Workgroup: Web Authorization Protocol
Internet-Draft: draft-ietf-oauth-rar-05
Published: 15 May 2021
Intended Status: Standards Track
Expires: 16 November 2021
Authors: T. Lodderstedt
yes.com

J. Richer
Bespoke Engineering

B. Campbell
Ping Identity

OAuth 2.0 Rich Authorization Requests

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 16 November 2021.

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",
   "locations": [
      "https://example.com/payments"
   ],
   "instructedAmount": {
      "currency": "EUR",
      "amount": "123.50"
   },
   "creditorName": "Merchant123",
   "creditorAccount": {
      "iban": "DE02100100109307118603"
   },
   "remittanceInformationUnstructured": "Ref Number Merchant"
}
```

This object contains detailed information about the intended payment, such as amount, currency, and creditor, that are required to inform the user and obtain her consent. The AS and the respective RS (providing the payment initiation API) will together enforce this consent.

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

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

Most notably, the field locations allows a client to specify where it intends to use a certain authorization, i.e., it is now possible to unambiguously assign permissions to resource servers. In situations with multiple resource servers, this prevents unintended client authorizations (e.g. a read scope value potentially applicable for an email as well as a cloud service). In combination with the resource token request parameter as specified in [RFC8707] or by specifing authorization details with a single location only in the token request, it enables the AS to mint RS-specific structured access tokens that only contain the permissions applicable to the respective RS.

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, in JSON notation, an array of 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 using the payment authorization object shown above:

```
[
   {
      "type": "payment_initiation",
      "actions": [
         "initiate",
         "status",
         "cancel"
      ],
      "locations": [
         "https://example.com/payments"
      ],
      "instructedAmount": {
         "currency": "EUR",
         "amount": "123.50"
      },
      "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"
      ],
      "locations": [
         "https://example.com/accounts"
      ]
   },
   {
      "type": "payment_initiation",
      "actions": [
         "initiate",
         "status",
         "cancel"
      ],
      "locations": [
         "https://example.com/payments"
      ],
      "instructedAmount": {
         "currency": "EUR",
         "amount": "123.50"
      },
      "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

The allowable contents of the authorization details object are determined by the type parameter.

type:

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

This field MUST be compared using an exact byte match of the string value against known types by the AS. The AS MUST ensure that there is no collision between different authorization data types that it supports. The AS MUST NOT do any collation or normalization of data types during comparison.

The value of the type field determines the allowable contents of the object which contains it. 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. All data elements are OPTIONAL for use by a given API definition. The allowable values of all elements are determined by the API being protected.

locations: An array of strings representing the location of the resource or resource server. These strings are typically URIs identifying the location of the RS.

actions: An array of strings representing the kinds of actions to be taken at the resource.

datatypes: 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.

privileges: An array of strings representing the types or levels of privilege being requested at the resource.

When different element types are used in combination, the permissions the client requests is the cartesian product of the values. That is to say, the object represents a request for all action values listed within the object to be used at all locations values listed within the object for all datatype values listed within the object. In the following example, the client is requesting read and write access to both the contacts and photos belonging to customers in a customer_information API. If this request is granted, the client would assume it would be able to use any combination of rights defined by the API, such as reading the photos and writing the contacts.

```
[
   {
      "type": "customer_information",
      "locations": [
         "https://example.com/customers",
      ]
      "actions": [
         "read",
         "write"
      ],
      "datatypes": [
         "contacts",
         "photos"
      ]
   }
]
```

If the client wishes to have finer control over its access, it can send multiple objects. In this example, the client is asking for read access to the contacts and write access to the photos in the same API endpoint. If this request is granted, the client would not be able to write to the contacts.

```
[
   {
      "type": "customer_information",
      "locations": [
         "https://example.com/customers"
      ],
      "actions": [
         "read"
      ],
      "datatypes": [
         "contacts"
      ]
   },
   {
      "type": "customer_information",
      "locations": [
         "https://example.com/customers"
      ],
      "actions": [
         "write"
      ],
      "datatypes": [
         "photos"
      ]
   }
]
```

An API MAY define its own extensions, subject to the type of the respective authorization object. It is anticipated that API designers will use a combination of common fields defined in this specification as well as fields specific to the API itself. The following non-normative example shows the use of both common and API-specific fields as part of two different fictitious API type values. The first access request includes the actions, locations, and datatypes fields specified here as well as the API-specific geolocation field. The second access request includes the actions and identifier fields specified here as well as the API-specific currency field.

```
"resources": [
    {
        "type": "photo-api",
        "actions": [
            "read",
            "write"
        ],
        "locations": [
            "https://server.example.net/",
            "https://resource.local/other"
        ],
        "datatypes": [
            "metadata",
            "images"
        ],
        "geolocation": [
            { lat: -32.364, lng: 153.207 },
            { lat: -35.364, lng: 158.207 }
        ]
    },
    {
        "type": "financial-transaction",
        "actions": [
            "withdraw"
        ],
        "identifier": "account-14-32-32-3",
        "currency": "USD"
    }
]
```

If this request is approved, the resulting access token's access rights will be the union of the requested types of access for each of the two APIs, just as above.

