Decentralized Identifiers (DIDs) v1.1

Core architecture, data model, and representations

W3C Candidate Recommendation Snapshot 05 March 2026

▼ **More details about this document**

This version:
https://www.w3.org/TR/2026/CR-did-1.1-20260305/

Latest published version:
https://www.w3.org/TR/did-1.1/

Latest editor's draft:
https://w3c.github.io/did/

History:
https://www.w3.org/standards/history/did-1.1/
Commit history

Test suite:
https://github.com/w3c/did-test-suite/

Implementation report:
https://w3c.github.io/did-test-suite/

Editors:
Manu Sporny (Digital Bazaar) (v1.0, v1.1)
Dmitri Zagidulin (Invited Expert) (v1.1)

Former editors:
Amy Guy (Digital Bazaar) (v1.0)
Markus Sabadello (Danube Tech) (v1.0)
Drummond Reed (Evernym/Avast) (v1.0)

Authors:
Manu Sporny (Digital Bazaar)
Dave Longley (Digital Bazaar)
Markus Sabadello (Danube Tech)
Drummond Reed (Evernym/Avast)
Orie Steele (Transmute)
Christopher Allen (Blockchain Commons)

Feedback:
GitHub w3c/did (pull requests, new issue, open issues)

public-did-wg@w3.org with subject line `[did-1.1] … message topic …` (archives)

Related Documents

DID Use Cases and Requirements
DID Extensions
DID Core Implementation Report

Abstract

Decentralized identifiers (DIDs) are a new type of identifier that enables verifiable, decentralized digital identity. A DID refers to any subject (e.g., a person, organization, thing, data model, abstract entity, etc.) as determined by the controller of the DID. In contrast to typical, federated identifiers, DIDs have been designed so that they may be decoupled from centralized registries, identity providers, and certificate authorities. Specifically, while other parties might be used to help enable the discovery of information related to a DID, the design enables the controller of a DID to prove control over it without requiring permission from any other party. DIDs are URIs that associate a DID subject with a DID document allowing trustable interactions associated with that subject.

Each DID document can express cryptographic material, verification methods, or services, which provide a set of mechanisms enabling a DID controller to prove control of the DID. Services enable trusted interactions associated with the DID subject. A DID might provide the means to return the DID subject itself, if the DID subject is an information resource such as a data model.

This document specifies the DID syntax, a common data model, core properties, serialized representations, DID operations, and an explanation of the process of resolving DIDs to the resources that they represent.

Status of This Document

This section describes the status of this document at the time of its publication. A list of current W3C publications and the latest revision of this technical report can be found in the W3C standards and drafts index.

The W3C Decentralized Identifier Working Group has published this document as a W3C Candidate Recommendation and is requesting that software developers and

DID Method specification authors provide experimental implementations designed to test the implementability of all of the features in this document.

Implementers are cautioned that any open class 1, 2, or 3 issues listed in the issue tracker might result in changes to the specification.

To exit the W3C Candidate Recommendation phase, the W3C DID Working Group requires the following:

1. For normative statements that are machine testable, at least two conforming implementations per feature; that is, when a conforming producer implements a feature, at least two conforming consumers are able to consume the feature and for every feature that is consumed, there are at least two conforming producers that produce the feature.
2. For normative statements that are not machine testable, at least two demonstrations of implementation per feature.
3. The *Decentralized Identifier Resolution (DID Resolution) v0.3* specification has met its exit criteria for the W3C Candidate Recommendation phase.

A feature is defined as one or more functionally related normative statements in the specification. For this specification, interoperability is defined as normative statements, such as those referring to data model properties and their values, being interpreted in the same way between two different DID Methods.

The resolution of a specific DID Method is out of scope for this specification, and is covered by the *Decentralized Identifier Resolution (DID Resolution) v0.3* specification.

This document was published by the Decentralized Identifier Working Group as a Candidate Recommendation Snapshot using the Recommendation track.

Publication as a Candidate Recommendation does not imply endorsement by W3C and its Members. A Candidate Recommendation Snapshot has received wide review, is intended to gather implementation experience, and has commitments from Working Group members to royalty-free licensing for implementations.

This Candidate Recommendation is not expected to advance to Recommendation any earlier than 05 April 2026.

This document was produced by a group operating under the W3C Patent Policy. W3C maintains a public list of any patent disclosures made in connection with the deliverables of the group; that page also includes instructions for disclosing a patent. An individual who has actual knowledge of a patent that the individual

believes contains Essential Claim(s) must disclose the information in accordance with section 6 of the W3C Patent Policy.

