Logius Standard - Draft

NLGov Profile for OpenID AuthZEN Authorization API

Logius Standard
Draft October 10, 2025

This version:
https://logius-standaarden.github.io/authzen-nlgov/

Latest published version:
https://logius-standaarden.github.io/logboek-dataverwerkingen/

Latest editor's draft:
https://logius-standaarden.github.io/authzen-nlgov/

Editors:
Stas Mironov (Logius)
Alexander Green (Logius)

Author:
Michiel Trimpe (VNG Realisatie)

Participate:
GitHub Logius-standaarden/authzen-nlgov
File an issue
Commit history
Pull requests

This document is also available in these non-normative format: PDF

Status of This Document

This is a draft that could be altered, removed or replaced by other documents. It is not a recommendation approved by TO.

Table of Contents

§ Conformance

As well as sections marked as non-normative, all authoring guidelines, diagrams, examples, and notes in this specification are non-normative. Everything else in this specification is normative.

The key words *MAY*, *MUST*, *MUST NOT*, *OPTIONAL*, *RECOMMENDED*, *REQUIRED*, *SHOULD*, and *SHOULD NOT* 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.

Abstract

The Authorization API enables Policy Decision Points (PDPs) and Policy Enforcement Points (PEPs) to communicate authorization requests and decisions to each other without requiring knowledge of each other's inner workings. The Authorization API is served by the PDP and is called by the PEP. The Authorization API includes evaluation endpoints, which provide specific access decisions, and search endpoints for discovering permissible subjects, resources, or actions.

This document is an adaptation of the 'OpenID AuthZEN Authorization API 1.0 – draft 04' (hereinafter: AuthZEN) of the OpenID Foundation. This does not indicate an endorsement by the OpenID Foundation. In as far as AuthZEN is incorporated in this document, the OpenID Copyright License applies.

§ Dutch government profile for OpenID AuthZEN Authorization API

This profile is based upon the AuthZEN Authorization API as published by the OpenID Foundation. It should be considered a fork of this standard geared more towards the Netherlands and operating within the context of the European Union.

Starting with chapter Introduction we follow the structure of the AuthZEN Authorization API. Content introduced by the NLGov profile is marked in green. Content from the AuthZEN Authorization API that is not used in the NLGov profile is marked ~~red~~.

§ 1. Introduction

Computational services often implement access control within their components by separating Policy Decision Points (PDPs) from Policy Enforcement Points (PEPs). PDPs and PEPs are defined in XACML ([*XACML20*]) and NIST's ABAC SP 800-162 ([*NIST.SP.800-162*]). Communication between PDPs and PEPs follows similar patterns across different software and services that require or provide authorization information. The Authorization API described in this document enables different providers to offer PDP and PEP capabilities without having to bind themselves to one particular implementation of a PDP or PEP.

2. Model

By convention, we refer to a service that implements this API as a Policy Decision Point, or PDP. The policy language, architecture, and state management aspects of a PDP are beyond the scope of this specification.

By convention, we refer to a client of the Authorization API as a Policy Enforcement Point, or PEP. Clients may consume the Authorization API for use cases that go beyond enforcement of authorization decisions; for example, the Resource Search API (8.5 Resource Search API) allows a caller to discover the resources on which a subject can perform an action. For consistency, we use the term PEP to describe a client of the API, regardless of the use case.

The Authorization API is defined in a transport-agnostic manner. A normative HTTPS binding is described in Transport (10. Transport). Other bindings, such as gRPC, may be defined in other profiles of this specification.

Authentication for the Authorization API itself is out of scope for this document, since authentication for APIs is well-documented elsewhere. Support for OAuth 2.0 ([*RFC6749*]) is *RECOMMENDED*.

3. Features

The core feature of the Authorization API is the Access Evaluation API (6. Access Evaluation API), which enables a PEP to determine whether a specific request can be permitted to access a specific resource. The following are non-normative examples:

- Can Alice view document #123?
- Can Alice view document #123 at 16:30 on Tuesday, June 11, 2024?
- Can a manager print?

The Access Evaluations API (7. Access Evaluations API) enables execution of multiple evaluations in a single request. The following are non-normative examples:

- Can Alice view documents 123, 234 and 345 on Tuesday, June 11, 2024?
- Can document 123 be viewed by Alice and Bob?

The Search APIs (8. Search APIs) provide lists of resources, subjects or actions that would be allowed access. The following are non-normative examples:

- Which documents can Alice view?
- Who can view document 123?
- What actions can Alice perform on document 123 on Tuesday, June 11, 2024?

4. API Version

This document describes the API version 1.0. Any updates to this API through subsequent revisions of this document or other documents *MAY* augment this API, but *MUST NOT* modify the API described here. Augmentation *MAY* include additional API methods or additional parameters to existing API methods, additional authorization mechanisms, or additional optional headers in HTTPS transport bindings. Endpoints for version 1.0 *SHOULD* include `v1` in the endpoint identifier (e.g. `https://pdp.example.com/access/v1/`).

5. Information Model

The information model for requests and responses include the following entities: Subject, Action, Resource, Context, and Decision. These are all defined below.

Specific implementations of the generic AuthZEN information model *SHOULD* be documented in a meta-information model. This enables an unambigous interpretation of the meaning of requests. It is *RECOMMENDED* to document to the meta-information model using [*MIM*].

It is *RECOMMENDED* to use [*JSON-LD11*] to enable automatic integration into existing semantic models, as described in 5.4.1.4 Meta-information Model.

5.1 Subject

A Subject is the user or machine principal about whom the Authorization API is being invoked. The Subject may be requesting access at the time the Authorization API is invoked.

A Subject is an object that contains two *REQUIRED* keys, `type` and `id`, which have a string value, and an *OPTIONAL* key, `properties`, with a value of an object.

`type`: : *REQUIRED*. A string value that specifies the type of the Subject. It is *RECOMMENDED* to define the type as a Linked Data URI.

`id`: : *REQUIRED*. A string value containing the unique identifier of the Subject, scoped to the `type`.

`properties`: : *OPTIONAL*. An object which can be used to express additional attributes of a Subject.

§ 5.1.1 Subject Properties

Many authorization systems are stateless, and expect the PEP to pass in all relevant attributes used in the evaluation of the authorization policy. To satisfy this requirement, Subjects *MAY* include additional attributes as key-value pairs, under the `properties` object. A property can contain both simple values, such as strings, numbers, booleans and nulls, and complex values, such as arrays and objects.

Examples of subject attributes can include, but are not limited to:

- department,
- group memberships,
- device identifier,
- IP address.

§ 5.1.2 Examples (non-normative)

The following is a non-normative example of a minimal Subject:

EXAMPLE 1: Example Subject

```
{
  "type": "user",
  "id": "alice@example.com"
}
```

The following is a non-normative example of a Subject which adds a string-valued `department` property:

EXAMPLE 2: Example Subject with Additional Property

```
{
  "type": "user",
  "id": "alice@example.com",
  "properties": {
    "department": "Sales"
  }
}
```

The following is a non-normative example of a subject which adds IP address and device identifier properties:

EXAMPLE 3: Example Subject with IP Address and Device ID

```
{
  "type": "user",
  "id": "alice@example.com",
  "properties": {
    "ip_address": "172.217.22.14",
    "device_id": "8:65:ee:17:7e:0b"
  }
}
```

5.2 Resource

A Resource is the target of an access request. It is an object that is constructed similar to a Subject entity. It has the following keys:

`type`: : *REQUIRED*. A string value that specifies the type of the Resource. It is *RECOMMENDED* to define the type as a Linked Data URI.

`id`: : *REQUIRED*. A string value containing the unique identifier of the Resource, scoped to the `type`.

`properties`: : *OPTIONAL*. An object which can be used to express additional attributes of a Resource.

5.2.1 Resource Properties

Similarly to the Subject properties, the PEP can also provide attributes for the Resource in the properties field.

Such attributes can include, but are not limited to, attributes of the resource used in access evaluations or metadata about the resource.

5.2.2 Examples (non-normative)

The following is a non-normative example of a Resource with a `type` and a simple `id`:

EXAMPLE 4: Example Resource

```
{
  "type": "book",
  "id": "123"
}
```

The following is a non-normative example of a Resource containing a `library_record` property, that is itself an object:

EXAMPLE 5: Example Resource with Additional Property

```
{
  "type": "book",
  "id": "123",
  "properties": {
    "library_record":{
      "title": "AuthZEN in Action",
      "isbn": "978-0593383322"
    }
  }
}
```

5.3 Action

An Action is the type of access that the requester intends to perform.

Action is an object that contains a *REQUIRED* `name` key with a string value, and an *OPTIONAL* `properties` key with an object value.

`name`: : *REQUIRED*. A string value containing the name of the Action.

`properties`: : *OPTIONAL*. An object which can be used to express additional attributes of an Action.

5.3.1 Action Properties

Similarly to the Subject and Resource properties, the PEP can also provide attributes for the Action in the properties field.

Such attributes can include, but are not limited to, parameters of the action that is being requested.

To increase interoperability, a few common properties are specified below:

§ *5.3.1.1 Processing Activity identifier*

Under Dutch and EU legislation, processing of personal data should be described in a Record of Processing Activities. In certain cases, e.g. when a single system processes data for multiple different processing activities, a relation to the processing activity *MAY* be included.