2.2. Authorization Data Types

Interpretation of the value of the type parameter, and the object elements that the type parameter allows, is under the control of the AS. However, the value of the type parameter is also generally documented and intended to be used by developers, it is RECOMMENDED that API designers choose type values that are easily copied without ambiguity. For example, some glyphs have multiple unicode code points for the same visual character, and a developer could potentially type a different character depending than what the AS has defined. Possible means of reducing potential confusion are limiting the value to ASCII characters, providing a machine-readable listing of data type values, or instructing developers to copy and paste directly from documentation.

If an application or API is expected to be deployed across different servers, such as the case in an open standard, the API designer is RECOMMENDED to use a collision-resistant namespace under their control, such as a URI that the API designer controls.

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"
         ]
      }
   ]
}
```

3. Authorization Request

The authorization_details authorization request parameter can be used to specify authorization requirements in all places where the scope parameter is used for the same purpose, examples include:

- *Authorization requests as specified in [RFC6749],
- *Device Authorization Request as specified in [RFC8628],
- *Backchannel Authentication Requests as defined in [OpenID.CIBA].

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 of the serialized JSON as shown in the following using the example from Section 2 (line breaks for display purposes only):

```
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_rEMTJ3bwc-uCHaoeK1t8U
   &authorization_details=%5B%7B%22type%22%3A%22account%5Finfo
   rmation%22%2C%22actions%22%3A%5B%22list%5Faccounts%22%2C%22
   read%5Fbalances%22%2C%22read%5Ftransactions%22%5D%2C%22loca
   tions%22%3A%5B%22https%3A%2F%2Fexample%2Ecom%2Faccounts%22%
   5D%7D%2C%7B%22type%22%3A%22payment%5Finitiation%22%2C%22act
   ions%22%3A%5B%22initiate%22%2C%22status%22%2C%22cancel%22%5
   D%2C%22locations%22%3A%5B%22https%3A%2F%2Fexample%2Ecom%2Fp
   ayments%22%5D%2C%22instructedAmount%22%3A%7B%22currency%22%
   3A%22EUR%22%2C%22amount%22%3A%22123%2E50%22%7D%2C%22credito
   rName%22%3A%22Merchant123%22%2C%22creditorAccount%22%3A%7B%
   22iban%22%3A%22DE02100100109307118603%22%7D%2C%22remittance
   InformationUnstructured%22%3A%22RefNumberMerchant%22%7D%5D HTTP/1.1
Host: server.example.com
```

Based on the data provided in the authorization_details parameter the AS will ask the user for consent to the requested access permissions. In this example, the client wants to get access to account information and intiate a payment:

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

3.1. Relationship to "scope" parameter

authorization_details and scope can be used in the same authorization request for carrying independent authorization requirements.

The AS MUST consider both sets of requirements in combination with each other for the given authorization request. The details of how the AS combines these parameters are specific to the APIs being protected and outside the scope of this specification.

It is RECOMMENDED that a given API uses only one form of requirement specification.

When gathering user consent, the AS MUST present the merged set of requirements represented by the authorization request.

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 (and scope values, if applicable).

3.2. Relationship to "resource" parameter

The resource authorization request parameter as defined in [RFC8707] can be used to further determine the resources where the requested scope can be applied. The resource parameter does not have any impact on the way the AS processes the authorization_details authorization request parameter.

4. Authorization Response

This specification does not define extensions to the authorization response.

5. Authorization Error Response

The AS MUST refuse to process any unknown authorization data type or authorization details not conforming to the respective type definition. If any of the objects in authorization_details contains an unknown authorization data type or an object of known type but containing unknown elements or elements of the wrong type or elements with invalid values or if required elements are missing, the AS MUST abort processing and respond with an error invalid_authorization_details to the client.

6. Token Request

The authorization_details token request parameter can be used to specify the authorization details a client wants the AS to assign to an access token. The AS checks whether the underlying grant (in case of grant types authorization_code, refresh_token, ...) or the client's policy (in case of grant type client_credential) allows the issuance of an access token with the requested authorization details. Otherwise, the AS refuses the request with error code invalid_authorization_details (similar to invalid_scope).

6.1. Comparing authorization details

Many actions in the OAuth protocol allow the AS and RS to make security decisions based on whether or not the request is asking for "more" or "less" than a previous, existing request. For example, upon refreshing a token, the client can ask for a new access token with "fewer permissions" than had been previously authorized by the resource owner. Since the nature of an authorization details request

is based solely on the API or APIs that it is describing, there is not a simple means of comparing any two arbitrary authorization details requests. Authorization servers should not rely on simple object comparison in most cases, as the intersection of some elements within a request could have side effects in the access rights granted, depending on how the API has been designed and deployed. This is a similar effect to the scope values used with some APIs.

However, when comparing a new request to an existing request, authorization servers can use the same processing techniques as used in granting the request in the first place to determine if a resource owner needs to authorize the request. The details of this comparison are dependent on the definition of the type of authorization request and outside the scope of this specification, but common patterns can be applied.

This shall be illustrated using our running example. The example authorization request in Section 3, if approved by the user, resulted in the issuance of an authorization code associated with the privileges to

- list accounts
- access the balance of one or more accounts,
- access the transactions of one or more accounts, and
- to initiate a payment.

The client could now request the AS to issue an access token assigned with the privilege to just access a list of accounts as follows:

