Stream: Internet Engineering Task Force (IETF)
RFC: 9396
Category: Standards Track
Published: May 2023
ISSN: 2070-1721

Authors: T. Lodderstedt *yes.com* J. Richer *Bespoke Engineering* B. Campbell *Ping Identity*

# RFC 9396
# 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 OAuth messages.

## Status of This Memo

This is an Internet Standards Track document.

This document is a product of the Internet Engineering Task Force (IETF). It represents the consensus of the IETF community. It has received public review and has been approved for publication by the Internet Engineering Steering Group (IESG). Further information on Internet Standards is available in Section 2 of RFC 7841.

Information about the current status of this document, any errata, and how to provide feedback on it may be obtained at https://www.rfc-editor.org/info/rfc9396.

## Copyright Notice

## Table of Contents

1. Introduction

 1.1. Conventions and Terminology

2. Request Parameter "authorization_details"

 2.1. Authorization Details Types

 2.2. Common Data Fields

3. Authorization Request

 3.1. Relationship to the "scope" Parameter

 3.2. Relationship to the "resource" Parameter

4. Authorization Response

5. Authorization Error Response

6. Token Request

 6.1. Comparing Authorization Details

7. Token Response

 7.1. Enriched Authorization Details in Token Response

8. Token Error Response

9. Resource Servers

 9.1. JWT-Based Access Tokens

 9.2. Token Introspection

10. Metadata

11. Implementation Considerations

 11.1. Using Authorization Details in a Certain Deployment

 11.2. Minimal Implementation Support

 11.3. Use of Machine-Readable Type Schemas

 11.4. Large Requests

12. Security Considerations

13. Privacy Considerations

14. IANA Considerations

14.1. OAuth Parameters Registration

14.2. JSON Web Token Claims Registration

14.3. OAuth Token Introspection Response Registration

14.4. OAuth Authorization Server Metadata Registration

14.5. OAuth Dynamic Client Registration Metadata Registration

14.6. OAuth Extensions Error Registration

15. References

15.1. Normative References

15.2. Informative References

Appendix A. Additional Examples

A.1. OpenID Connect

A.2. Remote Electronic Signing

A.3. Access to Tax Data

A.4. eHealth

Acknowledgements

Authors' Addresses

# 1. Introduction

"The OAuth 2.0 Authorization Framework" [RFC6749] defines the `scope` parameter that allows OAuth clients to specify the requested scope, i.e., the limited capability, 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." However, it is not sufficient to specify fine-grained authorization requirements, such as "please let me transfer an amount of 45 Euros to Merchant A" or "please give me read access to directory A and write access to file X."

This specification introduces a new parameter `authorization_details` that allows clients to specify their fine-grained authorization requirements using the expressiveness of JSON [RFC8259] data structures.

For example, an authorization request for a credit transfer (designated as "payment initiation" in several open banking initiatives) can be represented using a JSON object like this:

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

*Figure 1: Example of an Authorization Request for a Credit Transfer*

This object contains detailed information about the intended payment, such as amount, currency, and creditor, that is required to inform the user and obtain their consent. The authorization server (AS) and the respective resource server (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-Auth].

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

### 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" (AS), "resource server" (RS), "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, which is defined as follows:

type: An identifier for the authorization details type as a string. The value of the type field determines the allowable contents of the object that contains it. The value is unique for the described API in the context of the AS. This field is **REQUIRED**.

An authorization_details array **MAY** contain multiple entries of the same type.

Figure 2 shows an authorization_details of type payment_initiation using the example data shown above:

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

*Figure 2: Example of "authorization_details" for a Credit Transfer*

Figure 3 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": "Merchant A",
      "creditorAccount": {
         "iban": "DE02100100109307118603"
      },
      "remittanceInformationUnstructured": "Ref Number Merchant"
   }
]
```

*Figure 3: Example of "authorization_details" for a Combined Request*

The JSON objects with `type` fields of `account_information` and `payment_initiation` represent the different `authorization_details` to be used by the AS to ask for consent.

> Note: The AS will make this data subsequently available to the respective RSs (see Section 9).

## 2.1. Authorization Details Types

The AS controls the interpretation of the value of the `type` parameter as well as the object fields that the `type` parameter allows. 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 than what the AS has defined. Possible means of reducing potential confusion are limiting the value to ASCII [RFC0020] characters, providing a machine-readable listing of data type values, or instructing developers to copy and paste directly from the 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 type values.

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

*Figure 4: Example of "authorization_details" with a URL as Type Identifier*

## 2.2. Common Data Fields

This specification defines a set of common data fields that are designed to be usable across different types of APIs. This specification does not require the use of these common fields by an API definition but, instead, provides them as reusable generic components for API designers to make use of. The allowable values of all fields are determined by the API being protected, as defined by a particular "type" value.

`locations`: An array of strings representing the location of the resource or RS. These strings are typically URIs identifying the location of the RS. This field can allow a client to specify a particular RS, as discussed in Section 12.

`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 common data fields are used in combination, the permissions the client requests are the product of all the values. The object represents a request for all `actions` values listed within the object to be used at all `locations` values listed within the object for all `datatypes` 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 read access to the photos and write access to the contacts.

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

*Figure 5: Example of "authorization_details" with Common Data Fields*

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"
      ]
   }
]
```

*Figure 6: Example of "authorization_details" with Common Data Fields in Multiple Objects*

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 data 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, indicating access to photos taken at the given coordinates. The second access request includes the `actions` and `identifier` fields specified here as well as the API-specific `currency` fields.

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

*Figure 7: Example of "authorization_details" Using Common and Extension Data Fields*

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.

## 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 requests as specified in [RFC8628]
- backchannel authentication requests as defined in [OID-CIBA]

In case of authorization requests as defined in [RFC6749], implementers **MAY** consider using pushed authorization requests [RFC9126] to improve the security, privacy, and reliability of the flow. See Sections 12, 13, and 11.4 for details.

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 Figure 8, 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%22Merchant%20A%22%2C%22creditorAccount%22%3A%7B
   %22iban%22%3A%22DE02100100109307118603%22%7D%2C%22remittanc
   eInformationUnstructured%22%3A%22Ref%20Number%20Merchant%22
   %7D%5D HTTP/1.1
Host: server.example.com
```

*Figure 8: Example of Authorization Request with "authorization_details"*

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

> Note: The user may also grant a subset of the requested authorization details.

In Figure 9, the client wants to get access to account information and 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": "Merchant A",
      "creditorAccount": {
         "iban": "DE02100100109307118603"
      },
      "remittanceInformationUnstructured": "Ref Number Merchant"
   }
]
```

*Figure 9: URL Decoded "authorization_details"*

## 3.1. Relationship to the "scope" Parameter

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

Combined use of `authorization_details` and `scope` is supported by this specification in part to allow existing OAuth-based applications to incrementally migrate towards using `authorization_details` exclusively. It is **RECOMMENDED** that a given API use only one form of requirement specification.

The AS **MUST** process 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.

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 the "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 details type or authorization details not conforming to the respective type definition. The AS **MUST** abort processing and respond with an error invalid_authorization_details to the client if any of the following are true of the objects in the authorization_details structure:

- contains an unknown authorization details type value,
- is an object of known type but containing unknown fields,
- contains fields of the wrong type for the authorization details type,
- contains fields with invalid values for the authorization details type, or
- is missing required fields for the authorization details type.

## 6. Token Request

The authorization_details token request parameter can be used to specify the authorization details that 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, etc.) or the client's policy (in case of grant type client_credentials) allows the issuance of an access token with the requested authorization details. Otherwise, the AS refuses the request with the 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 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. The requested access token will convey the reduced permissions, but the resource owner's previous authorization is unchanged by such requests. Since the semantics of the fields in the authorization_details will be implementation specific to a given API or set of APIs, there is no standardized mechanism to compare two arbitrary authorization detail requests. An AS should not rely on simple object comparison in most cases, as the intersection of some fields within a request could have side effects on 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.