When included, the reference to the processing activity *SHOULD* be included using the following key:

`processing_activity_id`: : *REQUIRED*. A string value containing the URI of the processing activity within a Processing Activity registry.

> NOTE
>
> The processing activity identifier should only be used within the context of an organization and *SHOULD NOT* cross organizational boundaries.

§ *5.3.1.2 Algorithm identifier*

When data is processed as part of an algorithm in a public registry, such as "Het Algoritmeregister", a reference to the relevant algorithm *MAY* be included.

When included, the reference to the algorithm *SHOULD* be included using the following key:

`algorithm_id`: : *REQUIRED*. A string value containing the URI of the algorithm in an algorithm registry.

> NOTE
>
> The algorithm identifier should only be used within the context of an organization and *SHOULD NOT* cross organizational boundaries.

§ 5.3.2 Examples (non-normative)

The following is a non-normative example of an action:

EXAMPLE 6: Example Action

```
{
  "name": "can_read"
}
```

The following is a non-normative example of an action with additional properties:

EXAMPLE 7: Example Action with properties for extending a book loan.

```
{
  "name": "extend-loan",
  "properties": {
    "period": "2W"
  }
}
```

§ 5.4 Context

The Context represents the environment of the access evaluation request.

Context is an object which can be used to express attributes of the environment.

Examples of context attributes can include, but are not limited to:

- The time of day,
- Location from which the request was received,
- Capabilities of the PEP,
- JSON Schema or JSON-LD definitions for the request.

§ 5.4.1 Context Properties

Context *MAY* include zero or more additional attributes as key-value pairs.

To increase interoperability, a few common properties are specified below:

§ *5.4.1.1 Time*

The logical time at which the action was considered to be initiated, identified by the `time` field, whose value is a textual representation of the time as defined in [*RFC3339*].

This timestamp *SHOULD* be used when a PDP evaluates the access request uses information from data sources that support temporal queries. See for example the *API Design Rules* and its temporal extension.

§ *5.4.1.2 W3C Trace Context*

To enable tracing of requests, request identifiers *SHOULD* be included in the evaluation request. Request identifiers *SHOULD* be included in the Context object. They *SHOULD* be in the form of `tracestate` and `traceparent` values as defined by [*trace-context-1*].

When included, the W3C Trace Context *SHOULD* be included in the Context object using the following keys:

`traceparent`: : *REQUIRED*. An string value containing a value as defined in Section 3.2.2 of [*trace-context-1*]

`tracestate`: : *REQUIRED*. An string value containing a value as defined in Section 3.3.1.1 of [*trace-context-1*]

§ *5.4.1.3 Verifiable claims*

As described in 11.4 Trust, it is recommended to consider values in the information model as trusted and valid. For purposes of defense-in-depth and traceability, verifiable claims for values in the

information model *MAY* be provided. The verifiable claims *MAY* use standards such as, but not limited to, SAML ([*SAML2-CORE*]), Oauth ([*RFC6749*]), and Verifiable Credentials ([*vc-data-model-2.0*]).

§ *5.4.1.4 Meta-information Model*

It is *RECOMMENDED* to make the information model self-describing by including a URL to the meta-information model 5. Information Model in the context.

When included, the meta-information model *SHOULD* be included in Context object as the following key

`mim`: : *REQUIRED*. A string value containing a URL that links to the meta-information model for the request.

§ *5.4.1.5 Linked Data context*

When a transport (10. Transport) does not use a Linked Data format as its serialization, the Context *SHOULD* include a URL to a resource, called the "Linked Data context" that allows the information model to be converted to a Linked Data representation.

> NOTE
>
> The Linked Data context is *not* the same as the Context object. The Context object describes the context in which an evaluation request takes place. The Linked Data context describes how to convert the *entire* request, containing a Subject, Action, Resource and Context object, to a Linked Data representation.

When included, the Linked Data context *SHOULD* be included in Context object as the following key:

`ld-context`: : *REQUIRED*. An object that provides context for mapping the serialized information model to Linked Data, or a string value containing a URL from which the mapping can be retrieved.

When serializing the information model to JSON it is *RECOMMENDED* to use [*JSON-LD11*] to provide the Linked Data context. In that case, the value of the `ld-context` key should be considered as the value of the `@context` key at top-level.

§ 5.4.2 Examples (non-normative)

The following is a non-normative example of a Context:

EXAMPLE 8: Example Context

```
{
  "time": "1985-10-26T01:22-07:00"
}
```

5.5 Decision

A Decision is the result of the evaluation of an access request. It provides the information required for the PEP to enforce the decision.

Decision is an object that contains a *REQUIRED* `decision` key with a `boolean` value, and an *OPTIONAL* `context` key with an object value.

`decision`: : *REQUIRED*. A boolean value that specifies whether the Decision is to allow or deny the operation.

`context`: : *OPTIONAL*. An object which can convey additional information that can be used by the PEP as part of the decision enforcement process.

In this specification, assuming the evaluation was successful, there are only two possible values for the `decision`:

- `true`: The access request is permitted to go forward. If the PEP does not understand information in the `context` response object, the PEP *MAY* choose to reject the decision.
- `false`: The access request is denied and *MUST NOT* be permitted to go forward.

The following is a non-normative example of a minimal Decision:

EXAMPLE 9: Example Decision

```
{
  "decision": true
}
```

5.5.1 Decision Context

In addition to a `decision`, a response *MAY* contain a `context` field which contains an object. This context can convey additional information that can be used by the PEP as part of the decision

enforcement process.

Examples include, but are not limited to:

- Reason(s) a decision was made,
- "Advices" and/or "Obligations" tied to the access decision,
- Hints for rendering UI state,
- Instructions for step-up authentication,
- Environmental information,
- etc.

§ 5.5.2 Examples (non-normative)

The following are all non-normative examples of possible and valid contexts, provided to illustrate possible usages. The actual semantics and format of the `context` object are an implementation concern and outside the scope of this specification. For example, implementations *MAY* use keys that correspond to concepts from other standards, such as HTTP status codes, to convey common reasons in an interoperable manner.

§ *5.5.2.1 Non-normative Example 1: conveying decision Reasons*

The PDP may provide reasons to explain a decision. In the non-normative example below, an implementation might convey different reasons to administrators and end-users, using keys that could correspond to HTTP status codes:

EXAMPLE 10: Non-normative Example Response with reason Context

```
{
  "decision": false,
  "context": {
    "reason_admin": {
      "403": "Request failed policy C076E82F"
    },
    "reason_user": {
      "403": "Insufficient privileges. Contact your administrator"
    }
  }
}
```

§ *5.5.2.2 Non-normative Example 2: conveying metadata and environmental elements*

In the following non-normative example, the PDP justifies its decision by including environmental conditions that did not meet its policies. Metadata pertaining to the decision response times is also provided:

EXAMPLE 11: Non-normative Example Response with Environment and Metadata Context

```
{
  "decision": false,
  "context": {
    "metadata": {
      "response-time": 60,
      "response-time-unit": "ms"
    },
    "environment": {
      "ip": "10.10.0.1",
      "datetime": "2025-06-27T18:03:07Z",
      "os": "ubuntu24.04.2LTS-AMDx64"
    }
  }
}
```

§ *5.5.2.3 Non-normative Example 3: requesting step-up authentication*

In the following non-normative example, the PDP requests a step-up authentication of the requesting subject, by signalling the required `acr` and `amr` access token claim values it expects to see in order to approve the request:

EXAMPLE 12: Non-normative Example Response with a step-up request Context

```
{
  "decision": false,
  "context": {
    "acr_values": "urn:com:example:loa:3",
    "amr_values": "mfa hwk"
  }
}
```

6. Access Evaluation API

The Access Evaluation API defines the message exchange pattern between a PEP and a PDP for executing a single access evaluation.

6.1 The Access Evaluation API Request

The Access Evaluation request is an object consisting of four entities previously defined in the Information Model (5. Information Model):

`subject`: : *REQUIRED*. The subject (or principal) of type Subject

`action`: : *REQUIRED*. The action (or verb) of type Action.

`resource`: : *REQUIRED*. The resource of type Resource.

`context`: : *OPTIONAL*. The context (or environment) of type Context.

6.1.1 Example (non-normative)

EXAMPLE 13: Example Request

```
{
  "subject": {
    "type": "user",
    "id": "alice@example.com"
  },
  "resource": {
    "type": "account",
    "id": "123"
  },
  "action": {
    "name": "can_read",
    "properties": {
      "method": "GET"
    }
  },
  "context": {
    "time": "1985-10-26T01:22-07:00"
  }
}
```

6.2 The Access Evaluation API Response

The response of the Access Evaluation API consists of the Decision entity as defined in the Information Model (5. Information Model).

7. Access Evaluations API

The Access Evaluations API defines the message exchange pattern between a PEP and a PDP for evaluating multiple access evaluations within the scope of a single message exchange (also known as "boxcarring" requests).

§ 7.1 The Access Evaluations API Request

The Access Evaluation API Request builds on the information model presented in 5. Information Model and the object defined in the Access Evaluation Request (6.1 The Access Evaluation API Request).

To send multiple access evaluation requests in a single message, the PEP *MAY* add an `evaluations` key to the request. The `evaluations` key is an array which contains a list of objects, each typed as the object as defined in the Access Evaluation Request (6.1 The Access Evaluation API Request), and specifying a discrete request.