```
[
   {
      "type":"account_information",
      "actions":[
         "list_accounts"
      ],
      "locations":[
         "https://example.com/accounts"
      ]
   }
]
```

The example API is designed such that each field used by the account_information type contains additive rights, with each value

within the actions and locations arrays specifying a different element of access. To make a comparison in this instance, the AS would perform the following steps:

- compare that the authorization code issued in the previous step contains an authorization details object of type account_information
- compare whether the approved list of actions contains list_account, and
- whether the locations value includes only previously-approved locations.

If all checks succeed, the AS would issue the requested access token with the reduced set of access.

Note that this comparison is relevant to this specific API type definition. A different API type definition could have different processing rules. For example, the value of an action could subsume the rights associated with another action value. For example, if a client initially asks for a token with write access, which implies both read and write access to this API:

```
[
    {
        "type": "example_api",
        "actions": [
            "write"
        ]
    }
]
```

Later that same client makes a refresh request for read access:

```
[
    {
        "type": "example_api",
        "actions": [
            "read"
        ]
    }
]
```

The AS would compare the type value and the action value to determine that the read access is already covered by the write access previously granted to the client.

6.2. Interaction with the resource parameter

The resource token request parameter as defined in [RFC8707] MAY be used in the token request to request the creation of an audience restricted access token (as recommended in [I-D.ietf-oauth-security-topics]). If the client uses this parameter, the AS MUST consider the audience restriction defined by the locations elements of the authorization_details to filter the authorization data objects applicable to the respective resource(s).

The logic is as follows:

- For every authorization details object without a locations element: the authorization server treats it as applicable to all resources, i.e. it assigns this authorization details object to the access token.

- For every authorization details object with a locations element: the authorization server adds this object to the access token, if at least one of the locations values exactly matches the resource token request parameter value. The authorization server MUST compare both values using an exact byte match of the string values.

For example the following token request selects authorization details applicable for the resource server represented by the URI https://example.com/payments.

```
POST /token HTTP/1.1
Host: as.example.com
Authorization: Basic czZCaGRSa3F0MzpnWDFmQmF0M2JW
Content-Type: application/x-www-form-urlencoded

grant_type=authorization_code&code=SplxlOBeZQQYbYS6WxSbIA
&redirect_uri=https%3A%2F%2Fclient%2Eexample%2Ecom%2Fcb
&resource=https%3A%2F%2Fexample%2Ecom%2Fpayments
```

Using the example given above, this request would result in the assignment of the payment_initiation authorization details object from Section 2 to the access token to be issued (see below).

7. Token Response

The authorization details assigned to the access token issued in a token response are determined by the authorization_detail parameter of the corresponding token request as well as any related parameters such as resource and scope. If the client does not specify any of those token request parameters, the AS determines the resulting authorization details at its discretion.

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.

For our running example, this would look like this:

```
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"
         },
         "creditorName": "Merchant123",
         "creditorAccount": {
            "iban": "DE02100100109307118603"
         },
         "remittanceInformationUnstructured": "Ref Number Merchant"
      }
   ]
}
```

7.1. Enriched authorization details in Token Response

The authorization details attached to the access token MAY differ from what the client requests. In addition to the user authorizing less than what the client requested, there are use cases where the authorization server enriches the data in an authorization details object. For example, a client may ask for access to account information but leave the decision about the accounts it will be able to access to the user. The user would select the sub set of accounts they wants the client to entitle to access in the course of the authorization process. In order to allow the client to determine the accounts it is entitled to access, the authorization server will add this information to the respective authorization details object.

As an example, the requested authorization detail parameter could look like this:

```
"authorization_details": [
   {
      "type": "account_information",
      "access": {
         "accounts": [],
         "balances": [],
         "transactions": []
      },
      "recurringIndicator":true
   }
]
```

The authorization server then would expand the authorization details object and add the respective account identifiers.

