Technology

Background on the technology Nuts uses, such as Decentralized Identifiers (DIDs). For deploying and operating the Nuts node, see the Nuts node documentation.

- Decentralized Identifiers
- JSON-LD

Decentralized Identifiers

Nuts uses W3C Decentralized Identifiers (DIDs) to identify parties, such as care organizations and vendors.

A DID is an identifier (e.g. `did:web:example.com:iam:1234`) that points to a *DID document*. The DID document lists:

- the public keys of the party: to verify signatures (credentials, presentations, access token requests) and to encrypt data for it;
- optionally, services (endpoints) of the party.

The party holds the private keys. Only it can sign with them.

A DID on its own says nothing about who the party is: anyone can create one. A DID gets meaning through Verifiable Credentials issued to it. For example, an X509Credential links the DID to the care organization's UZI server certificate (URA number, name). See Credential Trust.

DID methods

How a DID document is created, stored and found is defined by its *DID method*: the part after `did:`. Nuts uses two methods:

	`did:web`	`did:nuts`
Used by	Spec v2 use cases (e.g. Huisartsinzage, 360-graden-clientbeeld)	Spec v1 use cases (e.g. eOverdracht, KIK-v)
Format	`did:web:<domain>:iam:<id>`	`did:nuts:<id>`
Where the DID document lives	On the Nuts node of the party, served over HTTPS	On every Nuts node, copied through the Nuts network
How others find it	HTTPS request to the party's domain	Lookup in their own node
Trust in the document comes from	The domain and its TLS certificate, checked against the operating system's CA bundle	The signature of the key the DID was derived from
Specification	did:web Method Specification	Nuts RFC006

did:web

A `did:web` DID contains a domain name. The DID document is a file on that domain:

- `did:web:example.com:iam:1234` resolves to `https://example.com/iam/1234/did.json`.

The Nuts node creates the DID on its configured domain (`url` in the node configuration) and serves the DID document on its public `/iam` endpoint.

Things to know:

- The domain is part of the identity. Changing the domain means new DIDs, and all credentials must be issued again. Choose a stable domain that you own; see Choosing a domain name.
- Whoever controls the domain controls the DID documents. Protect the domain against takeover; see Security considerations.
- There is no network to join: the other party only needs to reach the domain over HTTPS.

did:nuts

A `did:nuts` DID is derived from the public key it was created with (its SHA-256 thumbprint). The DID document is signed and sent over the Nuts network (gRPC), so every Nuts node has a copy of every `did:nuts` DID document and resolves it locally.

Things to know:

- To use `did:nuts`, the node must join the Nuts network, with a TLS certificate from a CA in the other nodes' trust store (`tls.truststorefile`).
- Spec v1 use cases list their endpoints as services in the DID document, e.g. the `NutsComm` endpoint of the node.
- The Nuts network is deprecated. New use cases use `did:web`.

Subjects: one party, several DIDs

From Nuts node v6, the node groups the DIDs of one party under a *subject*. When you create a subject, the node creates a DID for each enabled DID method (`didmethods` in the node configuration; default: `web` and `nuts`). Credentials, Discovery Service registrations and access token requests work per subject; the node picks the DID that the other party accepts.

A use case can limit the DID methods it accepts, e.g. with `did_methods` in the Discovery Service definition.

Services/Endpoints: Discovery Service instead of the DID document

- Spec v1 use cases register their endpoints as services in the `did:nuts` DID document. The endpoints are usually on the vendor's DID document, and the care organizations' DID documents refer to them. These services are managed with the DIDMan API. After upgrading the Nuts node to v6, you can manage them with the VDR v2 API instead (see

the next bullet).

- Spec v2 use cases register their endpoints at the use case's Discovery Service, as part of the registration. So `did:web` DID documents contain few services, if any. If a use case still needs a service in the DID document, the VDR v2 API adds it to all DIDs of the subject. Resolving it returns the service as it is: unlike DIDMan, it doesn't follow references to another DID document.

> Once you manage services with the VDR v2 API, you must not use the DIDMan API or the VDR v1 API anymore. VDR v2 keeps the DID documents of a subject (e.g. its `did:web` and `did:nuts` DIDs) in sync. DIDMan and VDR v1 only change the `did:nuts` DID document, so the subject's DID documents get out of sync.

Other DID methods

The Nuts node can also resolve (but not create) DIDs of these methods:

- `did:x509`: a DID based on an X.509 certificate chain, e.g. of a UZI server certificate. The DID names a CA in the chain (by its fingerprint) and fields the certificate must have, e.g. the URA. The chain itself comes with the credential, and the node builds the DID document from it, so nobody hosts it. The chain is checked against the CA named in the DID, not against the node's TLS trust store or the operating system's CA bundle; whether that CA is trusted is set by the use case (`ca-fingerprint`). It is the issuer of the X509Credential. Why certificates are used this way: see UZI certificates as a source of identity; trusting the CA: see X509Credential with UZI server certificates.
- `did:key` and `did:jwk`: a DID that contains the public key itself.

See Supported protocols and formats for the full list.

JSON-LD

JSON-LD (JSON Linked Data) is JSON with a `@context`: a list of contexts, named by URL, that define what each field and type means. Verifiable Credentials can be written in JSON-LD.

Why it matters for credentials

Credentials come in two formats:

	JSON-LD (`ldp_vc`)	**JWT (`jwt_vc`)**
Signature covers	The credential after JSON-LD processing	The JWT itself
Every field must be defined in a context	Yes	No
Issuer and verifier need the contexts	Yes, the same ones	No

For a JSON-LD credential, the signature is made over a normalized form of the credential. To make that form, the node expands every field using the contexts. A field or type that no context defines makes issuing or verifying fail.

Example: a HealthcareProviderRoleTypeCredential has the field `roleCodeNL`, which no JSON-LD context defines. Issuing it as `ldp_vc` fails; issuing it as `jwt_vc` works.

For new use cases, JWT credentials are simpler: no contexts to publish, agree on and configure.

Contexts in the Nuts node

A context URL is an identifier: a credential names it in `@context`, but it doesn't have to resolve. The Nuts node maps each URL to a copy it includes, so it never fetches these:

Context URL	**Defines**
`https://www.w3.org/2018/credentials/v1`	W3C Verifiable Credentials
`https://w3c-ccg.github.io/lds-jws2020/contexts/lds-jws2020-v1.json`	JsonWebSignature2020 proofs
`https://w3id.org/vc/status-list/2021/v1`	StatusList2021 revocation
`https://schema.org`	schema.org terms
`https://nuts.nl/credentials/v1`	NutsOrganizationCredential, NutsEmployeeCredential, NutsAuthorizationCredential, DiscoveryRegistrationCredential, NutsSelfSignedPresentation, CredentialRevocation

Context URL	Defines
`https://nuts.nl/credentials/2024`	NutsUraCredential

A use case with its own JSON-LD credential types publishes its own context. Every node in the use case must then be configured with it: either map the context URL to a local file, or allow the node to fetch it from that URL. See Custom credential configuration in the Nuts node documentation.