This document is governed by the 18 August 2025 W3C Process Document.

Table of Contents

§ 1. Introduction

This section is non-normative.

As individuals and organizations, many of us use globally unique identifiers in a wide variety of contexts. They serve as communications addresses (telephone numbers, email addresses, usernames on social media), ID numbers (for passports, drivers licenses, tax IDs, health insurance), and product identifiers (serial numbers, barcodes, RFIDs). URIs (Uniform Resource Identifiers) are used for resources on the Web and each web page you view in a browser has a globally unique URL (Uniform Resource Locator).

The vast majority of these globally unique identifiers are not under our control. They are issued by external authorities that decide who or what they refer to and when they can be revoked. They are useful only in certain contexts and recognized only by certain bodies not of our choosing. They might disappear or cease to be valid with the failure of an organization. They might unnecessarily reveal personal information. In many cases, they can be fraudulently replicated and asserted by a malicious third-party, which is more commonly known as "identity theft".

The Decentralized Identifiers (DIDs) defined in this specification are a new type of globally unique identifier. They are designed to enable individuals and organizations to generate their own identifiers using systems they trust. These new identifiers enable entities to prove control over them by authenticating using cryptographic proofs such as digital signatures.

Since the generation and assertion of Decentralized Identifiers is entity-controlled, each entity can have as many DIDs as necessary to maintain their desired separation of identities, personas, and interactions. The use of these identifiers can be scoped appropriately to different contexts. They support interactions with other people, institutions, or systems that require entities to identify themselves, or things they control, while providing control over how much personal or private data should be revealed, all without depending on a central authority to guarantee the continued existence of the identifier. These ideas are explored in the DID Use Cases document [*DID-USE-CASES*].

This specification does not presuppose any particular technology or cryptography to underpin the generation, persistence, resolution, or interpretation of DIDs. For example, implementers can create Decentralized Identifiers based on identifiers registered in federated or centralized identity management systems. Indeed, almost all types of identifier systems can add support for DIDs. This creates an interoperability bridge between the worlds of centralized, federated, and decentralized identifiers. This also enables implementers to design specific types of DIDs to work with the computing infrastructure they trust, such as distributed ledgers, decentralized file systems, distributed databases, and peer-to-peer networks.

This specification is for:

- Anyone that wants to understand the core architectural principles that are the foundation for Decentralized Identifiers;
- Software developers that want to produce and consume Decentralized Identifiers and their associated data formats;
- Systems integrators that want to understand how to use Decentralized Identifiers in their software and hardware systems;
- Specification authors that want to create new DID infrastructures, known as DID methods, that conform to the ecosystem described by this document.

In addition to this specification, readers might find the Use Cases and Requirements for Decentralized Identifiers [*DID-USE-CASES*] document useful.

§ 1.1 A Simple Example

This section is non-normative.

A DID is a simple text string consisting of three parts: 1) the `did` URI scheme identifier, 2) the identifier for the DID method, and 3) the DID method-specific identifier.

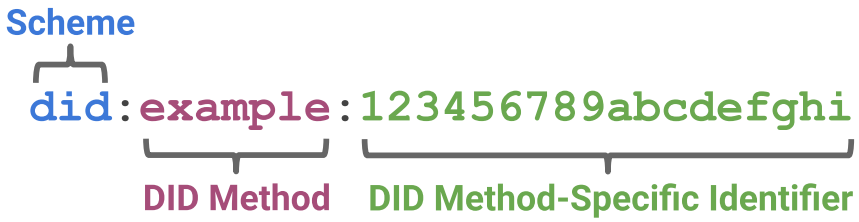

Figure 1 A simple example of a decentralized identifier (DID)

The example DID above resolves to a DID document. A DID document contains information associated with the DID, such as ways to cryptographically authenticate a DID controller.

EXAMPLE 1: A simple DID document

```
{
  "@context": "https://www.w3.org/ns/did/v1.1",
  "id": "did:example:123456789abcdefghi",
  "authentication": [{
    // used to authenticate as did:...fghi
    "id": "did:example:123456789abcdefghi#keys-1",
    "type": "Multikey",
    "controller": "did:example:123456789abcdefghi",
    "publicKeyMultibase": "z6MkmM42vxfqZQsv4ehtTjFFxQ4sQKS2w6WR7emozFAn
  }]
}
```

§ 1.2 Design Goals

This section is non-normative.