```
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":"account_information",
         "access":{
            "accounts":[
               {
                  "iban":"DE23100100101234567 89"
               },
               {
                  "maskedPan":"123456xxxxxx1234"
               }
            ],
            "balances":[
               {
                  "iban":"DE23100100101234567 89"
               }
            ],
            "transactions":[
               {
                  "iban":"DE23100100101234567 89"
               },
               {
                  "maskedPan":"123456xxxxxx1234"
               }
            ]
         },
         "recurringIndicator":true
      }
   ]
}
```

For another example, the client is asking for access to a medical record but does not know the record number at request time. In this example, the client specifies the type of access it wants but doesn't specify the location or identifier of that access.

```
{
"authorization_details": [
   {
      "type": "medical_record",
      "sens": [ "HIV", "ETH", "MART" ],
      "actions": [ "read" ],
      "datatypes": [ "Patient", "Observation", "Appointment" ]
   }
]
```

When the user interacts with the AS, they select which of the medical records they are responsible for to give to the client. This information gets returned with the access token.

```
{
   "access_token":"2YotnFZFEjr1zCsicMWpAA",
   "token_type":"example",
   "expires_in":3600,
   "refresh_token":"tGzv3JokF0XG5Qx2TlKWIA",
   "authorization_details":[
    {
      "type": "medical_record",
      "sens": [ "HIV", "ETH", "MART" ],
      "actions": [ "read" ],
      "datatypes": [ "Patient", "Observation", "Appointment" ]
      "identifier": "patient-541235",
      "locations": [ "https://records.example.com/" ]
     }
  ]
}
```

Note: the client needs to be aware upfront of the possibility that a certain authorization details object can be enriched. It is assumned that this property is part of the definition of the respective authorization details type.

8. Token Error Response

The AS MUST refuse to process any unknown authorization data type or authorization details not conforming to the respective type definition. If any of the objects in authorization_details contains an unknown authorization data type or an object of known type but containing unknown elements or elements of the wrong type, elements with invalid values, or if required elements are missing, the AS

MUST abort processing and respond with an error invalid_authorization_details to the client.

9. Resource Servers

In order to enable the RS to enforce the authorization details as approved in the authorization process, the AS MUST make this data available to the RS. The AS MAY add the authorization_details element to access tokens in JWT format or to Token Introspection responses.

9.1. JWT-based Access Tokens

If the access token is a JWT [RFC7519], the AS is RECOMMENDED to add the authorization_details object, filtered to the specific audience, as top-level claim.

The AS will typically also add further claims to the JWT the RS requires for request processing, e.g., user id, roles, and transaction specific data. What claims the particular RS requires is defined by the RS-specific policy with the AS.

The following shows the contents of an example JWT for the payment initation example above:

```
{
   "iss": "https://as.example.com",
   "sub": "24400320",
   "aud": "a7AfcPcsl2",
   "exp": 1311281970,
   "acr": "psd2_sca",
   "txn": "8b4729cc-32e4-4370-8cf0-5796154d1296",
   "authorization_details": [
      {
         "type": "https://www.someorg.com/payment_initiation",
         "actions": [
            "initiate",
            "status",
            "cancel"
         ],
         "locations": [
            "https://example.com/payments"
         ],
         "instructedAmount": {
            "currency": "EUR",
            "amount": "123.50"
         },
         "creditorName": "Merchant123",
         "creditorAccount": {
            "iban": "DE02100100109307118603"
         },
         "remittanceInformationUnstructured": "Ref Number Merchant"
      }
   ],
   "debtorAccount": {
      "iban": "DE40100100103307118608",
      "user_role": "owner"
   }
}
```


In this case, the AS added the following example claims:

*sub: conveys the user on which behalf the client is asking for payment initation

*txn: transaction id used to trace the transaction across the services of provider example.com

*debtorAccount: API-specific element containing the debtor account. In the example, this account was not passed in the authorization details but selected by the user during the authorization process. The field user_role conveys the role the

user has with respect to this particuar account. In this case, they is the owner. This data is used for access control at the payment API (the RS).

9.2. Token Introspection

In case of opaque access tokens, the data provided to a certain RS is determined using the RS's identifier with the AS (see [I-D.ietf-oauth-jwt-introspection-response], section 3).

The token endpoint response provides the RS with the authorization details applicable to it as a top-level JSON element along with the claims the RS requires for request processing.

Here is an example for the payment initation example RS:

```
{
   "active": true,
   "sub": "24400320",
   "aud": "s6BhdRkqt3",
   "exp": 1311281970,
   "acr": "psd2_sca",
   "txn": "8b4729cc-32e4-4370-8cf0-5796154d1296",
   "authorization_details": [
      {
         "type": "https://www.someorg.com/payment_initiation",
         "actions": [
            "initiate",
            "status",
            "cancel"
         ],
         "locations": [
            "https://example.com/payments"
         ],
         "instructedAmount": {
            "currency": "EUR",
            "amount": "123.50"
         },
         "creditorName": "Merchant123",
         "creditorAccount": {
            "iban": "DE02100100109307118603"
         },
         "remittanceInformationUnstructured": "Ref Number Merchant"
      }
   ],
   "debtorAccount": {
      "iban": "DE40100100103307118608",
      "user_role": "owner"
   }
}
```


10. Metadata

The AS publishes the list of authorization details types it supports using the metadata parameter authorization_details_types_supported, which is a JSON array.

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

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

11. Scope value "openid" and "claims" parameter

OpenID Connect [OIDC] specifies the JSON-based claims request parameter that can be used to specify the claims a client (acting as OpenID Connect Relying Party) wishes to receive in a fine-grained and privacy preserving way as well as assign those claims to a certain delivery mechanisms, i.e. ID Token or userinfo response.

The combination of the scope value openid and the additional parameter claims can be used beside authorization_details in the same way as every non-OIDC scope value.

Alternatively, there could be an authorization data type for OpenID Connect. Appendix A.1 gives an example of what such an authorization data type could look like.

12. Implementation Considerations

12.1. Using authorization details in a certain deployment

Using authorization details in a certain deployment will require the follwowing steps:

- Define authorization details types
- Publish authorization details types in the OAuth server metadata
- Determine how authorization details are shown to the user in the user consent
- (if needed) Enrich authorization details in the user consent process (e.g. add selected accounts or set expirations)
- (if needed) Determine how authorization details are reflected in access token content or introspection responses
- Determine how the resource server(s) process(s) the authorization details or token data derived from authorization details

12.2. Minimal product support

Products supporting this specification should provide the following basic functions:

- Support advertisement of supported authorization details types in OAuth server metadata
- Accept authorization_details parameter in authorization requests including basic syntax check for compliance with this specification

- Support storage of consented authorization details as part of a grant

- Implement default behavior for adding authorization details to access tokens and token introspection responses in order to make them available to resource servers (similar to scope values). This should work with any grant type, especially authorization_code and refresh_token.

- If the product supports resource indicators, it should also support filtering of the authorization details to be assigned to access tokens using the resource token request parameter.

Processing and presentation of authorization details will vary significantly among different authorization data types. Products should therefore support customization of the respective behavior. In particular products should

- allow deployments to determine presentation of the authorization details

- allow deployments to modify requested authorization details in the user consent process, e.g. adding fields

- allow deployments to merge requested and pre-existing authorization details

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.

12.3. Use of Machine-readable Type Schemas

Products might allow deployments to use machine-readable schema languages for defining authorization details types to facilitate creating and validating authorization details objects against such schemas. For example, if an authorization details type were defined using JSON Schemas [JSON.Schema], the JSON schema id could be used as type value in the respective authorization details objects.

Note however that type values are identifiers understood by the AS and, to the extent necessary, the client and RS. This specification makes no assumption that a type value point to a machine-readable schema format, or that any party in the system (such as the client, AS, or RS) dereference or process the contents of the type field in any specific way.

12.4. Large requests

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] in combination with the pushed request object mechanism as defined in [I-D.ietf-oauth-par] to pass authorization details in a reliable and secure manner. Here is an example of such a pushed authorization request that sends the authorization request data directly to the AS via a HTTPS-protected connection:

```
POST /as/par HTTP/1.1
Host: as.example.com
Content-Type: application/x-www-form-urlencoded
Authorization: Basic czZCaGRSa3F0Mzo3RmpmcDBaQnIxS3REUmJuZlZkbUl3

