Workgroup: Web Authorization Protocol
Internet-Draft: draft-ietf-oauth-rar-02
Published: 21 August 2020
Intended Status: Standards Track
Expires: 22 February 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 22 February 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 [I-D.ietf-oauth-resource-indicators] 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 resource request 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.

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.

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.

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.

When different element types are used in combination, the permissions the client requests is the cartesian product of the values. In the following example

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

the client is requesting read and write access to both the contacts and photos belonging to customers in a customer information API. If the client wishes to have finer control over its access, it can send multiple objects. For example:

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

The client is asking for read access to the contacts and write access to the photos in the same API endpoint.

An API MAY define its own extensions, subject to the type of the respective authorization object. It is assumed that the full structure of each of the authorization objects is tailored to the needs of a certain application, API, or resource type, and can contain a mix of general-purpose and api-specific elements within the structure. The example structures shown above are based on certain kinds of APIs that can be found in the Open Banking space.

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"
         ]
      }
   ]
}
```

2.3. 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 use only one form of requirement specification.

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

2.3.1. 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 how such an authorization data type could look like.

2.4. Relationship to "resource" parameter

The request parameter resource as defined in [I-D.ietf-oauth-resource-indicators] indicates 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 server specific access tokens.

If a client uses authorization_details with locations elements and the resource parameter in the same authorization request, the locations data take precedence over the data conveyed in the resource parameter for that particular authorization details object.

If such a client uses the resource parameter in a subsequent token requests, the AS MUST utilize the data provided in the locations elements to filter the authorization data objects applicable to the

respective resource server. The AS will select all authorization details object where the resource string matches as prefix of one of the URLs provided in the respective locations element.

This shall be illustrated using an example.

The client has sent an authorization request using the following example authorization details.

```
[
   {
      "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"
   }
]
```

If this client then sends the following token request to the AS,

```
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
```

that contains a resource parameter with the value of https://example.com/payments, this value will be matched against the locations elements (https://example.com/accounts and https://example.com/payments) and will select the element of type payment_initiation for inclusion in the access token as illustrated by the following example JWT content.

```
{
   "iss": "https://as.example.com",
   "sub": "24400320",
   "aud": "a7AfcPcsl2",
   "exp": 1311281970,
   ...
   "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"
      }
   ],
   ...
}
```

3. Using "authorization_details"

3.1. Authorization Request

The 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],
- Access token requests as specified in [RFC6749], if also used as authorization requests, e.g. in the case of assertion grant types [RFC7521],
- Request objects as specified in [I-D.ietf-oauth-jwsreq],
- 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 example:

```
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%5Finformati
   on%22%2C%22actions%22%3A%5B%22list%5Faccounts%22%2C%22read%5Fbal
   ances%22%2C%22read%5Ftransactions%22%5D%2C%22locations%22%3A%5B%
   22https%3A%2F%2Fexample%2Ecom%2Faccounts%22%5D%7D%5D HTTP/1.1
Host: server.example.com
```

Implementors MUST ensure to protect personal identifiable information in transit. One way is to utilize encrypted request objects as defined in [I-D.ietf-oauth-jwsreq]. In the context of a request object, 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_rEMTJ3bwc-uCHaoeK1t8U",
   "authorization_details": [
      {
         "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"
      }
   ]
}
```


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.lodderstedt-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
```

3.2. 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.

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

Note: If the authorization request also contained the scope parameter, the AS MUST present the merged set of requirements represented by the authorization request in the user consent.

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).

Note: 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 (see below).

3.3. Token Request

Clients utilizing authorization details are RECOMMENDED to use the resource token request parameter to allow the AS to issue audience restricted access tokens as recommended in [I-D.ietf-oauth-security-topics].

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
```

3.4. 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"
         },
         "creditorName": "Merchant123",
         "creditorAccount": {
            "iban": "DE02100100109307118603"
         },
         "remittanceInformationUnstructured": "Ref Number Merchant"
      }
   ]
}
```

3.4.1. Token Content

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.

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, she is the owner. This data is used for access control at the payment API (the RS).

3.5. Token Introspection Request

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).

3.6. Token Introspection Response

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"
   }
}
```

4. 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.

5. 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.

6. 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.

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.

7. 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).

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.

8. 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, Travis Spencer, Jørgen Binningsbø, Aamund Bremer, Steinar Noem, and Aaron Parecki for their valuable feedback to this draft.

9. 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

*[[possibly establish authorization data type registry (and declare: type, actions, locations, datatypes, identifier, others?)]]

*[[register type openid_claims on a URL by the OpenID foundation?]]

10. 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>.

[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/info/rfc7521>.

[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>.

[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>.

[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>.

[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>.

11. Informative References

[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-15, 5 April 2020, <https://tools.ietf.org/html/draft-ietf-oauth-security-topics-15>.

[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>.

[I-D.lodderstedt-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-lodderstedt-oauth-par-01, 3 November 2019, <https://tools.ietf.org/html/draft-lodderstedt-oauth-par-01>.

[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>.

[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>.

[I-D.ietf-oauth-jwsreq] Sakimura, N. and J. Bradley, "The OAuth 2.0 Authorization Framework: JWT Secured Authorization Request (JAR)", Work in Progress, Internet-Draft, draft-ietf-oauth-jwsreq-26, 27 July 2020, <https://tools.ietf.org/html/draft-ietf-oauth-jwsreq-26>.

[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-09, 25 April 2020, <https://tools.ietf.org/html/draft-ietf-oauth-jwt-introspection-response-09>.

[I-D.ietf-oauth-resource-indicators] Campbell, B., Bradley, J., and H. Tschofenig, "Resource Indicators for OAuth 2.0", Work in Progress, Internet-Draft, draft-ietf-oauth-resource-indicators-08, 11 September 2019, <https://tools.ietf.org/html/draft-ietf-oauth-resource-indicators-08>.

[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>.

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. She wants to look at the electronic patient records of a certain patient and she 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]]

-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