

Workgroup: connect
Internet-Draft: claim-assertions-00
Published: 2 April 2020
Intended: Standards Track
Status:
Authors: A. Pulido, V. Herraiz, Ed. J. Oliva
Santander, Santander, Santander

Table of Contents

- 1. Introduction
 - 1.1. Notational Conventions
 - 1.2. Terminology
- 2. Request
- 3. Expression Language
 - 3.1. Simple Types
 - 3.2. Complex Types
 - 3.3. Array Types
 - 3.4. Multiple Assertions Example
- 4. Response
- 5. OP Metadata
- 6. Privacy Considerations
- 7. IANA Considerations
- 8. Normative References
- Appendix A. Acknowledgements
- Appendix B. Notices
- Authors' Addresses

Claim Assertions

Abstract

This specification defines a new claim that allows assertions over existing claims.

1. Introduction

In order to avoid an unnecessary leak of information, the answer to some claims may be only a boolean, verifying the claim instead of returning the actual value. As an example, assert that one is older than 18 without revealing the actual age.

Section 5.5.1 of the OpenID Connect specification [OIDC] defines a query syntax that allows for the member value of the requested claim to be a JSON object with additional information/constraints. For doing so it defines three members (essential, value and values) with special query meanings and allows for other special members to be defined. Any members that are not understood, must be ignored. This mechanism does not cover the above requirements and in this specification we will try to complement the [OIDC] specification with a richer syntax.

1.1. Notational Conventions

The key words "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "MAY", and "CAN" in this document are to be interpreted as described in "Key words for use in RFCs to Indicate Requirement Levels" [RFC2119]. These key words are not used as dictionary terms such that any occurrence of them shall be interpreted as key words and are not to be interpreted with their natural language meanings.

1.2. Terminology

This specification uses the terms "Claim", "Claim Type", "Claims Provider", "ID Token", "OpenID Provider (OP)", "Relying Party (RP)", and "UserInfo Endpoint" defined by OpenID Connect [OIDC]

2. Request

This specification defines a generic mechanism to request assertions over claims using the new element `assertion_claims`. This new element will be used inside the parameter `claims`, as specified in section 5.5 of [OIDC] as a normal claim at `id_token` or `userinfo` level. It will contain the assertions of any claims relating to the End-User.

The top level elements of the `assertion_claims` JSON object are the actual claim names with `assertion` member nested inside:

```
{
  "id_token": {
    "assertion_claims": {
      "given_name": {
        "assertion": { "eq": "William" }
      }
    }
  }
}
```

The following members are defined for every claim:

- `assertion` REQUIRED, Object: Expression that will be evaluated against the actual value of the claim.
- `purpose` OPTIONAL, String: String describing the purpose of the request to the End-User.
- `essential` OPTIONAL, Boolean: As defined at section 5.5.1 [OIDC]

Every other member that is not recognized by the OP SHOULD be ignored.

3. Expression Language

The assertion member contains the expression (a JSON object) that will be evaluated as a boolean depending on the actual value of the named claim.

This language SHOULD be defined by the OP and it MUST be discoverable at the well-known endpoint, see section [OP Metadata](#).

Recommended operations (if applicable):

- `eq`: The value is equal to the given value.
- `gt`: The value should be greater than the given value.
- `lt`: The value should be lower than the given value.
- `gte`: The value is equal or greater than the given value.
- `lte`: The value is equal or lower than the given value.
- `in`: The value is equal to one of the elements in the given list.
- `or`: The value pass any of the given expressions.
- `some`: The value is equal to any of the values within an array.
- `none`: The value is not equal to any of the values within an array.
- `every`: The value is equal to all of the values within an array.

The OP is entitled to change the specification to match any individual requirements.

3.1. Simple Types

Every claim value has a type (i.e. String, Number) and depending on this type of the claim some operators will not be valid.

In some cases the value requires some manipulation before executing the expression (e.g. phone numbers require normalization).