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_rEMTJ3bwc-uCHaoeK1t8U
&authorization_details=%5B%7B%22type%22%3A%22account_information%22
%2C%22actions%22%3A%5B%22list_accounts%22%2C%22read_balances%22%2C%
22read_transactions%22%5D%2C%22locations%22%3A%5B%22https%3A%2F%2Fe
xample.com%2Faccounts%22%5D%7D%2C%7B%22type%22%3A%22payment_initiat
ion%22%2C%22actions%22%3A%5B%22initiate%22%2C%22status%22%2C%22canc
el%22%5D%2C%22locations%22%3A%5B%22https%3A%2F%2Fexample.com%2Fpaym
ents%22%5D%2C%22instructedAmount%22%3A%7B%22currency%22%3A%22EUR%22
%2C%22amount%22%3A%22123.50%22%7D%2C%22creditorName%22%3A%22Merchan
t123%22%2C%22creditorAccount%22%3A%7B%22iban%22%3A%22DE021001001093
07118603%22%7D%2C%22remittanceInformationUnstructured%22%3A%22Ref%2
0Number%20Merchant%22%7D%5D
```

13. 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] in conjunction with [I-D.ietf-oauth-par] to pass the URI of the request object to the authorization server.

All strings MUST be compared using the exact byte representation of the characters as defined by [RFC8259]. This is especially true for the type field, which dictates which other fields and functions are

allowed in the request. The server MUST NOT perform any form of collation, transformation, or equivalence on the string values.

14. 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 is 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).

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.

15. 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 Vladimir Dzhuvinov, Takahiko Kawasaki, Daniel Fett, Dave Tonge, Travis Spencer, Jørgen Binningsbø, Aamund Bremer, Steinar Noem, Francis Pouatcha, and Aaron Parecki for their valuable feedback to this draft.

16. IANA Considerations

16.1. JSON Web Token Claims Registration

This specification requests registration of the following value in the IANA "JSON Web Token Claims Registry" established by [RFC7519].

Claim Name: authorization_details

Claim Description: The request parameter authorization_details contains, in JSON notation, an array of objects. Each JSON object contains the data to specify the authorization requirements for a certain type of resource.
Change Controller: IESG
Specification Document(s): Section 2 of this document

16.2. OAuth Authorization Server Metadata

This specification requests registration of the following values in the IANA "OAuth Authorization Server Metadata" registry of [IANA.OAuth.Parameters] established by [RFC8414].

Metadata Name: authorization_details_types_supported
Metadata Description: JSON array containing the authorization details types the AS supports
Change Controller: IESG
Specification Document(s): Section 10 of [[this document]]

16.3. OAuth Dynamic Client Registration Metadata

This specification requests registration of the following value in the IANA "OAuth Dynamic Client Registration Metadata" registry of [IANA.OAuth.Parameters] established by [RFC7591].

Metadata Name: authorization_details_types
Metadata Description: Indicates what authorization details types the client uses.
Change Controller: IESG
Specification Document(s): Section 10 of [[this document]]

16.4. OAuth Extensions Error registry

This specification requests registration of the following value in the IANA "OAuth Extensions Error registry" registry of [IANA.OAuth.Parameters] established by [RFC6749].

Metadata Name: invalid_authorization_details
Metadata Description: indicates invalid authorization_details_parameterto the client.
Change Controller: IESG
Specification Document(s): Section 5 of [[this document]]

17. Normative References

[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/info/rfc8707>.

[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>.

[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>.

[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>.

[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>.

[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>.

18. Informative References

[I-D.ietf-oauth-jwsreq] Sakimura, N., Bradley, J., and M. B. Jones, "The OAuth 2.0 Authorization Framework: JWT Secured Authorization Request (JAR)", Work in Progress, Internet-Draft, draft-ietf-oauth-jwsreq-34, 8 April 2021, <https://tools.ietf.org/html/draft-ietf-oauth-jwsreq-34>.

[I-D.ietf-oauth-par] Lodderstedt, T., Campbell, B., Sakimura, N., Tonge, D., and F. Skokan, "OAuth 2.0 Pushed Authorization Requests", Work in Progress, Internet-Draft, draft-ietf-oauth-par-07, 12 April 2021, <https://tools.ietf.org/html/draft-ietf-oauth-par-07>.

[JSON.Schema] json-schema.org, "JSON Schema", <https://json-schema.org/>.

[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>.

[RFC7591] Richer, J., Ed., Jones, M., Bradley, J., Machulak, M., and P. Hunt, "OAuth 2.0 Dynamic Client Registration Protocol", RFC 7591, DOI 10.17487/RFC7591, July 2015, <https://www.rfc-editor.org/info/rfc7591>.

[CSC] Consortium, C. S., "Architectures and protocols for remote signature applications", 1 June 2019, <https://cloudsignatureconsortium.org/wp-content/uploads/2019/07/CSC_API_V1_1.0.4.0.pdf>.

[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>.

[I-D.ietf-oauth-security-topics] Lodderstedt, T., Bradley, J., Labunets, A., and D. Fett, "OAuth 2.0 Security Best Current Practice", Work in Progress, Internet-Draft, draft-ietf-oauth-security-topics-18, 13 April 2021, <https://tools.ietf.org/html/draft-ietf-oauth-security-topics-18>.

[ETSI] ETSI, "ETSI TS 119 432, Electronic Signatures and Infrastructures (ESI); Protocols for remote digital signature creation", 20 March 2019, <https://www.etsi.org/deliver/etsi_ts/119400_119499/119432/01.01.01_60/ts_119432v010101p.pdf>.

[transaction-authorization] Lodderstedt, T., "Transaction Authorization or why we need to re-think OAuth scopes", 20 April 2019, <https://medium.com/oauth-2/transaction-authorization-or-why-we-need-to-re-think-oauth-scopes-2326e2038948>.

[OpenID.CIBA] Fernandez, G., Walter, F., Nennker, A., Tonge, D., and B. Campbell, "OpenID Connect Client Initiated Backchannel Authentication Flow - Core 1.0", 16 January 2019, <https://openid.net/specs/openid-client-initiated-backchannel-authentication-core-1_0.html>.

[I-D.ietf-oauth-jwt-introspection-response] Lodderstedt, T. and V. Dzhuvinov, "JWT Response for OAuth Token Introspection", Work in Progress, Internet-Draft, draft-ietf-oauth-jwt-introspection-response-10, 18 October 2020, <https://tools.ietf.org/html/draft-ietf-oauth-jwt-introspection-response-10>.

[OIDC] Sakimura, N., Bradley, J., Jones, M., de Medeiros, B., and C. Mortimore, "OpenID Connect Core 1.0 incorporating errata set 1", 8 November 2014, <http://openid.net/specs/openid-connect-core-1_0.html>.

[IANA.OAuth.Parameters] IANA, "OAuth Parameters", <http://www.iana.org/assignments/oauth-parameters>.

Appendix A. Additional Examples

A.1. OpenID Connect

These hypothetical examples try to encapsulate all details specific to the OpenID Connect part of an authorization process into an authorization JSON object.

The top-level elements are based on the definitions given in [OIDC]:

- *claim_sets: names of predefined claim sets, replacement for respective scope values, such as profile

- *max_age: Maximum Authentication Age

- *acr_values: array of ACR values

- *claims: the claims JSON structure as defined in [OIDC]

This is a simple request for some claim sets.

