# Designing a Nuts Use Case

A Nuts use case enables organizations to that don't directly trust each other, but rely on trusted third parties, to find each other and securely exchange data. The use case specifies which data exchanged, on which authorization grounds and how API endpoints can be found.

This book is meant for those who want to understand what decisions need to be made when designing a use case, and how to build the artifacts (e.g. Presentation Definitions) that are required by those that run the use case.

**Note:** this guide is not intended for use case implementors, who should refer to Implementing a Nuts Use Case instead.

# Authorization

This chapter describes how authorization works and what decisions impact the design of a use case.

# OAuth2 Flows and Wallets

Nuts supports 2 OAuth2 flows for acquiring an access token. The service-to-service flow and the user flow.

## Service-to-Service flow

The service-to-service flows is for data exchanges that don't require the presence of a (human) user. Credentials that are presented during this flow are subject to legal organizations (e.g. registered care organizations).

This flow uses a custom grant type called `vp_token-bearer`. Presentation requests always and only target `organization` wallets.

The flow is secured with DPoP (optional). See "Security controls" for a detailed description.

### When to use

Data exchanges for which this flow is suitable are background processes or exchanges that aren't subject to GDPR (or other local privacy regulations).

## User flow

The user flow us for data exchanges that require the presence of a (human) user. Credentials that are presented during this flow are typically a combination of:

- credentials subject to a legal organization, and
- credentials subject to a natural person, registered caregiver or employee of a legal organization.

At the time of writing, there is no governing body that issues Verifiable Credentials to (human) users, meaning the only viable option is an employee-type credential that is a self-attestation (by the organization) that the current user is an employee.

A typical example of requested credentials are a legal care organization and self-attested `EmployeeCredential` that is issued by the care organization.

This flow uses OpenID for Verifiable Presentations (OpenID4VP), draft 21 at the time of writing.

The flow is secured with JAR, PKCE and DPoP (optional). See "Security controls" for a detailed description.

### When to use

Data exchanges for which this flow is suitable are ones for which the data holder/receiver requires an identity of the end-user for logging means (Dutch norm NEN-7510) and/or GDPR compliance.

## Security controls

The following security controls are used by the OAuth2 flows:

- JWT-Secured Authorization Request (JAR, RFC9101) provides integrity protection and authenticity for the credential presentation request. Usage is enforced by the server.
- Proof Key for Code Exchange (PKCE, RFC7636) provides authenticity of the client retrieving the access token. This mitigates a MITM stealing authorization codes. Usage is enforced by the server.
- Demonstrating Proof of Possession (DPoP, RFC9449) provides authenticity of the client using the access token. This mitigates a MITM stealing access tokens. Usage is optional, to be enabled by the client.

# OAuth2 Scopes and Presentation Definition Mapping

## Scope design

When designing a system that uses OAuth2, you have to decide how scopes map to resources that the client will attempt to access. "Resource access" is typically a specific REST-style HTTP operation on a specific URL, e.g. `POST /products/staplers/1`. Things to consider when designing scopes are discussed in this section.

### Broad v.s. narrow scopes

Broad scopes are generally high level e.g., a scope that gives access to a certain use case or larger group of resources. Narrow scopes are often low-level e.g., a scope that gives read access to a specific resource, limited set of resources or operations. Examples scopes for an employee that's authorized to buy supplies for their employer:

- Very broad: `buyer`
- Broad: `buyer office` (office supplies only)
- Broad: `buyer lt-1000` (orders less than 1000 euros)
- Narrower: `buyer office:staplers` (staplers only)
- Very narrow: `buyer office:staplers:red lt-10` (red staplers only, less than 10 euros)

How scopes are mapped to operations on resources influences:

- How often clients need to request a new access token, if the previous token does not give access to a required resource.
- When access to a specific resource is authorized: when the access token is issued, or when it's used.

#### Broad, high-level Scopes

High-level, broad scopes typically give access to an entire use case, service, or group of resources. Checks that are executed before an access token is issued are limited to the Verifiable Credentials the client can present.