In the following example, `given_name` type is string and the result of the expression evaluation becomes true only if the value of the claim is William.

```
{
  "assertion_claims": {
    "given_name": {
      "assertion": { "eq": "William" }
    }
  }
}
```

In the following case the type does not match and its behavior is undefined. The OP could then return an error in this case:

```
{
  "assertion_claims": {
    "given_name": {
      "assertion": { "eq": 1701 }
    }
  }
}
```

In some cases the evaluation requires conversions. For instance, `simple_balance` contains a decimal number as a string and requires a conversion before evaluation. The following expression becomes true only if the value of the claim is greater than 1234.00.

```
{
  "assertion_claims": {
    "simple_balance": {
      "assertion": { "gt": "1234.00" }
    }
  }
}
```

If there are multiple operators (e.g. `gt` or `lte`), the expression will be evaluated as true if all operators return true. In other words, it behaves as a logical and. The following example will be true only if the claim value is greater than 1234.00 and less than or equal to 20000.00:

```
{
  "assertion_claims": {
    "simple_balance": {
      "assertion": {
        "gt": "1234.00",
        "lte": "20000.00"
      }
    }
  }
}
```

An empty assertion always returns true.

3.2. Complex Types

Some claim values are objects and to provide assertions over properties of those values, a new operator is required. This will prevent any collision between operators and property names. We will use props for that purpose.

balance claim value example:

```
{
  "balance": {
    "amount": "1200.00",
    "currency": "GBP"
  }
}
```

props is used in the following example:

```
{
  "assertion_claims": {
    "balance": {
      "assertion": {
        "props": {
          "amount": { "gt": "1000.00" },
          "currency": { "eq": "GBP" }
        }
      }
    }
  }
}
```

This expression will become true only if amount and currency expressions are true.

Any property that is not included in the expression will not affect the evaluation. For example, if currency is not included in the assertion, it will not affect the outcome.

Any assertion over a missing property returns false.

3.3. Array Types

Some complex type claims could include properties as an array. If that's the case, the proposed set of operators also includes specific ones for creating assertions using those properties.

Here is an example assertion request for a complex `bank_account` type claim which includes a property called `identifiers`, containing an array with several identifications for different schemas. The request could allow validating ownership on an account passing the identifier in Sort Code Account Number schema:

```
{
  "assertion_claims": {
    "bank_account": {
      "assertion": {
        "props": {
          "identifiers": {
            "some": {
              "props": {
                "identification": { "eq": "09012700047186" },
                "type": { "eq": "UK.SortCodeAccountNumber" }
              }
            }
          }
        }
      }
    }
  }
}
```

3.4. Multiple Assertions Example

The following is a non normative example of a request containing multiple assertions:

```
{
  "id_token": {
    "assertion_claims": {
      "given_name": {
        "assertion": { "eq": "Leonard" }
      },
      "balance": {
        "assertion": {
          "props": {
            "amount": { "gt": "1000.00" },
            "currency": { "eq": "USD" }
          }
        }
      },
      "email": {
        "assertion": { "eq": "nimoy@enterprise.fp" }
      }
    }
  }
}
```

Each assertion will be evaluated independently and the user should be able to consent and share each individual result back to the RP.

4. Response

The request will return the result of the assertion execution and any potential errors.

Implementers MUST return an object for each claim inside `assertion_claims` element with the following fields:

- `result` REQUIRED, Boolean: It indicates if the claim value meets the assertion. If the claim is not found, does not match, the OP does not understand the expression or any other problem resolving the value, then this element should be equal to null.
- `error` OPTIONAL, Any: Available information about the error resolving the assertion.

