Designing a Nuts Use Case

A Nuts use case enables organizations 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 v2 Use Case instead.

- Service Discovery
- Authorization
  - OAuth2 Scopes and Presentation Definition Mapping
  - AuthN using Verifiable Credentials
  - Credential Trust
  - X509Credential with UZI server certificates
  - OAuth2 Flows and Wallets
  - Access Policy

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.

NOTE: We plan to have the Service Discovery feature to be replaced by the Ministry of Health's Generic Function "Addressering".

Configuring the Nuts node (where definitions are loaded from, acting as Discovery Server): see 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.

A registration (Verifiable Presentation) may be at most 64 KiB, a fixed limit of the Nuts node. A registration with an X509Credential and its full certificate chain is about 16 KB.

Required field

Every text field in the `credentialSubject` of the credentials in a registration can be searched, also fields without a constraint. If a use case requires finding participants by a credential field, add a constraint on that field, so every registration has it.

Only text fields can be searched; arrays, numbers and true/false values can't. Keep this in mind when choosing the fields participants are found by.

```
"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 `id` is used as key in the `fields` of the search results. To search, use the field's path (e.g. `credentialSubject.organization.name`), not its `id`.

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`. If the presentation definition has no input descriptor for `DiscoveryRegistrationCredential`, the registration parameters (including the authorization server URL) are left out of the registration.

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 of a service definition:

```
{
  "id": "coffeecorner",
  "did_methods": ["web", "nuts"],
  "endpoint": "https://example.com/discovery/coffeecorner",
  "presentation_max_validity": 36000,
  "presentation_definition": {
    "id": "coffeecorner2024",
    "format": {
      "ldp_vc": {
        "proof_type": [
          "JsonWebSignature2020"
        ]
      },
      "jwt_vp": {
        "alg": ["ES256"]
      }
    },
    "input_descriptors": [
      {
        "id": "NutsOrganizationCredential",
```

```
    "constraints": {
      "fields": [
        {
          "path": [
            "$.type"
          ],
          "filter": {
            "type": "string",
            "const": "NutsOrganizationCredential"
          }
        },
        {
          "path": [
            "$.credentialSubject.organization.name"
          ],
          "filter": {
            "type": "string"
          }
        },
        {
          "path": [
            "$.credentialSubject.organization.city"
          ],
          "filter": {
            "type": "string"
          }
        }
      ]
    }
  }, {
    "id": "DiscoveryRegistrationCredential",
    "constraints": {
      "fields": [
        {
          "id":   "auth_server_url",
          "path": [
            "$.credentialSubject.authServerURL"
          ]
        }
      ]
```

```
        }
      }
    ]
  }
}
```

A service definition consists of:

- `id`: the unique identifier of the service
- `did_methods`: the DID methods that are allowed (optional)
- `endpoint`: the URL of the service
- `presentation_max_validity`: the maximum validity of the Verifiable Presentation in seconds
- `presentation_definition`: the presentation definition that specifies the required Verifiable Credentials (see Presentation Exchange)

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.

> Narrow scopes with an AuthZen policy decision point are supported from Nuts node v6.3.

The Nuts node supports narrow scopes next to a broad scope. The vendor runs an AuthZen policy decision point (PDP) that decides on them:

- The client requests the broad scope together with the narrow scopes, e.g. `eOverdracht-sender patient:123:read`. The broad scope is the one with a credential profile in the policy file.
- In the policy file, the broad scope sets `scope_policy` to `dynamic`.
- The Nuts node checks the presented credentials against the broad scope's presentation definition. It then asks the PDP (`policy.authzen.endpoint`) about every requested scope, and sends it the values from the credentials: the fields with an `id`, the same ones as in the token introspection response.
- If the PDP denies the broad scope, the request is denied. The other scopes are granted only when the PDP allows them.

Things to consider:

- This leads to more access tokens, since each token has a more limited use. If user authentication involves manual input (e.g. presenting a credential with a mobile wallet), the user experience gets worse.
- Authorization decisions move to the moment the token is issued, and the vendor has to run and maintain the PDP.

Without a PDP, an access token request can't contain other scopes (`profile-only`, the default), or the other scopes are granted without checks (`passthrough`).

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 not intended 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"
                }
              }
            ]
          }
        }
      ]
    }
  }
}
```

What the use case decides

The use case's policy file sets, per scope:

- which credentials the care organization must present (`organization`);
- which credential fields the data holder gets in the token introspection response: each field with an `id` becomes a claim. Give every field the data holder uses for authorization or logging its own `id`;
- optionally, which credentials the service provider must present (`service_provider`, in development).