- Identification and authentication (user/client identity)
- General user access to the functionality (e.g. is admin, can buy supplies, etc.)

A real-life example of a broad scope is the Nuts eOverdracht use case, which specifies the following scopes:

- `eOverdracht-sender` which gives access to the receiver's services required by a care organization that wants to transfer a patient to another organization.
- `eOverdracht-receiver` which gives access to the sender's services to the transfer receiver.

However, when a resource is accessed, the system needs to verify that the scope gives access to the specific resource operation.

This type of scope is supported by the Nuts node.

## Narrow, low-level Scopes

Narrow, low-level scopes typically give access to specific operations on specific resources, e.g., reading a specific patient's medical summary.

This type of scope is **not** supported by the Nuts node, because:

- narrow scopes often contain resource identifiers, which requires wildcards/regexes in policy mapping (more on that below), which is currently not supported by the Nuts node.
- this leads to more access tokens, since each access token has a more limited use. If user authentication involves manual input (e.g., presenting a credential using a mobile wallet), user experience will deteriorate.

Another consideration is that using low-level scopes, moves most authorization decisions to the access token issuance. This is viable and supported by the Nuts node, but complicated: it requires the vendor to implement a REST API that understands Presentation Definitions.

# Policies: Mapping Scope to Authentication Subject

Nuts differentiates data exchanges that contain PII (Personally Identifiable Information) and/or medical data (e.g., Social Security Number or EHR records) and non-PII (e.g., medical condition without correlatable information, or technical identifiers). These types require different levels of authentication:

- Data exchanges containing non-PII may be exchanged by a service, with only the legal organization being authenticated.
- In data exchanges that do contain PII, both legal organization and human user must be authenticated. This means PII information cannot be exchanged without a human being present.

The authentication subject, meaning whether an organization or user must be authenticated, depends on the requested scope. This mapping is to be specified by the use case, distributed as JSON document and loaded into participating Nuts nodes.

This mapping also determines which protocol is used by the requesting party;

- exchanges with a human user always authenticate using OpenID4VP
- exchanges without human user (only the organization is authenticated) can authenticate using the OAuth2 `vp_token` grant for backend services or OpenID4VP.

## Mapping document

This section contains an example of a presentation definition mapping document as it could be specified by a use case. The Presentation Definition is described more in detail in AuthN using Verifiable Credentials.

```
{
 "zorgtoepassing": {
  "organization": {
   "format": {
    "ldp_vc": {
     "proof_type": [
      "JsonWebSignature2020"
     ]
    },
    "ldp_vp": {
     "proof_type": [
      "JsonWebSignature2020"
     ]
    },
    "jwt_vc": {
     "alg": [
      "ES256"
     ]
    },
    "jwt_vp": {
     "alg": [
      "ES256"
    ]
  }
},
"id": "pd_any_care_organization",
"name": "Care organization",
"purpose": "Finding a care organization for authorizing access to medical metadata",
"input_descriptors": [
  {
    "id": "id_nuts_care_organization_cred",
    "constraints": {
      "fields": [
        {
          "path": [
            "$.type"
          ],
          "filter": {
            "type": "string",
            "const": "NutsOrganizationCredential"
          }
        },
        {
          "id": "organization_name",
          "path": [
            "$.credentialSubject.organization.name",
            "$.credentialSubject[0].organization.name"
          ],
          "filter": {
            "type": "string"
          }
        },
        {
          "id": "organization_city",
          "path": [
            "$.credentialSubject.organization.city",
            "$.credentialSubject[0].organization.city"
          ],
          "filter": {
            "type": "string"
          }
        }
```

```
          ]
        }
      }
    ]
  },
  "user": {
    "format": {
      "ldp_vc": {
        "proof_type": [
          "JsonWebSignature2020"
        ]
      },
      "ldp_vp": {
        "proof_type": [
          "JsonWebSignature2020"
        ]
      },
      "jwt_vc": {
        "alg": [
          "ES256"
        ]
      },
      "jwt_vp": {
        "alg": [
          "ES256"
        ]
      }
    },
    "id": "pd_any_employee_credential",
    "name": "Employee",
    "purpose": "Finding an employee for authorizing access to medical metadata",
    "input_descriptors": [
      {
        "id": "id_employee_credential_cred",
        "constraints": {
          "fields": [
            {
              "path": [
                "$.type"
              ],
              "filter": {
                "type": "string",
                "const": "EmployeeCredential"
              }
            },
            {
              "id": "employee_identifier",
              "path": [
                "$.credentialSubject.identifier",
                "$.credentialSubject[0].identifier"
              ],
              "filter": {
                "type": "string"
              }
            },
            {
              "id": "employee_name",
              "path": [
                "$.credentialSubject.name",
                "$.credentialSubject[0].name"
              ],
              "filter": {
                "type": "string"
              }
            },
            {
              "id": "employee_role",
              "path": [
                "$.credentialSubject.roleName",
                "$.credentialSubject[0].roleName"
              ],
              "filter": {
                "type": "string"
              }
            }
          ]
        }
      }
    ]
  }
 }
}
```