The following is a non normative example of the response:

```
{
  "assertion_claims": {
    "name": { "result": true },
    "balance": {
      "result": null,
      "error": "unknown_operator"
    },
    "address": { "result": true },
    "email": { "result": false }
  }
}
```

5. OP Metadata

The OP SHOULD advertise their capabilities with respect to assertion claims in their `openid-configuration` (see [OIDC.Discovery]) using the following new elements:

- `assertion_claims_supported`: Boolean value indicating support of assertion claims.
- `claims_in_assertion_claims_supported`: List of claims that can be included in the `assertion_claims` element.
- `assertion_claims_query_language_supported`: List of members supported in the claims included in the `assertion_claims` element.

Non normative example:

```
{
  "assertion_claims_query_language_supported": {
    "date": [
      "eq",
      "gt",
      "lt",
      "lte",
      "in"
    ],
    "decimal": [
      "eq",
      "gt",
      "lt",
      "lte",
      "in"
    ],
    "number": [
      "eq",
      "gt",
      "lt",
      "lte",
      "in"
    ],
    "object": [],
    "phone_number": [
      "eq",
      "in"
    ],
    "string": [
      "eq",
      "in"
    ],
    "boolean": [
      "eq"
    ],
    "array": [
      "none",
      "every",
      "some"
    ]
  },
  "assertion_claims_supported": true,
  "claims_in_assertion_claims_supported": {
    "total_balance": {
      "type": "object",
      "props": {
        "amount": {
          "type": "decimal"
        },
        "currency": {
          "type": "string"
        }
      }
    },
    "phone_number": {
      "type": "phone_number"
    },
    "email": {
      "type": "string"
    },
    "birthdate": {
      "type": "date"
    },
    "family_name": {
      "type": "string"
    },
    "given_name": {
      "type": "string"
    },
    "bank_account": {
      "type": "object",
      "props": {
        "id": {
          "type": "string"
        },
        "currency": {
          "type": "string"
        },
        "type": {
          "type": "string"
        },
        "identifiers": {
          "type": "array",
          "items": {
            "type": "object",
            "props": {
              "type": {
                "type": "string"
              },
              "identification": {
                "type": "string"
              }
            }
          }
        }
      }
    }
  }
}
```

6. Privacy Considerations

The `assertion_claims` can contain Personally Identifiable Information (PII). As such, OP SHOULD obtain End-User consent for the release of the information at or prior to the request in accordance with relevant regulations.

How to get such consent is out of scope for this specification.

The RP SHOULD not keep the information within `assertion_claims` unless necessary and SHOULD delete such information as soon as it is no longer necessary in accordance with relevant regulations.

7. IANA Considerations

To be done.

8. Normative References

- [OIDC] Sakimura, N., Bradley, J., Jones, M., de Medeiros, B., and C. Mortimore, "OpenID Connect Core 1.0 incorporating errata set 1", 8 November 2014, <http://openid.net/specs/openid-connect-core-1_0.html>.
- [RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", March 1997, <<https://tools.ietf.org/html/rfc2119>>.
- [OIDC.Discovery] Sakimura, N., Bradley, J., Jones, M., and E. Jay, "OpenID Connect Discovery 1.0 incorporating errata set 1", 8 November 2014, <https://openid.net/specs/openid-connect-discovery-1_0.html>.

Appendix A. Acknowledgements

We would like to thank Kai Lehmann and Kosuke Koizumi for their valuable feedback and contributions that helped to evolve this specification.

Appendix B. Notices

MIT License

Copyright (c) 2020 Grupo Santander

Permission is hereby granted, free of charge, to any person obtaining a copy of this software and associated documentation files (the "Software"), to deal in the Software without restriction, including without limitation the rights to use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies of the Software, and to permit persons to whom the Software is furnished to do so, subject to the following conditions:

The above copyright notice and this permission notice shall be included in all copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.

Authors' Addresses

Alberto Pulido Moyano

Santander

Email: alberto.pulido@santander.co.uk

Victor Herraiz Posada

Santander

Email: victor.herraiz@santander.co.uk

Jorge Oliva Fernandez

Santander

Email: Jorge.OlivaFernandez@santander.co.uk