When comparing a new request to an existing request, an AS 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
- initiate, check the status of, and cancel 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"
      ]
   }
]
```

*Figure 10: Example of "authorization_details" Reduced Privileges*

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:

- verify that the authorization code issued in the previous step contains an authorization details object of type `account_information`,
- verify whether the approved list of actions contains `list_accounts`, and
- verify 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, an `actions` value could subsume the rights associated with another `actions` value. For example, if a client initially asks for a token with `write` access, this implies both read and write access to this API:

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

*Figure 11: Example of "authorization_details" Requesting "write" Access to an API*

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

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

*Figure 12: Example of "authorization_details" Requesting "read" Access to an API*

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

This same API could be designed with a possible value for `privileges` of `admin`, used in this example to denote that the resulting token is allowed to perform any of the functions on the resources. If that client is then granted such `admin` privileges to the API, the `authorization_details` would be as follows:

```
[
    {
        "type": "example_api",
        "privileges": [
            "admin"
        ]
    }
]
```

*Figure 13: Example of "authorization_details" with "admin" Access to an API*

The AS would compare the `type` value and find that the `privileges` value subsumes any aspects of `read` or `write` access that had been granted to the client previously. Note that other API definitions can use `privileges` such that values do not subsume one another.

The next example shows how the client can use the common data element `locations` (see Section 2.2) to request the issuance of an access token restricted to a certain RS. In our running example, the client may ask for all permissions of the approved grant of type `payment_initiation` applicable to the RS residing at `https://example.com/payments` as follows:

```
[
   {
      "type": "payment_initiation",
      "locations": [
         "https://example.com/payments"
      ]
   }
]
```

*Figure 14: Example of "authorization_details" Requesting an Audience-Restricted Access Token*

## 7. Token Response

In addition to the token response parameters as defined in [RFC6749], the AS **MUST** also return the `authorization_details` as granted by the resource owner and assigned to the respective access token.

The authorization details assigned to the access token issued in a token response are determined by the `authorization_details` parameter of the corresponding token request. If the client does not specify the `authorization_details` token request parameters, the AS determines the resulting `authorization_details` at its discretion.

The AS **MAY** omit values in the `authorization_details` to the client.

For our running example, it would look like this:

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

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

*Figure 15: Example Token Response*

### 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 some use cases where the AS enriches the data in an authorization details object. Whether enrichment is allowed and specifics of how it works are necessarily part of the definition of the respective authorization details type.

As one example, a client may ask for access to account information but leave the decision about the specific accounts it will be able to access to the user. During the course of the authorization process, the user would select the subset of their accounts that they want to allow the client to access. As one design option to convey the selected accounts, the AS could add this information to the respective authorization details object.

In that example, the requested `authorization_details` parameter might look like the following. In this example, the empty arrays serve as placeholders for where data will be added during enrichment by the AS. This example is illustrative only and is not intended to suggest a preference for designing the specifics of any authorization details type this way.

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

*Figure 16: Example of Requested "authorization_details"*

The AS 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-store

{
   "access_token":"2YotnFZFEjr1zCsicMWpAA",
   "token_type":"example",
   "expires_in":3600,
   "refresh_token":"tGzv3JokF0XG5Qx2TlKWIA",
   "authorization_details":[
      {
         "type":"account_information",
         "access":{
            "accounts":[
               {
                  "iban":"DE2310010010123456789"
               },
               {
                  "maskedPan":"123456xxxxxx1234"
               }
            ],
            "balances":[
               {
                  "iban":"DE2310010010123456789"
               }
            ],
            "transactions":[
               {
                  "iban":"DE2310010010123456789"
               },
               {
                  "maskedPan":"123456xxxxxx1234"
               }
            ]
         },
         "recurringIndicator":true
      }
   ]
}
```

*Figure 17: Example of Enriched "authorization_details"*

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" ]
   }
]}
```

*Figure 18: Example of Requested "authorization_details"*

When the user interacts with the AS, they select which of the medical records they are responsible for giving 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/" ]
     }
  ]
}
```

*Figure 19: Example of Enriched "authorization_details"*

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

## 8. Token Error Response

The Token Error Response **MUST** conform to the rules given in Section 5.

# 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` field to access tokens in JSON Web Token (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 a top-level claim.