# AuthN using Verifiable Credentials

To successfully negotiate an OAuth2 access token, the token issuer (OAuth2 Authorization Server) will ask the client to present Verifiable Credentials. Nuts uses DIF Presentation Exchange for requesting and presenting credentials during authentication. It used by service-to-service (`vp_token bearer` OAuth2 flow) and user flow (OpenID4VP). It is also used by Discovery Services to restrict what can be registered on it.

## Presentation Definition

The party requesting a presentation, typically during access token negotiation, provides a Presentation Definition to the credential wallet. The Presentation Definition specifies which credentials the wallet must provide. If the wallet can't fulfill the definition, access token negotiation will fail.

### `NutsCareOrganization` example

Below is an example Presentation Definition specifying a `NutsOrganizationCredential`, not restricted to a specific issuer. It specifies the following:

- Only JSON-LD Verifiable Credentials are supported, which must be signed through `JsonWebSignature2020`
- No restrictions on the Verifiable Presentation format
- Credential type must be `NutsOrganizationCredential`
- `credentialSubject` of the credential must an object `organization` with string properties `name` and `city`.

```
{
  "format": {
    "ldp_vc": {
      "proof_type": [
        "JsonWebSignature2020"
      ]
    }
  },
"id": "pd_any_care_organization",
"name": "Care organization",
"purpose": "Finding a care organization for authorizing access to medical metadata",
"input_descriptors": [
  {
    "id": "id_nuts_care_organization_cred",
    "constraints": {
      "fields": [
        {
          "path": [
            "$.type"
          ],
          "filter": {
            "type": "string",
            "const": "NutsOrganizationCredential"
          }
        },
        {
          "path": [
            "$.issuer"
          ],
          "filter": {
            "type": "string",
            "filter": {
              "type": "string"
            }
          }
        },
        {
          "path": [
            "$.credentialSubject.organization.name"
          ],
          "filter": {
            "type": "string"
          }
        },
        {
          "path": [
            "$.credentialSubject.organization.city"
          ],
              "filter": {
                "type": "string"
              }
            }
          ]
        }
      }
    ]
  }
```

## Authorizing Access Tokens through Presentation Exchange

The following example requires a

See the DIF Presentation Exchange specification for more information.

# Credential Trust

Authentication on Nuts heavily depends on trusted credential issuers: any attribute, revelant to the security model of the use case should be verifiable. E.g., if a party claims to be a care organization, it should be able to present a Verifiable Credential to prove it. The same applies to a user presenting their name or claiming to be a care professional.

Who should be the trusted issuer for a specific Verifiable Credential depends on the context. But generally, issuers are authoritative registries (e.g. Dutch CIBG) or even state-issued (PID of natural persons).