How the Nuts node reads the policy file is described in Access Token Policy in the Nuts node documentation. That covers the file format, introspection claims, returning part of a value, the two-VP flow and `credential_selection`.

Example: claims in the token introspection response

Each constraint field with an `id` becomes a claim in the token introspection response, named by that `id`. This field in the presentation definition:

```
{
  "id": "fullName",
  "path": ["$.credentialSubject.fullName"],
  "filter": {
    "type": "string"
  }
}
```

gives this introspection response (some fields left out):

```
{
  "iss": "did:web:example.com",
  "active": true,
  "scope": "example_scope",
  "fullName": "John Doe"
}
```

Example: returning part of a value

To return only part of a text value, use a `pattern` filter with one capture group. The claim then holds what the capture group matched. The credential has `"role": "Admin level 4"`; this field returns `"admin_level": "4"`:

```
{
  "id": "admin_level",
  "path": ["$.credentialSubject.role"],
  "filter": {
    "type": "string",
    "pattern": "Admin level ([0-9])"
  }
}
```

Example: service provider credentials (in development)

A constraint field with the same `id` in the `organization` and `service_provider` presentation definitions links the two presentations: the value found in the care organization's presentation must also be the value in the service provider's. Here, the service provider must present a `ServiceProviderDelegationCredential` issued by the DID that issued the care organization's `HealthcareProviderCredential`:

```
{
  "example_delegated_scope": {
    "organization": {
      "id": "org_pd",
      "input_descriptors": [{
        "id": "hcp_credential",
        "constraints": {
          "fields": [
            { "path": ["$.type"], "filter": { "type": "string", "const":
"HealthcareProviderCredential" } },
            { "id": "delegating_hcp", "path": ["$.issuer"] }
          ]
        }
      }]
    },
    "service_provider": {
      "id": "sp_pd",
      "input_descriptors": [{
        "id": "delegation_credential",
        "constraints": {
          "fields": [
            { "path": ["$.type"], "filter": { "type": "string", "const":
"ServiceProviderDelegationCredential" } },
            { "id": "delegating_hcp", "path": ["$.issuer"] }
```

```
            ]
          }
        }]
      }
    }
}
```

This must be enabled in the Nuts node, see Two-VP flow.

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 have 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"
      }
    },
    {
      "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

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).

Credential types are defined in the credential catalog. It is the normative source for credential definitions.

Where trust is configured

- **Spec v2:** the use case decides which issuers it trusts. It sets this in the presentation definitions of its Discovery Service definition and its access policy. For example, an X509Credential is trusted by the fingerprint of the UZI CA (`ca-fingerprint`); see X509Credential with UZI server certificates.
- **Spec v1:** each Nuts node explicitly trusts issuers per credential type. Credentials such as `NutsOrganizationCredential` and `NutsEmployeeCredential` are spec v1 only.

In practice, there are the following credential issuers:

- **Governing body issuing for a specific use case**
  - In the KIK-v use case (spec v1), governed by Zorginstituut Nederland, KIK-v Beheer issues to participating organizations:
    - A credential that identifies the party as participating 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 (spec v1), 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 (spec v1), participating care organizations issue a `NutsEmployeeCredential` to their active user. It is trusted when the organization has a trusted `NutsOrganizationCredential`

- In the Huisartsinzage, PZP, and Home monitoring use cases (spec v2), parties self-issue the `X509Credential`, holding organization identity, using their CIBG UZI server certificate.
- In the LSPxNuts use cases, CIBG UZI certificates are used as backing proof for the `HealthcareProfessionalDelegationCredential`, `HealthcareProviderCredential` and `PatientEnrollmentCredential`.

X509Credential with UZI server certificates

Care organizations authenticate with an X509Credential, created from their UZI server certificate. This page explains how the certificate ends up in the credential, and what a use case's Discovery Service and policy definitions need to check.

The X509Credential is defined in the credential catalog (normative). For `did:x509` and the background, see RFC023. For how to create one, see "Get an X509Credential from a UZI certificate" in Implementing a Nuts v2 Use Case.

Why X509Credential

- **Reuses existing checks:** CIBG already checks every care organization and its URA before it issues a UZI server certificate. No new issuer is needed.
- **The key holder speaks for the care organization:** the credential is signed with the UZI certificate's private key. Ideally the care organization holds that key; in practice the vendor often manages it.
- **Links the DID to the URA:** the credential is issued to the care organization's DID (`did:web`).
- **Limit:** only organizations with a URA and a UZI server certificate can get one.

> **HealthcareProviderCredential:** the HealthcareProviderCredential (credential catalog, draft) is seen as the next version of the X509Credential. It is built from the same UZI server certificate through `did:x509`, but contains the URA as a typed identifier instead of the raw certificate fields. This page covers only the X509Credential.