The AS will typically also add further claims to the JWT that the RS requires 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 initiation 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://scheme.example.com/payment_initiation",
         "actions": [
            "initiate",
            "status",
            "cancel"
         ],
         "locations": [
            "https://example.com/payments"
         ],
         "instructedAmount": {
            "currency": "EUR",
            "amount": "123.50"
         },
         "creditorName": "Merchant A",
         "creditorAccount": {
            "iban": "DE02100100109307118603"
         },
         "remittanceInformationUnstructured": "Ref Number Merchant"
      }
   ],
   "debtorAccount": {
      "iban": "DE40100100103307118608",
      "user_role": "owner"
   }
}
```

*Figure 20: Example of "authorization_details" in JWT-Based Access Token*

In this case, the AS added the following example claims to the JWT-based access token:

`sub`: indicates the user for which the client is asking for payment initiation.

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

`debtorAccount`: API-specific field containing the debtor account. In the example, this account was not passed in the `authorization_details` but was selected by the user during the authorization process. The field `user_role` conveys the role the user has with respect to this particular account. In this case, they are the owner. This data is used for access control at the payment API (the RS).

## 9.2. Token Introspection

Token introspection [RFC7662] provides a means for an RS to query the AS to determine information about an access token. If the AS includes authorization detail information for the token in its response, the information **MUST** be conveyed with `authorization_details` as a top-level member of the introspection response JSON object. The `authorization_details` member **MUST** contain the same structure defined in Section 2, potentially filtered and extended for the RS making the introspection request.

Here is an example introspection response for the payment initiation example:

```
{
   "active": true,
   "sub": "24400320",
   "aud": "s6BhdRkqt3",
   "exp": 1311281970,
   "acr": "psd2_sca",
   "txn": "8b4729cc-32e4-4370-8cf0-5796154d1296",
   "authorization_details": [
      {
         "type": "https://scheme.example.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"
   }
}
```

*Figure 21: Example of "authorization_details" in Introspection Response*

# 10. Metadata

To advertise its support for this feature, the supported list of authorization details types is included in the AS metadata response [RFC8414] using the metadata parameter `authorization_details_types_supported`, which is a JSON array.

This is illustrated by the following example:

```
{
   ...
   "authorization_details_types_supported":[
      "payment_initiation",
      "account_information"
   ]
}
```

*Figure 22: Example of Server Metadata about the Supported Authorization Details*

Clients **MAY** indicate the authorization details types they will use when requesting authorization with the client registration metadata parameter `authorization_details_types`, which is a JSON array.

This is illustrated by the following example:

```
{
   ...
   "authorization_details_types":[
      "payment_initiation"
   ]
}
```

*Figure 23: Example of Server Metadata about Authorization Details*

The registration of authorization details types with the AS is outside the scope of this specification.

# 11. Implementation Considerations

## 11.1. Using Authorization Details in a Certain Deployment

Using authorization details in a certain deployment will require the following 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 prompt.

- 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 RSs process the authorization details or token data derived from authorization details.
- If needed, entitle clients to use certain authorization details types.

## 11.2. Minimal Implementation Support

General AS implementations supporting this specification should provide the following basic functions:

- Support advertisement of supported authorization details types in OAuth server metadata
- Accept the `authorization_details` parameter in authorization requests in conformance 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 RSs (similar to scope values). This should work with any grant type, especially `authorization_code` and `refresh_token`.

Processing and presentation of authorization details will vary significantly among different authorization details types. Implementations should therefore support customization of the respective behavior. In particular, implementations should allow deployments to:

- determine presentation of the authorization details;
- modify requested authorization details in the user consent process, e.g., adding fields; and
- merge requested and preexisting authorization details.

One approach to supporting such customization 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 RS by way of structured access tokens or token introspection responses.

## 11.3. Use of Machine-Readable Type Schemas

Implementations 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 identifier 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 would point to a machine-readable schema format or that any party in the system (such as the client, AS, or RS) would dereference or process the contents of the `type` field in any specific way.

### 11.4. Large Requests

Authorization request URIs containing `authorization_details` in a request parameter or a request object can become very long. Therefore, implementers should consider using the `request_uri` parameter as defined in [RFC9101] in combination with the pushed request object mechanism as defined in [RFC9126] 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 an 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
```

*Figure 24: Example of Large Request including "authorization_details"*

## 12. Security Considerations

The `authorization_details` parameter is sent through the user agent in case of an OAuth authorization request, which makes them vulnerable to modifications by the user. If the integrity of the `authorization_details` is a concern, clients **MUST** protect `authorization_details` against tampering and swapping. This can be achieved by signing the request using signed request objects as defined in [RFC9101] or using the `request_uri` authorization request parameter as defined in [RFC9101] in conjunction with [RFC9126] to pass the URI of the request object to the AS.