If an `evaluations` array is NOT present or is empty, the Access Evaluations Request behaves in a backwards-compatible manner with the (single) Access Evaluation API Request (6.1 The Access Evaluation API Request).

If an `evaluations` array IS present and contains one or more objects, these form distinct requests that the PDP will evaluate. These requests are independent from each other, and may be executed sequentially or in parallel, left to the discretion of each implementation.

The top-level `subject`, `action`, `resource`, and `context` keys provide default values for their respective fields in the `evaluations` array. The top-level `subject`, `action` and `resource` keys *MAY* be omitted if the `evaluations` array is present, contains one or more objects, and every object in the `evaluations` array contains the respective top-level key. This behavior is described in 7.1.1 Default values.

The following is a non-normative example for specifying three requests, with no default values:

EXAMPLE 14

```
{
  "evaluations": [
    {
      "subject": {
        "type": "user",
        "id": "alice@example.com"
      },
      "action": {
        "name": "can_read"
      },
      "resource": {
        "type": "document",
        "id": "boxcarring.md"
      },
      "context": {
        "time": "2024-05-31T15:22-07:00"
      }
    },
    {
      "subject": {
        "type": "user",
        "id": "alice@example.com"
      },
      "action": {
        "name": "can_read"
      },
      "resource": {
        "type": "document",
        "id": "subject-search.md"
      },
      "context": {
        "time": "2024-05-31T15:22-07:00"
      }
    },
    {
      "subject": {
        "type": "user",
        "id": "alice@example.com"
      },
      "action": {
        "name": "can_read"
      },
      "resource": {
        "type": "document",
        "id": "resource-search.md"
```

```
      },
      "context": {
        "time": "2024-05-31T15:22-07:00"
      }
    }
  ]
}
```

7.1.1 Default values

While the example above provides the most flexibility in specifying distinct values in each request for every evaluation, it is common for boxcarred requests to share one or more values of the evaluation request. For example, evaluations *MAY* all refer to a single subject, and/or have the same contextual (environmental) attributes.

Default values offer a more compact syntax that avoids unnecessary duplication of request data.

The top-level `subject`, `action`, `resource`, and `context` keys provide default values for each object in the evaluations array. Any of these keys specified within an individual evaluation object overrides the corresponding top-level default. Because `subject`, `action`, and `resource` are required for a valid evaluation, any of these keys omitted from an evaluation object *MUST* be provided as a top-level key.

The following is a non-normative example for specifying three requests that refer to a single subject and context:

EXAMPLE 15

```
{
  "subject": {
    "type": "user",
    "id": "alice@example.com"
  },
  "context": {
    "time": "2024-05-31T15:22-07:00"
  },
  "evaluations": [
    {
      "action": {
        "name": "can_read"
      },
      "resource": {
        "type": "document",
        "id": "boxcarring.md"
      }
    },
    {
      "action": {
        "name": "can_read"
      },
      "resource": {
        "type": "document",
        "id": "subject-search.md"
      }
    },
    {
      "action": {
        "name": "can_read"
      },
      "resource": {
        "type": "document",
        "id": "resource-search.md"
      }
    }
  ]
}
```

The following is a non-normative example for specifying three requests that refer to a single `subject` and `context`, with a default value for `action`, that is overridden by the third request:

EXAMPLE 16

```
{
  "subject": {
    "type": "user",
    "id": "alice@example.com"
  },
  "context": {
    "time": "2024-05-31T15:22-07:00"
  },
  "action": {
    "name": "can_read"
  },
  "evaluations": [
    {
      "resource": {
        "type": "document",
        "id": "boxcarring.md"
      }
    },
    {
      "resource": {
        "type": "document",
        "id": "subject-search.md"
      }
    },
    {
      "action": {
        "name": "can_edit"
      },
      "resource": {
        "type": "document",
        "id": "resource-search.md"
      }
    }
  ]
}
```

§ 7.1.2 Evaluations options

The `evaluations` request payload includes an *OPTIONAL* `options` key, with a value that is an object.

This provides a general-purpose mechanism for providing PEP-supplied metadata on how the request is to be executed.

One such option controls *evaluation semantics*, and is described in 7.1.2.1 Evaluations semantics.

A non-normative example of the `options` field is shown below, following an `evaluations` array provided for the sake of completeness:

EXAMPLE 17

```
{
  "evaluations": [{
    "resource": {
      "type": "doc",
      "id": "1"
    },
    "subject": {
      "type": "doc",
      "id": "2"
    }
  }],
  "options": {
    "evaluations_semantic": "execute_all",
    "another_option": "value"
  }
}
```

§ *7.1.2.1 Evaluations semantics*

By default, every request in the `evaluations` array is executed and a response returned in the same array order. This is the most common use-case for boxcarring multiple evaluation requests in a single payload.

This specification supports three evaluation semantics:

1. *Execute all of the requests (potentially in parallel), return all of the results.* Any failure can be denoted by `"decision": false` and *MAY* provide a reason code in the context.
2. *Deny on first denial (or failure).* This semantic could be desired if a PEP wants to issue a few requests in a particular order, with any denial (error, or `"decision": false`) "short-circuiting" the evaluations call and returning on the first denial. This essentially works like the && operator in programming languages.

3. *Permit on first permit.* This is the converse "short-circuiting" semantic, working like the `||` operator in programming languages.

To select the desired evaluation semantic, a PEP can pass in `options.evaluations_semantic` with exactly one of the following values:

- `execute_all`
- `deny_on_first_deny`
- `permit_on_first_permit`

`execute_all` is the default semantic, so an `evaluations` request without the `options.evaluations_semantic` flag will execute using this semantic.

§ 7.1.2.1.1 Example: Evaluate `read` action for three documents using all three semantics

Execute all requests:

EXAMPLE 18

```
{
  "subject": {
    "type": "user",
    "id": "alice@example.com"
  },
  "action": {
    "name": "read"
  },
  "options": {
    "evaluations_semantic": "execute_all"
  },
  "evaluations": [
    {
      "resource": {
        "type": "document",
        "id": "1"
      }
    },
    {
      "resource": {
        "type": "document",
        "id": "2"
      }
    },
    {
      "resource": {
        "type": "document",
        "id": "3"
      }
    }
  ]
}
```

Response:

EXAMPLE 19

```
{
  "evaluations": [
    {
      "decision": true
    },
    {
      "decision": false
    },
    {
      "decision": true
    }
  ]
}
```

Deny on first deny:

EXAMPLE 20

```
{
  "subject": {
    "type": "user",
    "id": "alice@example.com"
  },
  "action": {
    "name": "read"
  },
  "options": {
    "evaluations_semantic": "deny_on_first_deny"
  },
  "evaluations": [
    {
      "resource": {
        "type": "document",
        "id": "1"
      }
    },
    {
      "resource": {
        "type": "document",
        "id": "2"
      }
    },
    {
      "resource": {
        "type": "document",
        "id": "3"
      }
    }
  ]
}
```

Response:

EXAMPLE 21

```
{
  "evaluations": [
    {
      "decision": true
    },
    {
      "decision": false,
      "context": {
        "code": "200",
        "reason": "deny_on_first_deny"
      }
    }
  ]
}
```

Permit on first permit:

EXAMPLE 22

```
{
  "subject": {
    "type": "user",
    "id": "alice@example.com"
  },
  "action": {
    "name": "read"
  },
  "options": {
    "evaluations_semantic": "permit_on_first_permit"
  },
  "evaluations": [
    {
      "resource": {
        "type": "document",
        "id": "1"
      }
    },
    {
      "resource": {
        "type": "document",
        "id": "2"
      }
    },
    {
      "resource": {
        "type": "document",
        "id": "3"
      }
    }
  ]
}
```

Response:

EXAMPLE 23

```
{
  "evaluations": [
    {
      "decision": true
    }
  ]
}
```

§ 7.2 The Access Evaluations API Response

Like the request format, the Access Evaluations Response format for an Access Evaluations Request adds an `evaluations` array that lists the decisions in the same order they were provided in the `evaluations` array in the request. Each value of the evaluations array is typed as a Decision as defined in the Information Model (5. Information Model).

In case the `evaluations` array is present, it is *RECOMMENDED* that the `decision` key of the response be omitted. If present, it can be ignored by the PEP.

The following is a non-normative example of a Access Evaluations Response to an Access Evaluations Request containing three evaluation objects:

EXAMPLE 24

```
{
  "evaluations": [
    {
      "decision": true
    },
    {
      "decision": false,
      "context": {
        "reason": "resource not found"
      }
    },
    {
      "decision": false,
      "context": {
        "reason": "Subject is a viewer of the resource"
      }
    }
  ]
}
```

§ 7.2.1 Errors

There are two types of errors, and they are handled differently:

1. Transport-level errors, or errors that pertain to the entire payload.
2. Errors in individual evaluations.

The first type of error is handled at the transport level. For example, for the HTTP binding, the 4XX and 5XX codes indicate a general error that pertains to the entire payload, as described in Transport (10. Transport).

The second type of error is handled at the payload level. Decisions default to *closed* (i.e. `false`), but the `context` field can include errors that are specific to that request.

