I-D.ietf-oauth-jwsreq]
Sakimura, N. and J. Bradley, "The OAuth 2.0 Authorization Framework: JWT Secured Authorization Request (JAR)", draft-ietf-oauth-jwsreq-19 (work in progress), June 2019.

[I-D.ietf-oauth-resource-indicators]
Campbell, B., Bradley, J., and H. Tschofenig, "Resource Indicators for OAuth 2.0", draft-ietf-oauth-resource-indicators-08 (work in progress), September 2019.

[I-D.ietf-oauth-security-topics]
Lodderstedt, T., Bradley, J., Labunets, A., and D. Fett, "OAuth 2.0 Security Best Current Practice", draft-ietf-oauth-security-topics-13 (work in progress), July 2019.

[I-D.lodderstedt-oauth-par]
Lodderstedt, T., Campbell, B., Sakimura, N., Tonge, D., and F. Skokan, "OAuth 2.0 Pushed Authorization Requests", draft-lodderstedt-oauth-par-00 (work in progress), September 2019.

[transaction-authorization]
Lodderstedt, T., "Transaction Authorization or why we need to re-think OAuth scopes", Apr 2019, <https://medium.com/oauth-2/transaction-authorization-or-why-we-need-to-re-think-oauth-scopes-2326e2038948>.

## Appendix A. Document History

[[ To be removed from the final specification ]]

-02

- Added Security Considerations
- Added Privacy Considerations
- Added notes on URI size and authorization details
- Added requirement to return the effective authorization details granted by the resource owner in the token response
- changed "authorization_details" structure from object to array
- added Justin Richer & Brian Campbell as Co-Authors

-00 / -01

- first draft

## Authors' Addresses

Torsten Lodderstedt
yes.com

Email: torsten@lodderstedt.net

Justin Richer
Bespoke Engineering

Email: ietf@justin.richer.org

Brian Campbell
Ping Identity

Email: bcampbell@pingidentity.com