Which CA to trust

UZI server certificates are issued by the UZI register (CIBG), under the PKIoverheid hierarchy. The X509Credential's issuer is a `did:x509` DID, which contains the fingerprint (`ca-fingerprint`) of one CA in the certificate's chain. A definition trusts a CA by listing its fingerprint.

The CA that issues the UZI server certificates is called the **UZI CA** below. It is an intermediate CA itself: the lowest CA in the chain, directly above the UZI server certificate (marked ◄ in the trees).

```
Production G1
Staat der Nederlanden Private Root CA - G1
└── Staat der Nederlanden Private Services CA - G1
    ├── UZI-register Private Server CA G1              ◄ ca-fingerprint
    │   └── UZI server certificate
    └── KPN, DigiCert, QuoVadis, ... (other providers, no URA)

Production G4 (from 12 November 2026)
Staat der Nederlanden - G4 Root Priv G-TLS - 2024
└── Staat der Nederlanden - G4 Intm Priv G-TLS SYS - 2024
    ├── UZI Server - G4 PKIo Priv G-TLS SYS - 2025      ◄ ca-fingerprint
    │   └── UZI server certificate
    └── KPN, DigiCert, Digidentity, ... (other providers, no URA)

Test G1
TEST Zorg CSP Private Root CA G1
└── TEST Zorg CSP Level 2 Private Services CA G1
    └── TEST UZI-register Private Server CA G1          ◄ ca-fingerprint
        └── UZI server certificate

Acceptance G4
ACCEPTATIE Zorg CSP - G4 Root Priv G-TLS - 2024
└── ACCEPTATIE Zorg CSP - G4 Intm Priv G-TLS SYS - 2024
    └── ACCEPTATIE UZI Server - G4 Priv G-TLS SYS - 2025 ◄ ca-fingerprint
        └── UZI server certificate
```

UZI CA	Environment	`ca-fingerprint` (SHA-256, base64url)
UZI-register Private Server CA G1	Production	`vdhg746H4rLH67NN1unhdxo6PF3shQunCA4-KQTb2Jc`
UZI Server - G4 PKIo Priv G-TLS SYS - 2025	Production, from 12 November 2026	`sDTL__yv54TtrvSf7601TtvHgj2NZSLT2m-Okf8oqkI`
TEST UZI-register Private Server CA G1	Test	`GwlhBZuEFlSHXSRUXQuTs3_YpQxAahColwJJj35US1A`
ACCEPTATIE UZI Server - G4 Priv G-TLS SYS - 2025	Test, acceptance	`svKzNxhS07V1dKkekWOZV0d6Ao7_qvjqXegWB32YjZE`

Rules for definitions

A definition **should** trust exactly the UZI CAs for its environment:

- **Production:** UZI-register Private Server CA G1 and UZI Server - G4 PKIo Priv G-TLS SYS - 2025. CIBG starts issuing G4 certificates on 12 November 2026; G1 certificates stop working on 12 November 2028. Until then, trust both.
- **Test and acceptance:** TEST UZI-register Private Server CA G1 and ACCEPTATIE UZI Server - G4 Priv G-TLS SYS - 2025.

A definition **should not** trust:

- a CA higher in the chain than the UZI CA (the "Staat der Nederlanden" or "Zorg CSP" root and intermediate CAs): those also issue certificates through other providers (e.g. KPN, DigiCert), which don't contain a URA;
- a test or acceptance CA in a production definition: test certificates are easy to get.

Deviate only with a reason, and write it down in the use case's IG.

From certificate to credential

These certificate fields end up in the X509Credential:

Certificate field	Credential field	Contains
`subject.O`	`credentialSubject.subject.O`	Name of the care organization
`subject.L`	`credentialSubject.subject.L`	City
`san.otherName`	`credentialSubject.san.otherName`	UZI number, URA and more, see below

The URA in `san.otherName`

`san.otherName` is one string with seven parts, separated by `-`:

```
<OID CA>-<version>-<UZI number>-<pass type>-<URA>-<role>-<AGB code>
2.16.528.1.1007.99.2110-1-900025039-S-90000382-00.000-00000000
```

Part	Example	Meaning
OID CA	`2.16.528.1.1007.99.2110`	The CIBG
Version	`1`	Version of this format
UZI number	`900025039`	Number of the certificate holder, same as `subject.serialNumber`
Pass type	`S`	Always `S` (server)
URA	`90000382`	The care organization's URA ("abonneenummer")
Role	`00.000`	Always `00.000`