The following is a non-normative example of a response to an Access Evaluations Request containing three evaluation objects, two of them demonstrating how errors can be returned for two of the evaluation requests:

EXAMPLE 25

```
{
  "evaluations": [
    {
      "decision": true
    },
    {
      "decision": false,
      "context": {
        "error": {
          "status": 404,
          "message": "Resource not found"
        }
      }
    },
    {
      "decision": false,
      "context": {
        "reason": "Subject is a viewer of the resource"
      }
    }
  ]
}
```

8. Search APIs

The Search APIs enable a PEP to discover the set of subjects, resources, or actions that are permitted within a specific authorization context. Their purpose is to return a list of authorized entities, rather than verify a single access request.

To perform a search, the PEP provides the Subject, Resource, Action, and Context entities defined in the Information Model (5. Information Model), but omits the unique identifier of the entity being queried. The PDP then responds with the set of authorized entities for the queried entity type which would be authorized according to the provided criteria.

8.1 Semantics

A search is designed to return entities that would correspond to a permitted decision. Therefore, any result from a Search API, when subsequently used in an Access Evaluation API call, *SHOULD* result

in a `"decision": true` response. However, because the evaluation is implementation-specific and may depend on other variables (such as time), this outcome is not guaranteed.

In addition, it is *RECOMMENDED* that a search be performed transitively, traversing intermediate attributes and/or relationships. For example, if user U is a member of group G, and group G is designated as a viewer on a document D, then a search for all subjects of type user that can view document D will include user U.

§ 8.2 Pagination

Search APIs can return large result sets. To manage this, a PDP *MAY* support pagination, allowing a PEP to navigate and retrieve subsets of the total result set.

Pagination does not guarantee an atomic snapshot of the result set. Consequently, if items are added or removed while paginating, results *MAY* be repeated or omitted between pages.

Pagination is based on the use of opaque tokens. A PEP makes an initial request for data by sending a query that does not contain a token. If the PDP determines that the result set contains too many results to fit in a single response, the PDP returns a partial result set and a token that the PEP can use to retrieve the next page of results.

A paginated response *MUST* be clearly identified by the inclusion of a `page` object containing a non-empty, opaque `next_token`. This token is the signal to the PEP that more results are available.

To retrieve the next page, the PEP sends a subsequent request containing a `page` object with the `token` field set to the `next_token` value from the previous response. This process is repeated until the PDP returns a `page` object in which the value of the `next_token` field is an empty string, signaling the end of the result set.

When a request contains a token, all entities (e.g., `subject`, `resource`, `action`, `context`) and pagination parameters (e.g., `limit`) *MUST* be identical to the preceding request. PDPs *SHOULD* return an error when any entity or parameter has been changed.

PEPs that wish to sequentially iterate through the entire result set *SHOULD* use the core pagination mechanism described above, which is designed to work consistently across all PDPs that support the search APIs.

§ 8.2.1 Paginated Requests

A Search API Request *MAY* include a `page` object indicating which subset of the larger result set the PEP would like to receive.

The `page` object in a Search API Request consists of the following keys:

`token`: : *OPTIONAL*. An opaque string value from the `next_token` of a previous response.

`limit`: : *OPTIONAL*. A non-negative integer indicating the maximum number of results to return in the response.

`properties`: : *OPTIONAL*. An object containing additional implementation-specific pagination request attributes, such as, but not limited to, sorting and filtering.

Apart from the `token`, all values from the initial request *MUST* remain identical for subsequent pages. If a different value is provided mid-pagination the PDP *SHOULD* return an error.

Additional keys *MAY* be included in the `page` object. If they are, they *MUST* be defined in a specification referenced in the AuthZEN Policy Decision Point Capabilities Registry (12.3 AuthZEN Policy Decision Point Capabilities Registry). Furthermore, the PDP *MUST* declare support for the corresponding capability URN in its `supported_capabilities` metadata (9.1.2 Capabilities Parameters).

§ 8.2.2 Paginated Responses

Any Search API Response *MAY* include a `page` object, but if a response does not contain the entire result set, it *MUST* include this object.

The `page` object contains the following keys:

`next_token`: : *REQUIRED*. An opaque string value indicating the next page of results to return. If there are no more results after this page, its value *MUST* be an empty string.

`count`: : *OPTIONAL*. A non-negative integer indicating the number of results included in this response. When included at the start of a response, as described in the Search API Response (8.3 The Search API Response), this enables a PEP to display a progress indicator when processing large or slow responses.

`total`: : *OPTIONAL*. A non-negative integer indicating the total number of results matching the query criteria at the time of the request. This value is not guaranteed to equal the total number of items returned across all pages if the underlying data set changes during pagination.

`properties`: : *OPTIONAL*. An object containing additional pagination response attributes. Examples include, but are not limited to, estimated totals or the number of remaining results.

§ 8.2.3 Examples (non-normative)

The following is a non-normative example of a request-response cycle to retrieve a total of three results with a page size limit of two.

EXAMPLE 26: Example initial Search API Request

```
{
  "subject": {
    "type": "user",
    "id": "alice@example.com"
  },
  "action": {
    "name": "can_read"
  },
  "resource": {
    "type": "account"
  },
  "page": {
    "limit": 2
  }
}
```

EXAMPLE 27: Example initial Search API Response

```
{
  "page": {
    "next_token": "a3M9NDU2O3N6PTI=",
    "count": 2,
    "total": 3
  },
  "results": [
    {
      "type": "account",
      "id": "123"
    },
    {
      "type": "account",
      "id": "456"
    }
  ]
}
```

EXAMPLE 28: Example second Search API Request

```
{
  "subject": {
    "type": "user",
    "id": "alice@example.com"
  },
  "action": {
    "name": "can_read"
  },
  "resource": {
    "type": "account"
  },
  "page": {
    "token": "a3M9NDU2O3N6PTI="
  }
}
```

EXAMPLE 29: Example second Search API Response

```
{
  "page": {
    "next_token": "",
    "count": 1,
    "total": 3
  },
  "results": [
    {
      "type": "account",
      "id": "789"
    }
  ]
}
```

8.3 The Search API Response

The response to a Search API Request always follows the same structure. Each Search API Response is a JSON object with the following keys:

`page`: : *OPTIONAL*. An object providing pagination information, as defined in Paginated Responses (8.2.2 Paginated Responses). It is *RECOMMENDED* that the `page` object be the first key in the response, as this allows a PEP to use the `count` value to display a progress indicator when processing large or slow responses.

`context`: : *OPTIONAL*. An object that can convey additional information that can be used by the PEP, similar to its function in the Access Evaluation Response (see 6.2 The Access Evaluation API Response).

`results`: : *REQUIRED*. An array containing zero or more entities, as defined in the Information Model (5. Information Model). It *MUST* contain only entities of the type being searched for (e.g., Subjects, Resources, or Actions).

The following is a non-normative example of a search response returning resources:

EXAMPLE 30: Example Resource Search API Response

```
{
  "page": {
    "count": 2,
    "total": 102
  },
  "context": {
    "query_execution_time_ms": 42
  },
  "results": [
    {
      "type": "account",
      "id": "123"
    },
    {
      "type": "account",
      "id": "456"
    }
  ]
}
```

8.4 Subject Search API

The Subject Search API returns all subjects of a given type that are permitted according to the provided Action (5.3 Action), Resource (5.2 Resource), and Context (5.4 Context).

8.4.1 The Subject Search API Request

The Subject Search request is an object consisting of the following entities:

`subject`: : *REQUIRED*. The subject (or principal) of type Subject. The Subject *MUST* contain a `type`, but the Subject `id` *SHOULD* be omitted, and if present, *MUST* be ignored.

`action`: : *REQUIRED*. The action (or verb) of type Action.

`resource`: : *REQUIRED*. The resource of type Resource.

`context`: : *OPTIONAL*. Contextual data about the request.

`page`: : *OPTIONAL*. A page object for paginated requests.

§ 8.4.2 Example (non-normative) {#subject-search-example"}

The following payload defines a request for the subjects of type `user` that can perform the `can_read` action on the resource of type `account` and ID `123`.

EXAMPLE 31: Example Subject Search API Request

```
{
  "subject": {
    "type": "user"
  },
  "action": {
    "name": "can_read"
  },
  "resource": {
    "type": "account",
    "id": "123"
  },
  "context": {
    "time": "2024-10-26T01:22-07:00"
  }
}
```

The following payload defines a valid response to this request.

EXAMPLE 32: Example Subject Search API Response

```
{
  "results": [
    {
      "type": "user",
      "id": "alice@example.com"
    },
    {
      "type": "user",
      "id": "bob@example.com"
    }
  ]
}
```

§ 8.5 Resource Search API

The Resource Search API returns all resources of a given type that are permitted according to the provided Action (5.3 Action), Subject (5.1 Subject), and Context (5.4 Context).

§ 8.5.1 The Resource Search API Request

The Resource Search request is an object consisting of the following entities:

`subject`: : *REQUIRED*. The subject (or principal) of type Subject.

`action`: : *REQUIRED*. The action (or verb) of type Action.

`resource`: : *REQUIRED*. The resource of type Resource. The Resource *MUST* contain a `type`, but the Resource `id` *SHOULD* be omitted, and if present, *MUST* be ignored.

`context`: : *OPTIONAL*. Contextual data about the request.

`page`: : *OPTIONAL*. A page object for paginated requests.

§ 8.5.2 Example (non-normative) {#resource-search-example"}

The following payload defines a request for the resources of type `account` on which the subject of type `user` and ID `alice@example.com` can perform the `can_read` action.