```
[
   {
      "type": "openid",
      "locations": [
         "https://op.example.com/userinfo"
      ],
      "claim_sets": [
         "email",
         "profile"
      ]
   }
]
```

Note: locations specifies the location of the userinfo endpoint since this is the only place where an access token is used by a client (RP) in OpenID Connect to obtain claims.

A more sophisticated example is shown in the following

```
[
   {
      "type": "openid",
      "locations": [
         "https://op.example.com/userinfo"
      ],
      "max_age": 86400,
      "acr_values": "urn:mace:incommon:iap:silver",
      "claims": {
         "userinfo": {
            "given_name": {
               "essential": true
            },
            "nickname": null,
            "email": {
               "essential": true
            },
            "email_verified": {
               "essential": true
            },
            "picture": null,
            "http://example.info/claims/groups": null
         },
         "id_token": {
            "auth_time": {
               "essential": true
            }
         }
      }
   }
]
```

A.2. Remote Electronic Signing

The following example is based on the concept layed out for remote electronic signing in ETSI TS 119 432 [ETSI] and the CSC API for remote signature creation [CSC].

```
[
   {
      "type": "sign",
      "locations": [
         "https://signing.example.com/signdoc"
      ],
      "credentialID": "60916d31-932e-4820-ba82-1fcead1c9ea3",
      "documentDigests": [
         {
            "hash": "sTOgwOm+474gFj0q0x1iSNspKqbcse4IeiqlDg/HWuI=",
            "label": "Credit Contract"
         },
         {
            "hash": "HZQzZmMAIWekfGH0/ZKW1nsdt0xg3H6bZYztgsMTLw0=",
            "label": "Contract Payment Protection Insurance"
         }
      ],
      "hashAlgorithmOID": "2.16.840.1.101.3.4.2.1"
   }
]
```

The top-level elements have the following meaning:

- credentialID: identifier of the certificate to be used for signing

- documentDigests: array containing the hash of every document to be signed (hash elements). Additionally, the corresponding label element identifies the respective document to the user, e.g. to be used in user consent.

- hashAlgorithm: algomrithm that was used to calculate the hash values.

The AS is supposed to ask the user for consent for the creation of signatues for the documents listed in the structure. The client uses the access token issued as result of the process to call the sign doc endpoint at the respective signing service to actually create the signature. This access token is bound to the client, the user id and the hashes (and signature algorithm) as consented by the user.

A.3. Access to Tax Data

This example is inspired by an API allowing third parties to access citizen's tax declarations and income statements, for example to determine their credit worthiness.