All string comparisons in an `authorization_details` parameter are to be done as defined by [RFC8259]. No additional transformation or normalization is to be done in evaluating equivalence of string values.

The common data field `locations` allows a client to specify where it intends to use a certain authorization, i.e., it is possible to unambiguously assign permissions to RSs. In situations with multiple RSs, this prevents unintended client authorizations (e.g., a `read` scope value potentially applicable for an email as well as a cloud service) through audience restriction.

The AS **MUST** properly sanitize and handle the data passed in the `authorization_details` in order to prevent injection attacks.

The Security Considerations of [RFC6749], [RFC7662], and [RFC8414] also apply.

# 13. Privacy Considerations

It is especially important for implementers to 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 [RFC9101] or transmission of `authorization_details` via end-to-end encrypted connections between client and AS by utilizing [RFC9126] and the `request_uri` authorization request parameter as defined in [RFC9101]. The latter does not require application-level encryption, but it requires another message exchange between the client and the AS.

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

The AS needs to take into consideration the privacy implications when sharing `authorization_details` with the client or RSs. The AS should share this data with those parties on a "need to know" basis as determined by local policy.

# 14. IANA Considerations

## 14.1. OAuth Parameters Registration

The following parameter has been registered in the "OAuth Parameters" registry [IANA.OAuth.Parameters] established by [RFC6749].

Name: `authorization_details`

Parameter Usage Location: authorization request, token request, token response

Change Controller: IETF

Reference: RFC 9396

## 14.2. JSON Web Token Claims Registration

The following value has been registered in the IANA "JSON Web Token Claims" registry established by [RFC7519].

Claim Name: `authorization_details`

Claim Description: The claim `authorization_details` contains a JSON array of JSON objects representing the rights of the access token. Each JSON object contains the data to specify the authorization requirements for a certain type of resource.

Change Controller: IETF

Reference: Section 9.1 of RFC 9396

## 14.3. OAuth Token Introspection Response Registration

The following value has been registered in the IANA "OAuth Token Introspection Response" registry established by [RFC7662].

Name: `authorization_details`

Description: The member `authorization_details` contains a JSON array of JSON objects representing the rights of the access token. Each JSON object contains the data to specify the authorization requirements for a certain type of resource.

Change Controller: IETF

Reference: Section 9.2 of RFC 9396

## 14.4. OAuth Authorization Server Metadata Registration

The following values have been registered 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: IETF

Reference: Section 10 of RFC 9396

## 14.5. OAuth Dynamic Client Registration Metadata Registration

The following value has been registered in the IANA "OAuth Dynamic Client Registration Metadata" registry of [IANA.OAuth.Parameters] established by [RFC7591].

Client Metadata Name: `authorization_details_types`

Client Metadata Description: Indicates what authorization details types the client uses.

Change Controller: IETF

Reference: Section 10 of RFC 9396

### 14.6. OAuth Extensions Error Registration

The following value has been registered in the IANA "OAuth Extensions Error Registry" of [IANA.OAuth.Parameters] established by [RFC6749].

Name: `invalid_authorization_details`

Usage Location: token endpoint, authorization endpoint

Protocol Extension: OAuth 2.0 Rich Authorization Requests

Change Controller: IETF

Reference: Section 5 of RFC 9396

## 15. References

### 15.1. Normative References

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

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

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

**[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>.

**[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>.

**[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>.

## 15.2. Informative References

**[CSC]** Cloud Signature Consortium, "Architectures and protocols for remote signature applications", Version 1.0.4.0, June 2019, <https://cloudsignatureconsortium.org/wp-content/uploads/2020/01/CSC_API_V1_1.0.4.0.pdf>.

**[ETSI]** ETSI, "Electronic Signatures and Infrastructures (ESI); Protocols for remote digital signature creation", V1.1.1, ETSI TS 119 432, March 2019, <https://www.etsi.org/deliver/etsi_ts/119400_119499/119432/01.01.01_60/ts_119432v010101p.pdf>.