In practice, there are the following credential issuers:

- **Governing body issuing for a specific use case**
  - In the KIK-v use case, governed by Zorginstituut Nederland, KIK-v Beheer issues to participating organizations:
    - A credential that identifies the party as participating (care?) organization, containing a Chamber of Commerce registration number.
    - Credentials that allow a participant to perform specific SPARQL queries at another participant.
- **Use case implementors issuing with explicit trust**
  - In the eOverdracht use case, implementing software vendors issue `NutsOrganizationCredential` for their clients. Software vendors explicitly trust each other.
- **Use case participant issuing with delegated trust**
  - In the eOverdracht use case, participating care organizations issue a `NutsEmployeeCredential` to their active user. It is trusted when the organization has a trusted `NutsEmployeeCredential`

# Access Policy (TODO)

**Anti-patterns**

- **Bad**: "Clients can access /Observation, but the FHIR server has to limit it to /Observation?patient=XYZ" Requires transformation of the HTTP request at the Policy Enforcement Point.
  **Better**: TODO
- **Bad**: "Clients can update the FHIR resource at /Task/<XYZ> using an HTTP PUT, but only the status field. HTTP PUT is a replace operation, which requires the Policy Decision Point to verify whether delta of the update only updates the status field, which can't be performed atomically. Alternatively, it requires a use case-specific FHIR API, causing more implementation effort.
  **Better**: "Clients can update the status field of FHIR resource /Task/<XYZ> using an HTTP PATCH. Updates to other fields must be rejected"

# Discovery

Discovery is the act of finding parties to exchange data with.

# Endpoint Discovery

Endpoint resolving is the act of finding endpoints (e.g. the location of a REST API) given a DID.

# Using DID document services

A party often exchanges data through its API endpoints. If a client only has the party's DID, services in the DID document can be used to register API endpoints. A service in a DID document describes its type and content (`serviceEndpoint`). A typical example is the FHIR base URL of the FHIR API which is to be accessed by other parties.

## Service Type

Use cases should specify the service types when using DID document services for endpoint discovery. It's recommended to include the use case name in the service type, to avoid clashes when multiple use cases use same service type. This might lead to duplication of endpoints, but enables parties to differentiate API endpoints in case multiple use cases specify the same API family.

E.g., given the use cases "Pizza Delivery Service" and "Pasta Delivery Service" both having an "order" API, a bad approach would be:

```
{
  "services": [
    {
      "id": "#order",
      "type": "order-api",
      "serviceEndpoint": "https://example.com/api/pizza/order"
    }
  ]
}
```

An improved use case specification differentiates the service type to avoid clashes:

```
{
  "services": [
    {
      "id": "#order-pizza",
      "type": "order-pizza-api",
      "serviceEndpoint": "https://example.com/api/pizza/order"
    },
    {
      "id": "#order-pasta",
      "type": "order-pasta-api",
      "serviceEndpoint": "https://example.com/api/pasta/order"
    }
  ]
}
```

## Service Endpoint

The `serviceEndpoint` property can contain a JSON string, object, array, or nesting of these types. If a use case requires multiple API endpoints, it could use a JSON object to specify multiple API endpoints:

```
{
  "services": [
    {
      "id": "#order-pizza",
      "type": "pizza-restaurant",
      "serviceEndpoint": {
        "api": "https://example.com/api/pizza/order",
        "menu": "https://example.com/pizza-list.pdf"
      }
    }
  ]
}
```

Note that services in DID documents are not restricted to describing API endpoints only; another example use for services in DID documents could be informing about who is administering the DID document.

# Using well-known URIs

Many HTTP-based protocols (e.g. OpenID, OAuth2, SMART on FHIR) use well-known URIs to discover protocol metadata, which in turn contains API endpoints and other protocol-specific information. An alternative to DID document services could be using a well-known URI.

Although Nuts implements well-known URIs for OAuth/OpenID protocol support, it does not support exposing arbitrary data on a well-known URI. Hosting a use case-specific well-known endpoint could then be configured in a reverse proxy. But, as `did:web` DID documents can be resolved just as easily as documents on well-known URIs, it might be preferred to use services in DID documents instead.

Also note, that to be truly well-known as specified by RFC 8615, the URI must be registered at IANA. In any case, a URI must be chosen that does not conflict with existing well-known URIs.

# Designing Step-by-Step

This chapter puts the principles together by working towards the artifacts required for implementing the use case.

## Case Study: ???