Decentralized Identifiers are a component of larger systems, such as the Verifiable Credentials ecosystem [*VC-DATA-MODEL*], which influenced the design goals for this specification. The design goals for Decentralized Identifiers are summarized here.

Goal	Description
Decentralization	Eliminate the requirement for centralized authorities or single points of failure in identifier management, including the registration of globally unique identifiers, public verification keys, services, and other information.
Control	Give entities, both human and non-human, the power to directly control their digital identifiers without the need to rely on external authorities.
Privacy	Enable entities to control the privacy of their information, including minimal, selective, and progressive disclosure of attributes or other data.
Security	Enable sufficient security for requesting parties to depend on DID documents for their required level of assurance.
Proof-based	Enable DID controllers to provide cryptographic proof when interacting with other entities.
Discoverability	Make it possible for entities to discover DIDs for other entities, to learn more about or interact with those entities.
Interoperability	Use interoperable standards so DID infrastructure can make use of existing tools and software libraries designed for interoperability.
Portability	Be system- and network-independent and enable entities to use their digital identifiers with any system that supports DIDs and DID methods.
Simplicity	Favor a reduced set of simple features to make the technology easier to understand, implement, and deploy.

Goal	Description
Extensibility	Where possible, enable extensibility provided it does not greatly hinder interoperability, portability, or simplicity.

§ 1.3 Architecture Overview

This section is non-normative.

This section provides a basic overview of the major components of Decentralized Identifier architecture.

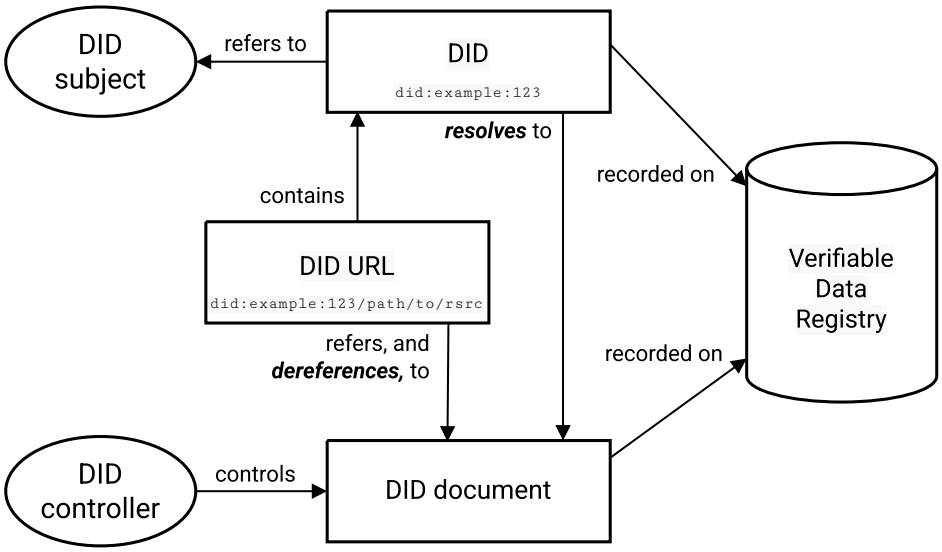

Figure 2 Overview of DID architecture and the relationship of the basic components. See also: narrative description.

DIDs and DID URLs

A Decentralized Identifier, or DID, is a URI composed of three parts: the scheme `did:`, a method identifier, and a unique, method-specific identifier specified by the DID method. DIDs are resolvable to DID documents. A DID URL extends the syntax of a basic DID to incorporate other standard URI components such as path, query, and fragment in order to locate a particular resource—for example, a cryptographic public key inside a DID document, or a resource external to the DID document. These concepts are elaborated upon in 3.1 DID Syntax and 3.2 DID URL Syntax.

DID subjects

The subject of a DID is, by definition, the entity identified by the DID. The DID subject might also be the DID controller. Anything can be the subject of a DID:

person, group, organization, thing, or concept. This is further defined in 5.1.1 DID Subject.

DID controllers

The controller of a DID is the entity (person, organization, or autonomous software) that has the capability—as defined by a DID method—to make changes to a DID document. This capability is typically asserted by the control of a set of cryptographic keys used by software acting on behalf of the controller, though it might also be asserted via other mechanisms. Note that a DID might have more than one controller, and the DID subject can be the DID controller, or one of them. This concept is documented in 5.1.2 DID Controller.