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

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

**[OID-CIBA]** Fernandez, G., Walter, F., Nennker, A., Tonge, D., and B. Campbell, "OpenID Connect Client-Initiated Backchannel Authentication Flow - Core 1.0", 1 September 2021, <https://openid.net/specs/openid-client-initiated-backchannel-authentication-core-1_0.html>.

**[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, <https://openid.net/specs/openid-connect-core-1_0.html>.

**[RFC0020]** Cerf, V., "ASCII format for network interchange", STD 80, RFC 20, DOI 10.17487/RFC0020, October 1969, <https://www.rfc-editor.org/info/rfc20>.

**[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>.

**[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>.

**[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>.

**[RFC9101]** Sakimura, N., Bradley, J., and M. Jones, "The OAuth 2.0 Authorization Framework: JWT-Secured Authorization Request (JAR)", RFC 9101, DOI 10.17487/RFC9101, August 2021, <https://www.rfc-editor.org/info/rfc9101>.

**[RFC9126]** Lodderstedt, T., Campbell, B., Sakimura, N., Tonge, D., and F. Skokan, "OAuth 2.0 Pushed Authorization Requests", RFC 9126, DOI 10.17487/RFC9126, September 2021, <https://www.rfc-editor.org/info/rfc9126>.

**[Transaction-Auth]** 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>.

# Appendix A. Additional Examples

## A.1. OpenID Connect

OpenID Connect [OIDC] specifies the JSON-based `claims` request parameter that can be used to specify the claims a client (acting as an OpenID Connect Relying Party) wishes to receive in a fine-grained and privacy-preserving way as well as assign those claims to 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 details type for OpenID Connect. This section gives an example of what such an authorization details type could look like, but defining this authorization details type is outside the scope of this specification.

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 fields are based on the definitions given in [OIDC]:

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

`max_age`: Maximum Authentication Age

`acr_values`: requested Authentication Context Class Reference (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"
      ]
   }
]
```

*Figure 25: Example of OpenID Connect Request Utilizing "authorization_details"*

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

A more sophisticated example is shown in Figure 26.

```
[
   {
      "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.com/claims/groups": null
         },
         "id_token": {
            "auth_time": {
               "essential": true
            }
         }
      }
   }
]
```

*Figure 26: Advanced Example of OpenID Connect Request Utilizing "authorization_details"*

## A.2. Remote Electronic Signing

The following example is based on the concept laid out for remote electronic signing in ETSI TS 119 432 [ETSI] and the Cloud Signature Consortium (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"
   }
]
```

*Figure 27: Example of Electronic Signing*

The top-level fields 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` fields). Additionally, the corresponding `label` field identifies the respective document to the user, e.g., to be used in user consent.

`hashAlgorithm`: algorithm that was used to calculate the hash values

The AS is supposed to ask the user for consent for the creation of signatures for the documents listed in the structure. The client uses the access token issued as a result of the process to call the document signature API 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 creditworthiness.

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

*Figure 28: Example of Tax Data Access*

The top-level fields have the following meaning:

`periods`: the periods the client wants to access

`duration_of_access`: how long the clients intend to access the data in days

`tax_payer_id`: identifier of the taxpayer (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 their computer using a local Electronic Health Records (EHR) system. They want to look at the electronic patient records of a certain patient, and they also want to fetch the patient's 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]"
                }
            }
        }
    }
}
```

*Figure 29: eHealth Example*

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.example.org/fhir/org-type",
                        "code": "dept",
                        "display": "Hospital Department"
                     }
                  ]
               },
               "name": "Akuttmottak"
            },
            "profession": {
               "coding": [
                  {
                     "system": "http://snomed.example.org/sct",
                     "code": "36682004",
                     "display": "Physical therapist"
                  }
               ]
            }
         }
      }
```

```
    }
 ]
```

*Figure 30: Advanced eHealth Example*

Description of the fields:

`patient_identifier`: the identifier of the patient composed of a system identifier in OID format (namespace) and the actual 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.

## Acknowledgements

We 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 specification.

We would also like to thank Vladimir Dzhuvinov, Takahiko Kawasaki, Daniel Fett, Dave Tonge, Travis Spencer, Joergen Binningsboe, Aamund Bremer, Steinar Noem, Francis Pouatcha, Jacob Ideskog, Hannes Tschofenig, and Aaron Parecki for their valuable feedback to this specification.

## 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