EXAMPLE 33: Example Resource Search API Request

```
{
  "subject": {
    "type": "user",
    "id": "alice@example.com"
  },
  "action": {
    "name": "can_read"
  },
  "resource": {
    "type": "account"
  }
}
```

The following payload defines a valid response to this request.

EXAMPLE 34: Example Resource Search API Response

```
{
  "results": [
    {
      "type": "account",
      "id": "123"
    },
    {
      "type": "account",
      "id": "456"
    }
  ]
}
```

§ 8.6 Action Search API

The Action Search API returns all actions that are permitted according to the provided Subject (5.1 Subject), Resource (5.2 Resource), and Context (5.4 Context).

§ 8.6.1 The Action Search API Request

The Action Search request is an object consisting of the following entities:

`subject`: : *REQUIRED*. The subject (or principal) of type Subject.

`resource`: : *REQUIRED*. The resource of type Resource.

`context`: : *OPTIONAL*. Contextual data about the request.

`page`: : *OPTIONAL*. A page object for paginated requests.

NOTE

Unlike the Subject and Resource Search APIs, the `action` key is omitted from the Action Search request payload.

8.6.2 Example (non-normative) {#action-search-example"}

The following payload defines a request for the actions that the subject of type `user` with ID `123` may perform on the resource of type `account` and ID `123` at 01:22 AM.

EXAMPLE 35: Example Action Search API Request

```
{
  "subject": {
    "type": "user",
    "id": "alice@example.com"
  },
  "resource": {
    "type": "account",
    "id": "123"
  },
  "context": {
    "time": "2024-10-26T01:22-07:00"
  }
}
```

The following payload defines a valid response to this request.

EXAMPLE 36: Example Action Search API Response

```
{
  "results": [
    {
      "name": "can_read"
    },
    {
      "name": "can_write"
    }
  ]
}
```

9. Policy Decision Point Metadata

It is *RECOMMENDED* that PDPs provide metadata describing their configuration.

§ 9.1 Data structure

The following Policy Decision Point metadata parameters are used by this specification and are registered in the IANA "AuthZEN Policy Decision Point Metadata" registry established in 12.1 AuthZEN Policy Decision Point Metadata Registry.

§ 9.1.1 Endpoint Parameters

`policy_decision_point`: : *REQUIRED*. The Policy Decision Point identifier, which is a URL that uses the "https" scheme and has no query or fragment components. Policy Decision Point metadata is published at a location that is ".well-known" according to [*RFC8615*] derived from this Policy Decision Point identifier, as described in 9.2 Obtaining Policy Decision Point Metadata. The Policy Decision Point identifier is used to prevent Policy Decision Point mix-up attacks.

`access_evaluation_endpoint`: : *REQUIRED*. URL of Access Evaluation API endpoint

`access_evaluations_endpoint`: : *OPTIONAL*. URL of Access Evaluations API endpoint

`search_subject_endpoint`: : *OPTIONAL*. URL of Search API endpoint for subject entities

`search_action_endpoint`: : *OPTIONAL*. URL of Search API endpoint for action entities

`search_resource_endpoint`: : *OPTIONAL*. URL of Search API endpoint for resource entities

> NOTE
>
> The absence of any of these parameters is sufficient for the PEP to determine that the PDP is not capable and therefore will not return a result for the associated API.

§ 9.1.2 Capabilities Parameters

`capabilities`: : *OPTIONAL*. JSON array containing a list of registered IANA URNs referencing PDP specific capabilities.

9.1.3 Signature Parameter

In addition to JSON elements, metadata parameters *MAY* also be provided as a `signed_metadata` value, which is a JSON Web Token [*RFC7519*] that asserts metadata values about the PDP as a bundle. A set of metadata parameters that can be used in signed metadata as claims are defined in 9.1.1 Endpoint Parameters. The signed metadata *MUST* be digitally signed or MACed using JSON Web Signature [*RFC7515*] and *MUST* contain an `iss` (issuer) claim denoting the party attesting to the claims in the signed metadata.

A PEP *MAY* ignore the signed metadata if they do not support this feature. If the PEP supports signed metadata, metadata values conveyed in the signed metadata *MUST* take precedence over the corresponding values conveyed using plain JSON elements. Signed metadata is included in the Policy Decision Point metadata JSON object using this *OPTIONAL* metadata parameter:

`signed_metadata`: : A JWT containing metadata parameters about the protected resource as claims. This is a string value consisting of the entire signed JWT. A `signed_metadata` parameter *SHOULD NOT* appear as a claim in the JWT; it is *RECOMMENDED* to reject any metadata in which this occurs.

9.2 Obtaining Policy Decision Point Metadata

PDPs supporting metadata *MUST* make a JSON document containing metadata as specified in the AuthZEN Policy Decision Point Metadata Registry (12.1 AuthZEN Policy Decision Point Metadata Registry) available at a URL formed by inserting a well-known URI string between the host component and the path and/or query components, if any. The well-known URI string used is `/.well-known/authzen-configuration`.

The syntax and semantics of .well-known are defined in [*RFC8615*]. The well-known URI path suffix used is registered in the *IANA "Well-Known URIs" registry*.

An example of a PDP supporting multiple tenants will have a discovery endpoint as follows:

```
https://pdp.example.com/.well-known/authzen-configuration/tenant1
```

9.2.1 Policy Decision Point Metadata Request

A Policy Decision Point metadata document *MUST* be queried using an HTTP GET request at the previously specified URL. The consumer of the metadata would make the following request when the

resource identifier is `https://pdp.example.com`:

EXAMPLE 37

```
GET /.well-known/authzen-configuration HTTP/1.1
Host: pdp.example.com
```

§ 9.2.2 Policy Decision Point Metadata Response

The response is a set of metadata parameters about the protected resource's configuration.

A successful response *MUST* use the HTTP status code `200` and a `Content-Type` of `application/json`. Its body *MUST* be a JSON object that contains a set of metadata parameters as defined in the AuthZEN Policy Decision Point Metadata Registry (12.1 AuthZEN Policy Decision Point Metadata Registry).

Any metadata parameters in the response that are not understood by the PEP *MUST* be ignored.

Parameters that have multiple values are represented as JSON arrays. Parameters that have no values *MUST* be omitted from the response.

An error response uses the applicable HTTP status code value.

The following is a non-normative example response:

EXAMPLE 38

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

{
  "policy_decision_point": "https://pdp.example.com",
  "access_evaluation_endpoint": "https://pdp.example.com/access/v1/evalua
  "search_subject_endpoint": "https://pdp.example.com/access/v1/search/su
  "search_resource_endpoint": "https://pdp.example.com/access/v1/search/r
}
```

9.2.3 Policy Decision Point Metadata Validation

The `policy_decision_point` value returned *MUST* be identical to the Policy Decision Point identifier value into which the well-known URI string was inserted to create the URL used to retrieve the metadata. If these values are not identical, the data contained in the response *MUST NOT* be used.

The recipient *MUST* validate that any signed metadata was signed by a key belonging to the issuer and that the signature is valid. If the signature does not validate or the issuer is not trusted, the recipient *SHOULD* treat this as an error condition.

10. Transport

This specification defines an HTTPS binding using JSON serialization which *MUST* be implemented by a compliant PDP.

Additional transport bindings (e.g. gRPC or CoAP) *MAY* be defined in the future in the form of profiles, and *MAY* be implemented by a PDP.

10.1 HTTPS JSON Binding

All API requests within this binding are made via an HTTPS `POST` request.

Requests *MUST* include a `Content-Type` header with the value `application/json`, and the request body for each endpoint *MUST* be a JSON object that conforms to the corresponding request structure, as defined in .

A successful response is an HTTPS response with a status code of `200` and a `Content-Type` of `application/json`. Its body is a JSON object that conforms to the corresponding response structure, as defined in .

The request URL *MUST* be the value of the corresponding endpoint parameter, as defined in , if it is provided in the Policy Decision Point metadata (9.1.1 Endpoint Parameters). If the parameter is not provided, the URL *SHOULD* be formed by appending the default path, as defined in , to the PDP's base URL (which is the `policy_decision_point` value from the Policy Decision Point metadata, if available.

The following table provides an overview of the API endpoints defined in this binding:

API Endpoint	Default Path	Metadata Parameter	Request Schema	Response Schema
Access Evaluation	/access/v1/evaluation	access_evaluation_endpoint	6.1 The Access Evaluation API Request	6.2 The Access Evaluation API Response
Access Evaluations	/access/v1/evaluations	access_evaluations_endpoint	7.1 The Access Evaluations API Request	7.2 The Access Evaluations API Response
Subject Search	/access/v1/search/subject	search_subject_endpoint	8.4.1 The Subject Search API Request	8.3 The Search API Response
Resource Search	/access/v1/search/resource	search_resource_endpoint	8.5.1 The Resource Search API Request	8.3 The Search API Response
Action Search	/access/v1/search/action	search_action_endpoint	8.6.1 The Action Search API Request	8.3 The Search API Response

10.1.1 JSON Serialization

This section specifies the serialization of the information model entities and API schemas defined in this document to the JSON format [*RFC8259*]. The top-level element of all request and response bodies *MUST* be a JSON object (Section 4 of [*RFC8259*]). Implementations *SHOULD* also adhere to the security recommendations in JSON Payload Considerations (11.5 JSON Payload Considerations).

The data types defined in this specification are mapped to JSON types as follows:

Object: : Represented as a JSON object (Section 4 of [*RFC8259*]). The values of its members can be any valid JSON value as defined in Section 3 of [*RFC8259*], including other objects and arrays, unless specified otherwise.

Array: : Represented as a JSON array (Section 5 of [*RFC8259*]).

String: : Represented as a JSON string (Section 7 of [*RFC8259*]).

Integer: : Represented as a JSON number (Section 6 of [*RFC8259*]). Note the recommendation in 11.5 JSON Payload Considerations to not encode values that exceed IEEE 754 double-precision.

Boolean: : Represented as the JSON literals `true` or `false` (Section 3 of [*RFC8259*]).

§ 10.1.2 Error Responses

The following error responses are common to all methods of the Authorization API. The error response is indicated by an HTTPS status code (Section 15 of [*RFC9110*]) that indicates error.

The following errors are indicated by the status codes defined below:

Code	Description	HTTPS Body Content
400	Bad Request	An error message string
401	Unauthorized	An error message string
403	Forbidden	An error message string
500	Internal Error	An error message string

> NOTE
>
> HTTPS errors are returned by the PDP to indicate an error condition relating to the request or its processing; they are unrelated to the outcome of an authorization decision and are distinct from it. A successful request that results in a "deny" is indicated by a 200 OK status code with a { "decision": false } payload.

To make this concrete:

- a `401` HTTPS status code indicates that the PEP did not properly authenticate to the PDP - for example, by omitting a required `Authorization` header, or using an invalid access token.
- the PDP indicates to the PEP that the authorization request is denied by sending a response with a `200` HTTPS status code, along with a payload of `{ "decision": false }`.

§ 10.1.3 Request Identification

All requests to the API *MAY* have request identifiers to uniquely identify them. The PEP is responsible for generating the request identifier. If present, it is *RECOMMENDED* to use the HTTPS Header `X-Request-ID` as the request identifier. The value of this header is an arbitrary string. The following non-normative example describes this header:

EXAMPLE 39: Example HTTPS request with a Request Id Header