Verifiable data registries

In order to be resolvable to DID documents, DIDs are typically recorded on an underlying system or network of some kind. Regardless of the specific technology used, any such system that supports recording DIDs and returning data necessary to produce DID documents is called a verifiable data registry. Examples include distributed ledgers, decentralized file systems, databases of any kind, peer-to-peer networks, and other forms of trusted data storage. This concept is further elaborated upon in 7. Methods.

DID documents

DID documents contain information associated with a DID. They typically express verification methods, such as cryptographic public keys, and services relevant to interactions with the DID subject. The generic properties supported in a DID document are specified in 5. Core Properties. A DID document can be serialized to a byte stream (see 6. Representations). The properties present in a DID document can be updated according to the applicable operations outlined in 7. Methods.

DID methods

DID methods are the mechanism by which a particular type of DID and its associated DID document are created, resolved, updated, and deactivated. DID methods are defined using separate DID method specifications as defined in 7. Methods.

DID resolvers and DID resolution

A DID resolver is a system component that takes a DID as input and produces a conforming DID document as output. This process is called DID resolution. The steps for resolving a specific type of DID are defined by the relevant DID method specification. The process of DID resolution is elaborated upon in [*DID-RESOLUTION*].

DID URL dereferencers and DID URL dereferencing

A DID URL dereferencer is a system component that takes a DID URL as input and produces a resource as output. This process is called DID URL

dereferencing. The process of DID URL dereferencing is elaborated upon in [*DID-RESOLUTION*].

§ 1.4 Conformance

As well as sections marked as non-normative, all authoring guidelines, diagrams, examples, and notes in this specification are non-normative. Everything else in this specification is normative.

The key words *MAY*, *MUST*, *MUST NOT*, *RECOMMENDED*, *SHOULD*, and *SHOULD NOT* in this document are to be interpreted as described in BCP 14 [*RFC2119*] [*RFC8174*] when, and only when, they appear in all capitals, as shown here.

This document contains examples that contain JSON and JSON-LD content. Some of these examples contain characters that are invalid, such as inline comments (`//`) and the use of ellipsis (`...`) to denote information that adds little value to the example. Implementers are cautioned to remove this content if they desire to use the information as valid JSON or JSON-LD.

Some examples contain terms, both property names and values, that are not defined in this specification. These are indicated with a comment (`// external (property name|value)`). Such terms, when used in a DID document, are expected to be registered in the repository of DID Extensions [*DID-EXTENSIONS*] with links to both a formal definition and a JSON-LD context.

Interoperability of implementations for DIDs and DID documents is tested by evaluating an implementation's ability to create and parse DIDs and DID documents that conform to this specification. Interoperability for producers and consumers of DIDs and DID documents is provided by ensuring the DIDs and DID documents conform. Interoperability for DID method specifications is provided by the details in each DID method specification. It is understood that, in the same way that a web browser is not required to implement all known URI schemes, conformant software that works with DIDs is not required to implement all known DID methods. However, all implementations of a given DID method are expected to be interoperable for that method.

A ***conforming DID*** is any concrete expression of the rules specified in 3. Identifier which complies with relevant normative statements in that section.

A ***conforming DID document*** is any concrete expression of the data model described in this specification which complies with the relevant normative statements in 4. Data Model and 5. Core Properties. A serialization format for the

conforming document is deterministic, bi-directional, and lossless, as described in 6. Representations.

A ***conforming producer*** is any algorithm realized as software and/or hardware that generates conforming DIDs or conforming DID Documents and complies with the relevant normative statements in 6. Representations.

A ***conforming consumer*** is any algorithm realized as software and/or hardware that consumes conforming DIDs or conforming DID documents and complies with the relevant normative statements in 6. Representations.

A ***conforming DID method*** is any specification that complies with the relevant normative statements in 7. Methods.

§ 1.5 Audience

This section is non-normative.

This specification has two primary audiences: implementers of conformant DID methods; and implementers of systems and services that wish to interact and interface with DIDs. The intended audience includes, but is not limited to, software architects, data modelers, application developers, service developers, testers, operators, and user experience (UX) specialists. Other people involved in a broad range of standards efforts related to decentralized identity, verifiable credentials, and secure storage might also be interested in reading this specification.

§ 2. Terminology

This section is non-normative.