```
[
    {
        "type": "tax_data",
        "locations": [
            "https://taxservice.govehub.no"
        ],
        "actions":"read_tax_declaration",
        "periods": ["2018"],
        "duration_of_access": 30,
        "tax_payer_id": "23674185438934"
    }
]
```

The top-level elements have the following meaning:

- *periods: determines the periods the client wants to access

- *duration_of_access: how long does the client intend to access the data in days

- *tax_payer_id: identifier of the tax payer (if known to the client)

A.4. eHealth

These two examples are inspired by requirements for APIs used in the Norwegian eHealth system.

In this use case the physical therapist sits in front of her computer using a local Electronic Health Records (EHR) system. They wants to look at the electronic patient records of a certain patient and they also wants to fetch the patients journal entries in another system, perhaps at another institution or a national service. Access to this data is provided by an API.

The information necessary to authorize the request at the API is only known by the EHR system, and must be presented to the API.

In the first example the authorization details object contains the identifier of an organization. In this case the API needs to know if the given organization has the lawful basis for processing personal health information to give access to sensitive data.

```
"authorization_details":{
    "type":"patient_record",
    "requesting_entity": {
        "type": "Practitioner",
        "identifier": [
        {
            "system": " urn:oid:2.16.578.1.12.4.1.4.4",
            "value": "1234567"
        }],
        "practitioner_role":{
            "organization":{
                "identifier": {
                    "system":"urn:oid:2.16.578.1.12.4.1.2.101",
                    "type":"ENH",
                    "value":"[organizational number]"
                }
            }
        }
    }
}
```

In the second example the API requires more information to authorize the request. In this case the authorization details object contains additional information about the health institution and the current profession the user has at the time of the request. The additional level of detail could be used for both authorization and data minimization.

```
[
   {
      "type": "patient_record",
      "location": "https://fhir.example.com/patient",
      "actions": [
         "read"
      ],
      "patient_identifier": [
         {
            "system": "urn:oid:2.16.578.1.12.4.1.4.1",
            "value": "12345678901"
         }
      ],
      "reason_for_request": "Clinical treatment",
      "requesting_entity": {
         "type": "Practitioner",
         "identifier": [
            {
               "system": " urn:oid:2.16.578.1.12.4.1.4.4",
               "value": "1234567"
            }
         ],
         "practitioner_role": {
            "organization": {
               "identifier": [
                  {
                     "system": "urn:oid:2.16.578.1.12.4.1.2.101",
                     "type": "ENH",
                     "value": "<organizational number>"
                  }
               ],
               "type": {
                  "coding": [
                     {
                        "system":
                           "http://hl7.org/fhir/organization-type",
                        "code": "dept",
                        "display": "Hospital Department"
                     }
                  ]
               },
               "name": "Akuttmottak"
            },
            "profession": {
               "coding": [
                  {
                     "system": "http://snomed.info/sct",
                     "code": "36682004",
                     "display": "Physical therapist"
```

```
                }
             ]
          }
       }
    }
]
```

Description of the elements:

- *patient_identifier: the identifier of the patient composed of a system identifier in OID format (namespace) and the acutal value within this namespace.

- *reason_for_request: the reason why the user wants to access a certain API

- *requesting_entity: specification of the requester by means of identity, role and organizational context. This data is provided to facilitate authorization and for auditing purposes.

In this use case, the AS authenticates the requester, who is not the patient, and approves access based on policies.

Appendix B. Document History

[[To be removed from the final specification]]

-05

- *added authorization_details token request parameter and discussion on authorization details comparison

- *added privileges field to authorization details (to align with GNAP)

- *added IANA text and changed metadata parameter names

- *added text about use of machine-readable type schemas, e.g JSON Schema

- *added text on how authorization details are determined for access token issued with token response

- *added token error response and further error conditions to authorization error response

-04

* restructured draft for better readability

* simplified normative text about use of the resource parameter with authorization_details

* added implementation considerations for deployments and products

* added type union language from GNAP

* added recommendation to use PAR to cope with large requests and for request protection

-03

* Updated references to current revisions or RFC numbers

* Added section about enrichment of authorization details objects by the AS

* Clarified processing of unknown authorization details parameters

* clarified dependencies between resource and authorization_details parameters

-02

* Clarify "type" parameter processing

-01

* Minor fix-up in a few examples

-00 (WG draft)

* initial WG revision

-03

* Reworked examples to illustrate privacy preserving use of authorization_details

* Added text on audience restriction

* Added description of relationship between scope and authorization_details

* Added text on token request & response and authorization_details

*Added text on how authorization details are conveyed to RSs by JWTs or token endpoint response

*Added description of relationship between claims and authorization_details

*Added more example from different sectors

*Clarified string comparison to be byte-exact without collation

-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