```
POST /access/v1/evaluation HTTP/1.1
Authorization: Bearer mF_9.B5f-4.1JqM
X-Request-ID: bfe9eb29-ab87-4ca3-be83-a1d5d8305716
```

When an Authorization API request contains a request identifier the PDP *MUST* include a request identifier in the response. It is *RECOMMENDED* to specify the request identifier using the HTTPS Response header `X-Request-ID`. If the PEP specified a request identifier in the request, the PDP *MUST* include the same identifier in the response to that request.

The following is a non-normative example of an HTTPS Response with this header:

EXAMPLE 40: Example HTTPS response with a Request Id Header

```
HTTP/1.1 OK
Content-Type: application/json
X-Request-ID: bfe9eb29-ab87-4ca3-be83-a1d5d8305716
```

§ 10.1.4 Examples (non-normative)

The following is a non-normative example of the HTTPS binding of the Access Evaluation Request:

EXAMPLE 41: Example of an HTTPS Access Evaluation Request

```
POST /access/v1/evaluation HTTP/1.1
Host: pdp.example.com
Content-Type: application/json
Authorization: Bearer
X-Request-ID: bfe9eb29-ab87-4ca3-be83-a1d5d8305716

{
  "subject": {
    "type": "user",
    "id": "alice@example.com"
  },
  "resource": {
    "type": "todo",
    "id": "1"
  },
  "action": {
    "name": "can_read"
  },
  "context": {
    "time": "1985-10-26T01:22-07:00"
  }
}
```

The following is a non-normative example of an HTTPS Access Evaluation Response:

EXAMPLE 42: Example of an HTTP Access Evaluation Response

```
HTTP/1.1 OK
Content-Type: application/json
X-Request-ID: bfe9eb29-ab87-4ca3-be83-a1d5d8305716

{
  "decision": true
}
```

The following is a non-normative example of a the HTTPS binding of the Access Evaluations Request:

EXAMPLE 43: Example of an HTTPS Access Evaluations Request

```
POST /access/v1/evaluations HTTP/1.1
Host: pdp.example.com
Content-Type: application/json
Authorization: Bearer
X-Request-ID: bfe9eb29-ab87-4ca3-be83-a1d5d8305716

{
  "subject": {
    "type": "user",
    "id": "alice@example.com"
  },
  "context": {
    "time": "2024-05-31T15:22-07:00"
  },
  "action": {
    "name": "can_read"
  },
  "evaluations": [
    {
      "resource": {
        "type": "document",
        "id": "boxcarring.md"
      }
    },
    {
      "resource": {
        "type": "document",
        "id": "subject-search.md"
      }
    },
    {
      "action": {
        "name": "can_edit"
      },
      "resource": {
        "type": "document",
        "id": "resource-search.md"
      }
    }
  ]
}
```

The following is a non-normative example of an HTTPS Access Evaluations Response:

EXAMPLE 44: Example of an HTTPS Access Evaluations Response

```
HTTP/1.1 OK
Content-Type: application/json
X-Request-ID: bfe9eb29-ab87-4ca3-be83-a1d5d8305716

{
  "evaluations": [
    {
      "decision": true
    },
    {
      "decision": false,
      "context": {
        "error": {
          "status": 404,
          "message": "Resource not found"
        }
      }
    },
    {
      "decision": false,
      "context": {
        "reason": "Subject is a viewer of the resource"
      }
    }
  ]
}
```

The following is a non-normative example of the HTTPS binding of the Subject Search Request:

EXAMPLE 45: Example of an HTTPS Subject Search Request

```
POST /access/v1/search/subject HTTP/1.1
Host: pdp.example.com
Content-Type: application/json
Authorization: Bearer
X-Request-ID: bfe9eb29-ab87-4ca3-be83-a1d5d8305716

{
  "subject": {
    "type": "user"
  },
  "action": {
    "name": "can_read"
  },
  "resource": {
    "type": "account",
    "id": "123"
  }
}
```

The following is a non-normative example of an HTTPS Subject Search Response:

EXAMPLE 46: Example of an HTTPS Subject Search Response

```
HTTP/1.1 OK
Content-Type: application/json
X-Request-ID: bfe9eb29-ab87-4ca3-be83-a1d5d8305716

{
  "page": {
    "next_token": "a3M9NDU2O3N6PTI="
  },
  "results": [
    {
      "type": "user",
      "id": "alice@example.com"
    },
    {
      "type": "user",
      "id": "bob@example.com"
    }
  ]
}
```

The following is a non-normative example of the HTTPS binding of the Resource Search Request:

EXAMPLE 47: Example of an HTTPS Resource Search Request

```
POST /access/v1/search/resource HTTP/1.1
Host: pdp.example.com
Content-Type: application/json
Authorization: Bearer
X-Request-ID: bfe9eb29-ab87-4ca3-be83-a1d5d8305716

{
  "subject": {
    "type": "user",
    "id": "alice@example.com"
  },
  "action": {
    "name": "can_read"
  },
  "resource": {
    "type": "account"
  }
}
```

The following is a non-normative example of an HTTPS Resource Search Response:

EXAMPLE 48: Example of an HTTPS Resource Search Response

```
HTTP/1.1 OK
Content-Type: application/json
X-Request-ID: bfe9eb29-ab87-4ca3-be83-a1d5d8305716

{
  "page": {
    "next_token": "a3M9NDU2O3N6PTI="
  },
  "results": [
    {
      "type": "account",
      "id": "123"
    },
    {
      "type": "account",
      "id": "456"
    }
  ]
}
```

The following is a non-normative example of the HTTPS binding of the Action Search Request:

EXAMPLE 49: Example of an HTTPS Action Search Request

```
POST /access/v1/search/action HTTP/1.1
Host: pdp.example.com
Content-Type: application/json
Authorization: Bearer
X-Request-ID: bfe9eb29-ab87-4ca3-be83-a1d5d8305716

{
  "subject": {
    "type": "user",
    "id": "alice@example.com"
  },
  "resource": {
    "type": "account",
    "id": "123"
  },
  "context": {
    "time": "2024-10-26T01:22-07:00"
  }
}
```

The following is a non-normative example of an HTTPS Action Search Response:

EXAMPLE 50: Example of an HTTPS Action Search Response

```
HTTP/1.1 OK
Content-Type: application/json
X-Request-ID: bfe9eb29-ab87-4ca3-be83-a1d5d8305716

{
  "page": {
    "next_token": "a3M9NDU2O3N6PTI="
  },
  "results": [
    {
      "name": "can_read"
    },
    {
      "name": "can_write"
    }
  ]
}
```

§ 11. Security Considerations

§ 11.1 Communication Integrity and Confidentiality

In the ABAC architecture, the PEP-PDP connection is the most sensitive one and needs to be secured to guarantee:

- Integrity
- Confidentiality

As a result, the connection between the PEP and the PDP *MUST* be secured using the most adequate means given the choice of transport (e.g. TLS for HTTP REST).

§ 11.2 Policy Confidentiality and Sender Authentication

Additionally, the PDP *SHOULD* authenticate the calling PEP. There are several ways authentication can be established. These ways are out of scope of this specification. They *MAY* include:

- Mutual TLS
- OAuth-based authentication
- API key

The choice and strength of either mechanism is not in scope.

Authenticating the PEP allows the PDP to avoid common attacks (such as DoS - see below) and/or reveal its internal policies. A malicious actor could craft a large number of requests to try and understand what policies the PDP is configured with. Requesting a PEP be authenticated mitigates that risk.

§ 11.3 Sender Authentication Failure

If the protected resource request does not include the proper authentication credentials, or does not have a valid authentication scheme proof that enables access to the protected resource, the resource server *MUST* respond with a 401 HTTP status code and *SHOULD* include the HTTP "WWW-Authenticate" response header field; it *MAY* include it in response to other conditions as well. The

"WWW-Authenticate" header field uses the framework defined by HTTP/1.1 [*RFC2617*] and indicates the expected authentication scheme as well as the realm that has authority for it.

The following is a non-normative example response:

```
HTTP/1.1 401 Unauthorized
WWW-Authenticate: Bearer realm="https://as.example.com"
```

§ 11.4 Trust

In ABAC, there are occasionally conversations around the trust between PEP and PDP: how can the PDP trust that the PEP is sending the correct values? The architecture of this model assumes the PDP must trust the PEP, as the PEP is ultimately responsible for enforcing the decision the PDP produces.

§ 11.5 JSON Payload Considerations

To ensure the unambiguous interpretation of JSON payloads, implementations *SHOULD* process and generate JSON payloads in a manner consistent with the I-JSON profile ([*RFC7493*]). In particular, implementations *SHOULD* ensure that:

- JSON text is encoded as UTF-8, and strings do not contain invalid Unicode sequences such as unpaired surrogates (Section 2.1 of [*RFC7493*]).
- Numeric values do not exceed the magnitude or precision supported by IEEE 754 double-precision (Section 2.2 of [*RFC7493*]).
- Member names within a JSON object are unique after processing escape characters (Section 2.3 of [*RFC7493*]).

To avoid ambiguity between a property that is absent and one that is present with a null value, properties with a value of null *SHOULD* be omitted from JSON objects.

§ 11.6 Authorization Response Integrity

The PDP *MAY* choose to sign its authorization response, ensuring the PEP can verify the integrity of the data it receives. This practice is valuable for maintaining trust in the authorization process.

The PEP can ensure that the authorization response is not tampered with by verifying the signature of the authorization decision if it is signed. It ensures response accuracy and completeness.

TLS effectively protects data in transit for a direct, point-to-point connection but does not guarantee data integrity for the full connection path between the PEP and the PDP if there are intermediaries, such as proxies or gateways.

Digital signatures offer important advantages in this context. They provide non-repudiation, allowing verification that the response genuinely originated from the PDP. Moreover, digital signatures ensure the integrity of the authorization response, confirming that its contents have not been altered in transit.

§ 11.7 Availability & Denial of Service {#security-avail-dos}}

The PDP *SHOULD* apply reasonable protections to avoid common attacks tied to request payload size, the number of requests, invalid JSON, nested JSON attacks, or memory consumption. Rate limiting is one such way to address such issues.

§ 11.8 Differences between Unsigned and Signed Metadata

Unsigned metadata is integrity protected by use of TLS at the site where it is hosted. This means that its security is dependent upon the Internet Public Key Infrastructure (PKI) [*RFC9525*]. Signed metadata is additionally integrity protected by the JWS signature applied by the issuer, which is not dependent upon the Internet PKI. When using unsigned metadata, the party issuing the metadata is the PDP itself. Whereas, when using signed metadata, the party issuing the metadata is represented by the `iss` (issuer) claim in the signed metadata. When using signed metadata, applications can make trust decisions based on the issuer that performed the signing -- information that is not available when using unsigned metadata. How these trust decisions are made is out of scope for this specification.

§ 11.9 Metadata Caching

Policy Decision Point metadata is retrieved using an HTTP GET request, as specified in 9.2.1 Policy Decision Point Metadata Request. Normal HTTP caching behaviors apply, meaning that the GET may retrieve a cached copy of the content, rather than the latest copy. Implementations should utilize HTTP caching directives such as Cache-Control with max-age, as defined in [*RFC7234*], to enable caching of retrieved metadata for appropriate time periods.

12. IANA Considerations

This specification requests IANA to take four actions: the creation of a new protocol registry group named 'AuthZEN', the establishment of two new registries within this group ('AuthZEN Policy Decision Point Metadata' and 'AuthZEN Policy Decision Point Capabilities'), the registration of a new Well-Known URI ('authzen-configuration'), and the registration of a new URN sub-namespace ('authzen').

The following registration procedure is used for the registries established by this specification.

Values are registered on a Specification Required [*RFC8126*] basis after a two-week review period on the openid-specs-authzen@lists.openid.net mailing list, following review and approval by one or more Designated Experts. However, to allow for the allocation of values prior to publication of the final version of a specification, the Designated Experts may approve registration once they are satisfied that the specification will be completed and published. However, if the specification is not completed and published in a timely manner, as determined by the Designated Experts, the Designated Experts may request that IANA withdraw the registration.

Registration requests sent to the mailing list for review should use an appropriate subject (e.g., "Request to register AuthZEN Policy Decision Point Metadata: example").

Within the review period, the Designated Experts will either approve or deny the registration request, communicating this decision to the review list and IANA. Denials should include an explanation and, if applicable, suggestions as to how to make the request successful. The IANA escalation process is followed when the Designated Experts are not responsive within 14 days.

Criteria that should be applied by the Designated Experts includes determining whether the proposed registration duplicates existing functionality, determining whether it is likely to be of general applicability or whether it is useful only for a single application, and whether the registration makes sense.

IANA must only accept registry updates from the Designated Experts and should direct all requests for registration to the review mailing list.

It is suggested that multiple Designated Experts be appointed who are able to represent the perspectives of different applications using this specification, in order to enable broadly-informed review of registration decisions. In cases where a registration decision could be perceived as creating a conflict of interest for a particular Expert, that Expert should defer to the judgment of the other Experts.

The reason for the use of the mailing list is to enable public review of registration requests, enabling both Designated Experts and other interested parties to provide feedback on proposed registrations. The reason to allow the Designated Experts to allocate values prior to publication as a final

specification is to enable giving authors of specifications proposing registrations the benefit of review by the Designated Experts before the specification is completely done, so that if problems are identified, the authors can iterate and fix them before publication of the final specification.

12.1 AuthZEN Policy Decision Point Metadata Registry

This specification asks IANA to establish the "AuthZEN Policy Decision Point Metadata" registry under the registry group "AuthZEN Parameters". The registry records the Policy Decision Point metadata parameter and a reference to the specification that defines it.

12.1.1 Registry Definition

Registry Name: AuthZEN Policy Decision Point Metadata

Registration Policy: Specification Required per [*RFC8126*]

Reference: [This Document]

12.1.2 Registration Template

Metadata Name: : The name requested (e.g., "resource"). This name is case-sensitive. Names may not match other registered names in a case-insensitive manner unless the Designated Experts state that there is a compelling reason to allow an exception.

Metadata Description: : Brief description of the metadata (e.g., "Resource identifier URL").

Change Controller: : For IETF stream RFCs, list the "IETF". For others, give the name of the responsible party. Other details (e.g., postal address, email address, home page URI) may also be included.

Specification Document(s): : Reference to the document or documents that specify the parameter, preferably including URIs that can be used to retrieve copies of the documents. An indication of the relevant sections may also be included but is not required.

§ 12.1.3 Initial Registrations

Metadata Name: : `policy_decision_point`

Metadata Description: : Base URL of the Policy Decision Point

Change Controller: : OpenID Foundation AuthZEN Working Group : openid-specs-authzen@lists.openid.net

Specification Document(s): : 9.1.1 Endpoint Parameters of [This Document]

Metadata Name: : `access_evaluation_endpoint`

Metadata Description: : URL of the Policy Decision Point's Access Evaluation API endpoint

Change Controller: : OpenID Foundation AuthZEN Working Group : openid-specs-authzen@lists.openid.net

Specification Document(s): : 9.1.1 Endpoint Parameters of [This Document]

Metadata Name: : `access_evaluations_endpoint`

Metadata Description: : URL of the Policy Decision Point's Access Evaluations API endpoint

Change Controller: : OpenID Foundation AuthZEN Working Group : openid-specs-authzen@lists.openid.net

Specification Document(s): : 9.1.1 Endpoint Parameters of [This Document]

Metadata Name: : `search_subject_endpoint`