Part	Example	Meaning
AGB code	`00000000`	The care organization's AGB code

To get the URA in a definition, use a pattern with a capture group on this field. The node then returns only the captured part, e.g. in search results and token introspection:

```
{
  "id": "organization_ura",
  "path": ["$.credentialSubject.san.otherName"],
  "filter": { "type": "string", "pattern": "^[0-9.]+-\\d+-\\d+-S-(\\d+)-00\\.000-\\d+$" }
}
```

Appendix: examples

Fictional care organization "De Regenboog" in Hengelo, URA `90000380`, with a UZI server certificate from the TEST UZI CA.

did:x509 DID

```
did:x509:0:sha256:GwlhBZuEFlSHXSRUXQuTs3_YpQxAahColwJJj35US1A::subject:O:De%20Regenboog:L:Hengelo::san:otherName:2.16.528.1.1007.99.2110-1-900030787-S-90000380-00.000-11223344
```

Part	Meaning
`sha256:GwlhBZuE…`	`ca-fingerprint`: SHA-256 hash of the DER-encoded TEST UZI-register Private Server CA G1 certificate, base64url-encoded
`subject:O:De%20Regenboog:L:Hengelo`	Certificate fields `subject.O` (OID 2.5.4.10) and `subject.L` (OID 2.5.4.7), URL-encoded
`san:otherName:…`	Certificate field `san.otherName` (OID 2.5.5.5), with the URA `90000380`

Resolved DID document

The node builds the DID document from the certificate chain in the credential's JWT header (`x5c`):

```
{
  "@context": ["https://www.w3.org/ns/did/v1"],
  "id":
"did:x509:0:sha256:GwlhBZuEFlSHXSRUXQuTs3_YpQxAahColwJJj35US1A::subject:O:De%20Regenboog:L:Hengelo::san:otherName:2.16.528.1.1007.99.2110-1-900030787-S-90000380-00.000-11223344",
  "controller":
["did:x509:0:sha256:GwlhBZuEFlSHXSRUXQuTs3_YpQxAahColwJJj35US1A::subject:O:De%20Regenboog:L:He
```

```
ngelo::san:otherName:2.16.528.1.1007.99.2110-1-900030787-S-90000380-00.000-11223344"],
  "verificationMethod": [
    {
      "id":
"did:x509:0:sha256:GwlhBZuEFlSHXSRUXQuTs3_YpQxAahColwJJj35US1A::subject:O:De%20Regenboog:L:Hen
gelo::san:otherName:2.16.528.1.1007.99.2110-1-900030787-S-90000380-00.000-11223344#0",
      "type": "JsonWebKey2020",
      "controller":
"did:x509:0:sha256:GwlhBZuEFlSHXSRUXQuTs3_YpQxAahColwJJj35US1A::subject:O:De%20Regenboog:L:Hen
gelo::san:otherName:2.16.528.1.1007.99.2110-1-900030787-S-90000380-00.000-11223344",
      "publicKeyJwk": { "kty": "RSA", "n": "<modulus>", "e": "AQAB" }
    }
  ],
  "assertionMethod":
["did:x509:0:sha256:GwlhBZuEFlSHXSRUXQuTs3_YpQxAahColwJJj35US1A::subject:O:De%20Regenboog:L:He
ngelo::san:otherName:2.16.528.1.1007.99.2110-1-900030787-S-90000380-00.000-11223344#0"]
}
```


X509Credential (JWT payload)


```
{
  "iss":
"did:x509:0:sha256:GwlhBZuEFlSHXSRUXQuTs3_YpQxAahColwJJj35US1A::subject:O:De%20Regenboog:L:Hen
gelo::san:otherName:2.16.528.1.1007.99.2110-1-900030787-S-90000380-00.000-11223344",
  "sub": "did:web:nuts.vendor.example:iam:deregenboog",
  "vc": {
    "@context": ["https://www.w3.org/2018/credentials/v1", "https://nuts.nl/credentials/v1"],
    "type": ["VerifiableCredential", "X509Credential"],
    "credentialSubject": {
      "id": "did:web:nuts.vendor.example:iam:deregenboog",
      "subject": { "O": "De Regenboog", "L": "Hengelo" },
      "san": { "otherName": "2.16.528.1.1007.99.2110-1-900030787-S-90000380-00.000-11223344" }
    }
  }
}
```


The JWT header contains the full certificate chain in `x5c`.

Sources for the CA certificates: production and acceptance on zorgcsp.nl.

OAuth2 Flows and Wallets

Nuts supports a custom OAuth2 flow 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

Anti-patterns

- **Bad**: "Clients can update the FHIR resource at /Task/ 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/ using an HTTP PATCH. Updates to other fields must be rejected"