This section defines the terms used in this specification and throughout decentralized identifier infrastructure. A link to these terms is included whenever they appear in this specification.

amplification attack
: A class of attack where the attacker attempts to exhaust a target system's CPU, storage, network, or other resources by providing small, valid inputs into the system that result in damaging effects that can be exponentially more costly to process than the inputs themselves.

***decentralized identifier* (DID)**

A globally unique persistent identifier that does not require a centralized registration authority and is often generated and/or registered cryptographically. The generic format of a DID is defined in 3.1 DID Syntax. A specific DID scheme is defined in a DID method specification. Many—but not all—DID methods make use of distributed ledger technology (DLT) or some other form of decentralized network.

DID controller

An entity that has the capability to make changes to a DID document. A DID might have more than one DID controller. The DID controller(s) can be denoted by the optional `controller` property at the top level of the DID document. Note that a DID controller might be the DID subject.

DID delegate

An entity to whom a DID controller has granted permission to use a verification method associated with a DID via a DID document. For example, a parent who controls a child's DID document might permit the child to use their personal device in order to authenticate. In this case, the child is the DID delegate. The child's personal device would contain the private cryptographic material enabling the child to authenticate using the DID. However, the child might not be permitted to add other personal devices without the parent's permission.

DID document

A set of data that enables cryptographically verifiable interactions with a DID subject. This includes mechanisms, such as cryptographic public keys, that a DID subject or a DID delegate can use to authenticate itself and prove its association with a DID. A DID document might have one or more different representations as defined in 6. Representations or in the W3C *Decentralized Identifier Extensions*.

DID fragment

The portion of a DID URL that follows the first `#` character (U+0023 NUMBER SIGN). DID fragment syntax is identical to URI fragment syntax.

DID method

A definition of how a specific DID method scheme is implemented. A DID method is defined by a DID method specification, which specifies the precise operations by which DIDs and DID documents are created, resolved, updated, and deactivated. See 7. Methods.

DID path

The portion of a DID URL that begins with and includes the first `/` character (U+002F SOLIDUS) and ends with but does not include either a `?` character (U+003F QUESTION MARK), a `#` character (U+0023 NUMBER SIGN), or the end of the DID URL. DID path syntax is identical to URI path syntax. See Path.

DID query
: The portion of a DID URL that follows and includes the first `?` character (U+003F QUESTION MARK). DID query syntax is identical to URI query syntax. See Query.

DID resolution
: The process that takes as its input a DID and a set of resolution options and returns a DID document in a conforming representation plus additional metadata. This process relies on the "Read" operation of the applicable DID method. The inputs and outputs of this process are defined in [*DID-RESOLUTION*].

DID resolver
: A DID resolver is a software and/or hardware component that performs the DID resolution function by taking a DID as input and producing a conforming DID document as output.

DID scheme
: The formal syntax of a decentralized identifier. The generic DID scheme begins with the prefix `did:` as defined in 3.1 DID Syntax. Each DID method specification defines a specific DID method scheme that works with that specific DID method. In a specific DID method scheme, the DID method name follows the first colon and terminates with the second colon, e.g., `did:example:`

DID subject
: The entity identified by a DID and described by a DID document. Anything can be a DID subject: person, group, organization, physical thing, digital thing, logical thing, etc.

DID URL
: A DID plus any additional syntactic component that conforms to the definition in 3.2 DID URL Syntax. This includes an optional DID path with its leading `/` character (U+002F SOLIDUS), optional DID query with its leading `?` character (U+003F QUESTION MARK), and optional DID fragment with its leading `#` character (U+0023 NUMBER SIGN).

DID URL dereferencing
: The process that takes as its input a DID URL and a set of input metadata, and returns a resource. This resource might be a DID document plus additional metadata, a secondary resource contained within the DID document, or a resource entirely external to the DID document. The process uses DID resolution to fetch a DID document indicated by the DID contained within the DID URL. The dereferencing process can then perform additional processing on the DID document to return the dereferenced resource indicated by the DID URL. The inputs and outputs of this process are defined in [*DID-RESOLUTION*].