Metadata Description: : URL of the Policy Decision Point's Search API endpoint for Subject entities

Change Controller: : OpenID Foundation AuthZEN Working Group : openid-specs-authzen@lists.openid.net

Specification Document(s): : 9.1.1 Endpoint Parameters of [This Document]

Metadata Name: : `search_resource_endpoint`

Metadata Description: : URL of the Policy Decision Point's Search API endpoint for Resource entities

Change Controller: : OpenID Foundation AuthZEN Working Group : openid-specs-authzen@lists.openid.net

Specification Document(s): : 9.1.1 Endpoint Parameters of [This Document]

Metadata Name: : `search_action_endpoint`

Metadata Description: : URL of the Policy Decision Point's Search API endpoint for Action entities

Change Controller: : OpenID Foundation AuthZEN Working Group : openid-specs-authzen@lists.openid.net

Specification Document(s): : 9.1.1 Endpoint Parameters of [This Document]

Metadata Name: : `capabilities`

Metadata Description: : Array of URNs describing specific Policy Decision Point capabilities

Change Controller: : OpenID Foundation AuthZEN Working Group : openid-specs-authzen@lists.openid.net

Specification Document(s): : 9.1.2 Capabilities Parameters of [This Document]

Metadata Name: : `signed_metadata`

Metadata Description: : JWT containing metadata parameters about the protected resource as claims.

Change Controller: : OpenID Foundation AuthZEN Working Group : openid-specs-authzen@lists.openid.net

Specification Document(s): : 9.1.3 Signature Parameter of [This Document]

12.2 Well-Known URI Registry

This specification asks IANA to register the well-known URI defined in 9.2 Obtaining Policy Decision Point Metadata in the IANA "Well-Known URIs" registry [*IANA.well-known-uris*].

12.2.1 Registry Contents

URI Suffix: : authzen-configuration

Reference: : [This Document]

Status: : permanent

Change Controller: : OpenID Foundation AuthZEN Working Group : openid-specs-authzen@lists.openid.net

Related Information: : (none)

§ 12.3 AuthZEN Policy Decision Point Capabilities Registry

This specification asks IANA to establish the "AuthZEN Policy Decision Point Capabilities" registry under the registry group "AuthZEN Parameters". The registry contains PDP-specific capabilities or features. These URNs are intended to be used in Policy Decision Point metadata discovery documents (as described in 9. Policy Decision Point Metadata) to allow a PEP to determine the supported functionality of a given PDP instance. The content of this registry will be specified by AuthZEN-compliant PDP vendors that want to declare interoperable capabilities.

§ 12.3.1 Registry Definition

Registry Name: AuthZEN Policy Decision Point Capabilities

Registration Policy: Specification Required per [*RFC8126*]

Reference: [This Document]

§ 12.3.2 Registration Template

Capability Name: : The name of the capability. This name *MUST* begin with the colon (":") character. This name is case-sensitive. Names may not match other registered names in a case-insensitive manner unless the Designated Experts state that there is a compelling reason to allow an exception.

Capability URN: The URN of the AuthZEN Policy Decision Point Capability.

Capability Description: : Brief description of the capability.

Change Controller: : OpenID Foundation AuthZEN Working Group : openid-specs-authzen@lists.openid.net

Specification Document(s): : Reference to the document or documents that specify the parameter, preferably including URIs that can be used to retrieve copies of the documents. An indication of the relevant sections may also be included but is not required.

12.4 Registration of "authzen" URN Sub-namespace

This specification asks IANA to register a new URN sub-namespace within the "IETF URN Sub-namespace for Registered Protocol Parameter Identifiers" registry defined in [*RFC3553*].

Registry Name: authzen

Specification: [This Document]

Repository: "AuthZEN Policy Decision Point Capabilities" registry (12.3 AuthZEN Policy Decision Point Capabilities Registry of [This Document])

Index value: Sub-parameters *MUST* be specified in UTF-8, using standard URI encoding where necessary.

A. Terminology

Subject

The user or machine principal for whom an authorization decision is being requested.

Resource

The target of the request; the resource about which the Authorization API is being made.

Action

The operation the `Subject` has attempted on the `Resource` in an Authorization API call.

Context

The environmental or contextual attributes for this request.

Decision

The value of the evaluation decision made by the PDP: `true` for "allow", `false` for "deny".

PDP

Policy Decision Point. The component or system that provides authorization decisions over the network interface defined here as the Authorization API.

PEP

Policy Enforcement Point. The component or system that requests decisions from the PDP and enforces access to specific requests based on the decisions obtained from the PDP.

§ B. Acknowledgements

This template uses extracts from templates written by Pekka Savola, Elwyn Davies and Henrik Levkowetz.

§ C. Index

§ C.1 Terms defined by this specification

§ C.2 Terms defined by reference

§ D. References

§ D.1 Normative references

[IANA.well-known-uris]
IANA "Well-Known URIs" registry. 2010-01-20. URL: https://www.iana.org/assignments/well-known-uris/well-known-uris.xhtml

[RFC2119]
Key words for use in RFCs to Indicate Requirement Levels. S. Bradner. IETF. March 1997. Best Current Practice. URL: https://www.rfc-editor.org/rfc/rfc2119

[RFC3339]

Date and Time on the Internet: Timestamps. G. Klyne; C. Newman. IETF. July 2002. Proposed Standard. URL: https://www.rfc-editor.org/rfc/rfc3339

[RFC3553]

An IETF URN Sub-namespace for Registered Protocol Parameters. M. Mealling; L. Masinter; T. Hardie; G. Klyne. IETF. June 2003. Best Current Practice. URL: https://www.rfc-editor.org/rfc/rfc3553

[RFC7515]

JSON Web Signature (JWS). M. Jones; J. Bradley; N. Sakimura. IETF. May 2015. Proposed Standard. URL: https://www.rfc-editor.org/rfc/rfc7515

[RFC7519]

JSON Web Token (JWT). M. Jones; J. Bradley; N. Sakimura. IETF. May 2015. Proposed Standard. URL: https://www.rfc-editor.org/rfc/rfc7519

[RFC8174]

Ambiguity of Uppercase vs Lowercase in RFC 2119 Key Words. B. Leiba. IETF. May 2017. Best Current Practice. URL: https://www.rfc-editor.org/rfc/rfc8174

[RFC8259]

The JavaScript Object Notation (JSON) Data Interchange Format. T. Bray, Ed. IETF. December 2017. Internet Standard. URL: https://www.rfc-editor.org/rfc/rfc8259

[RFC8615]

Well-Known Uniform Resource Identifiers (URIs). M. Nottingham. IETF. May 2019. Proposed Standard. URL: https://www.rfc-editor.org/rfc/rfc8615

[RFC9110]

HTTP Semantics. R. Fielding, Ed.; M. Nottingham, Ed.; J. Reschke, Ed. IETF. June 2022. Internet Standard. URL: https://httpwg.org/specs/rfc9110.html

§ D.2 Informative references

[ADR]

API Design Rules. Jasper Roes; Joost Farla. Logius. URL: https://gitdocumentatie.logius.nl/publicatie/api/adr/2.0

[JSON-LD11]

JSON-LD 1.1. Gregg Kellogg; Pierre-Antoine Champin; Dave Longley. W3C. 16 July 2020. W3C Recommendation. URL: https://www.w3.org/TR/json-ld11/

[MIM]

Metamodel Informatie Modellering. 13 juni 2024. URL: https://docs.geostandaarden.nl/mim/mim/

[NIST.SP.800-162]

Guide to Attribute Based Access Control (ABAC) Definition and Considerations . Vincent C. Hu; David Ferraiolo; Rick Kuhn; Adam Schnitzer; Kenneth Sandlin; Robert Miller; Karen Scarfone.

January 2014 . URL: https://doi.org/10.6028/NIST.SP.800-162

[RFC2617]
HTTP Authentication: Basic and Digest Access Authentication. J. Franks; P. Hallam-Baker; J. Hostetler; S. Lawrence; P. Leach; A. Luotonen; L. Stewart. IETF. June 1999. Draft Standard. URL: https://www.rfc-editor.org/rfc/rfc2617

[RFC6749]
The OAuth 2.0 Authorization Framework. D. Hardt, Ed. IETF. October 2012. Proposed Standard. URL: https://www.rfc-editor.org/rfc/rfc6749

[RFC7234]
Hypertext Transfer Protocol (HTTP/1.1): Caching. R. Fielding, Ed.; M. Nottingham, Ed.; J. Reschke, Ed. IETF. June 2014. Proposed Standard. URL: https://httpwg.org/specs/rfc7234.html

[RFC7493]
The I-JSON Message Format. T. Bray, Ed. IETF. March 2015. Proposed Standard. URL: https://www.rfc-editor.org/rfc/rfc7493

[RFC8126]
Guidelines for Writing an IANA Considerations Section in RFCs. M. Cotton; B. Leiba; T. Narten. IETF. June 2017. Best Current Practice. URL: https://www.rfc-editor.org/rfc/rfc8126

[RFC9525]
Service Identity in TLS. P. Saint-Andre; R. Salz. IETF. November 2023. Proposed Standard. URL: https://www.rfc-editor.org/rfc/rfc9525

[SAML2-CORE]
Assertions and Protocols for SAML V2.0. Scott Cantor; John Kemp; Rob Philpott; Eve Maler. 15 March 2005. URL: http://docs.oasis-open.org/security/saml/v2.0/saml-core-2.0-os.pdf

[trace-context-1]
Trace Context. Sergey Kanzhelev; Morgan McLean; Alois Reitbauer; Bogdan Drutu; Nik Molnar; Yuri Shkuro. W3C. 23 November 2021. W3C Recommendation. URL: https://www.w3.org/TR/trace-context-1/

[vc-data-model-2.0]
Verifiable Credentials Data Model v2.0. Ivan Herman; Michael Jones; Manu Sporny; Ted Thibodeau Jr; Gabe Cohen. W3C. 15 May 2025. W3C Recommendation. URL: https://www.w3.org/TR/vc-data-model-2.0/

[XACML20]
OASIS eXtensible Access Control Markup Language (XACML) Version 2.0. Tim Moses. 1 February 2005. URL: http://docs.oasis-open.org/xacml/2.0/access_control-xacml-2.0-core-spec-os.pdf

↥