

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 is 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 artefacts (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.

- [Service Discovery](#)
- [Authorization](#)
 - [OAuth2 Scopes and Presentation Definition Mapping](#)
 - [AuthN using Verifiable Credentials](#)
 - [Credential Trust](#)
 - [OAuth2 Flows and Wallets](#)
 - [Access Policy \(TODO\)](#)
- [Designing Step-by-Step](#)
- [Case Study: ???](#)

Service Discovery

A party often exchanges data through its API endpoints. If a client only has the party's name or identifier, it needs to find the API endpoints. Service Discovery via the Nuts node let clients find involved parties and additional information, like API endpoints.

Technical details can be found on [Read the Docs](#)

Service definition

Every use case MUST define a service definition. The service definition specifies which Verifiable Credentials are required, which issuers can be trusted and which registration parameters are required. Because the service definition defines the trust, this is extremely important to get right. A wrong definition may render all security measures ineffective!

Service identifier

Use cases must specify a service identifier. It's recommended to include the use case name in the identifier, to avoid clashes with other use cases. It's also recommended to add a version number or year of publication. Since the service identifiers are used in URLs, it's recommended to use lower-case letters, numbers and a limited set of special characters: `_-.:`

Examples:

- name2021
- long_namev1.0
- main:sub:v1

Discovery server

For each use case there can be only one discovery server. Each node can act as discovery server. Choose one to start with, it can be changed later on. The `endpoint` for a service definition is constructed as `https://<host>/discovery/<service_id>`.

DID methods

A service definition may limit the supported DID methods. This can be used to prevent incompatibilities when updating Nuts node versions that introduce new DID methods.

Max validity

A service definition contains a `presentation_max_validity` parameter. This parameter defines how long a registration is valid for. Clients will automatically refresh their registration if needed. This parameter can be used to tweak when registrations are removed automatically. Automatic removal is added to make sure abandoned registrations can no longer be found.

Presentation definition

The presentation definition is the most important part of the service definition. It defines which Verifiable Credentials are required by a client to present and, maybe even more important, who the issuer of those credentials may be. This defines the trust anchors for the use case.

A special credential is added for each registration in which the authorization server URL is added and where there's options for additional parameters like endpoints or feature flags.

A presentation definition uses the [Presentation Exchange](#) (or PEX for short) standard.

Required field

All constraints added in a presentation definition will become searchable by default. If a use case requires to find a participant based on a credential field then it's wise to add it as an required field.

```
"presentation_definition": {
  "id": "coffeecorner2024",
  "format": "...",
  "input_descriptors": [
    {
      "id": "NutsOrganizationCredential",
      "constraints": {
        "fields": [
          {
            "path": [
              "$.type"
            ],
            "filter": {
              "type": "string",
              "const": "NutsOrganizationCredential"
            }
          },
          {
            "id": "organization_name",
            "path": [
              "$.credentialSubject.organization.name",
              "$.credentialSubject[*].organization.name",
```

```

    ],
    "filter": {
      "type": "string"
    }
  },
  {
    "id": "organization_city",
    "path": [
      "$.credentialSubject.organization.city",
      "$.credentialSubject[*].organization.city",
    ],
    "filter": {
      "type": "string"
    }
  }
]
}
}
]
}
}

```

The `NutsOrganizationCredential` constraint example from above requires a credential to have 3 fields: `type`, `credentialSubject.organization.name` and `credentialSubject.organization.city`. The constraint on `type` also requires the value to be a string equal to `NutsOrganizationCredential`. The latter 2 field constraints also specify an `id`. The value of this identifier can be used in a search query.

note: the double path definitions have to do with the fact a VC/VP can be in JWT or JSON format.

Require registration parameters

As discussed earlier, it's possible for a client to add additional information to a registration. These registration parameters can be made required in the presentation definition. This is done the same way as for other required credential fields. The required credential type is `DiscoveryRegistrationCredential`. Below is an example of an input descriptor making the `auth_server_url` required.

```

...
{
  "id": "DiscoveryRegistrationCredential",
  "constraints": {
    "fields": [
      {
        "path": ["$.type"],

```

```

    "filter": {
      "type": "string",
      "const": "DiscoveryRegistrationCredential"
    }
  },
  {
    "id": "auth_server_url",
    "path": [
      "$.credentialSubject.authServerURL",
      "$.credentialSubject[*].authServerURL"
    ]
  }
}
...

```

Any key/value passed under `registrationParameters` in the API is transferred to the `credentialSubject` of the `DiscoveryRegistrationCredential`.

Limit issuer

Most credentials will be issued by authentic sources. These sources will have a single issuer identifier. You can limit the issuer of a credential with the following snippet:

```

"constraints": {
  "fields": [
    {
      "path": ["$.issuer"],
      "purpose": "We can only accept credentials from a trusted issuer",
      "filter": {
        "type": "string",
        "pattern": "^did:web:example.com:iam:[\\w-]$"
      }
    }
  ]
}

```

The example above uses a `regex` to limit the issuer of a credential to any subject hosted by example.com. It's possible to use a `const` instead of a regex to pin a single issuer.

Full Example

A full example can be found [here](#)

Registration parameters

Registration parameters should be used to register endpoints that are required for a service. The authorization server URL is added by default. Most services require a data endpoint, eg: a FHIR endpoint. Such an endpoint can be made required in the service definition.

Registration parameters can also be used to define a limited set of roles a party fulfills within the use case, eg: `sender` vs `receiver`. Since these values could be used to search on, a use case MUST specify the allowed values.

Definition distribution

Since the service definition defines the trust anchors, it's important to setup a secure distribution channel where clients can download the definition.

Authorization

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

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

Due to the ongoing development of personal authentication methods and associated protocols, the Nuts node currently only supports the OAuth2 `vp_token` grant for production. User authentication via OpenID4VP is experimental and usable for production. It's still required to pass user claims within the token request if a data exchange contains PII (Personally Identifiable Information) and/or medical data (e.g., Social Security Number or EHR records)

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"  
    }  
  }  
]  
}  
]  
}  
}  
}
```

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's used by the service-to-service (vp_token bearer) OAuth2 flow. 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 be 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"
        }
      }
    },
    {
      "id": "organization_name",
      "path": [
        "$.credentialSubject.organization.name"
      ],
      "filter": {
        "type": "string"
      }
    },
    {
      "id": "organization_city",
      "path": [
        "$.credentialSubject.organization.city"
      ],
      "filter": {
        "type": "string"
      }
    }
  ]
}
```

```
}  
}  
]  
}
```

The identifiers used in the field constraints will be available in the token introspection result. The key will be the field `id` and the value will be the value in the credential that matches the `path`.

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, relevant 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`

OAuth2 Flows and Wallets

Nuts supports a custom OAuth2 flows for acquiring an access token: the service-to-service flow.

Service-to-Service flow

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. User claims can be passed as tokens. If and how the user claims correspond to the organization attestations is done by the authorization step.

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

Security controls

The following security controls are used by the OAuth2 flows:

- VP-Secured Authorization Request (Nuts RFC021) provides integrity protection and authenticity for the request.
- 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.

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"

Designing Step-by-Step

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

Case Study: ???