DID URL dereferencer
: A software and/or hardware system that performs the DID URL dereferencing function for a given DID URL or DID document.

distributed ledger **(DLT)**
: A non-centralized system for recording events. These systems establish sufficient confidence for participants to rely upon the data recorded by others to make operational decisions. They typically use distributed databases where different nodes use a consensus protocol to confirm the ordering of cryptographically signed transactions. The linking of digitally signed transactions over time often makes the history of the ledger effectively immutable.

resource
: A thing that is identified by a URI, as defined in [*RFC3986*]. Similarly, any resource might serve as a DID subject identified by a DID.

representation
: A concrete expression of a resource, as defined by [*RFC9110*]: "information that is intended to reflect a past, current, or desired state of a given resource, in a format that can be readily communicated via the protocol. A representation consists of a set of representation metadata and a potentially unbounded stream of representation data." A DID document is a representation of information which enables cryptographically verifiable interactions with a DID subject. See 6. Representations.

representation-specific entries
: Entries in a DID document whose meaning is particular to a specific representation. Defined in 4. Data Model and 6. Representations.

service
: A means of communicating or interacting with the DID subject or associated entities via one or more service endpoints. Examples include discovery services, agent services, social networking services, file storage services, and verifiable credential repository services.

did service endpoint
: A network address, such as an HTTP URL, at which services operate on behalf of a DID subject.

Uniform Resource Identifier **(URI)**
: The standard identifier format for all resources on the World Wide Web as defined by [*RFC3986*]. A DID is a type of URI scheme.

verifiable data registry
: A system that facilitates the creation, verification, updating, and/or deactivation of decentralized identifiers and DID documents. A verifiable data registry might also be used for other cryptographically-verifiable data

structures such as verifiable credentials. For more information, see the W3C Verifiable Credentials specification [*VC-DATA-MODEL*].

verifiable timestamp
: A verifiable timestamp enables a third-party to verify that a data object existed at a specific moment in time and that it has not been modified or corrupted since that moment in time. If the data integrity could reasonably have been modified or corrupted since that moment in time, the timestamp is not verifiable.

In addition to the terminology above, this specification also uses terminology from the [*INFRA*] specification to formally define the data model. When [*INFRA*] terminology is used, such as string, set, and map, it is linked directly to that specification.

3. Identifier

This section describes the formal syntax for DIDs and DID URLs. The term "generic" is used to differentiate the syntax defined here from syntax defined by *specific* DID methods in their respective specifications. The creation processes, and their timing, for DIDs and DID URLs are described in 7.2 Method Operations and B.2 Creation of a DID.

3.1 DID Syntax

The generic DID scheme is a URI scheme conformant with [*RFC3986*]. The ABNF definition can be found below, which uses the syntax in [*RFC5234*] and the corresponding definitions for `ALPHA` and `DIGIT`. All other rule names not defined in the ABNF below are defined in [*RFC3986*]. All DIDs *MUST* conform to the DID Syntax ABNF Rules.

The DID Syntax ABNF Rules

```
did                = "did:" method-name ":" method-specific-id
method-name        = 1*method-char
method-char        = %x61-7A / DIGIT
method-specific-id = *( *idchar ":" ) 1*idchar
idchar             = ALPHA / DIGIT / "." / "-" / "_" / pct-encoded
pct-encoded        = "%" HEXDIG HEXDIG
```

For requirements on DID methods relating to the DID syntax, see Section 7.1 Method Syntax.

§ 3.2 DID URL Syntax

A DID URL is a network location identifier for a specific resource. It can be used to retrieve things like representations of DID subjects, verification methods, services, specific parts of a DID document, or other resources.

The following is the ABNF definition using the syntax in [*RFC5234*]. It builds on the `did` scheme defined in 3.1 DID Syntax. The `path-abempty`, `query`, and `fragment` components are defined in [*RFC3986*]. All DID URLs *MUST* conform to the DID URL Syntax ABNF Rules. DID methods can further restrict these rules, as described in 7.1 Method Syntax.

The DID URL Syntax ABNF Rules

```
did-url = did path-abempty [ "?" query ] [ "#" fragment ]
```

NOTE: Semicolon character is reserved for future use

Although the semicolon (`;`) character can be used according to the rules of the DID URL syntax, future versions of this specification may use it as a sub-delimiter for parameters as described in [*MATRIX-URIS*]. To avoid future conflicts, developers ought to refrain from using it.

§ Path

A DID path is identical to a generic URI path and conforms to the `path-abempty` ABNF rule in RFC 3986, section 3.3. As with URIs, path semantics can be specified by DID Methods, which in turn might enable DID controllers to further specialize those semantics.

> EXAMPLE 2
>
> ```
> did:example:123456/path
> ```

§ Query

A DID query is identical to a generic URI query and conforms to the `query` ABNF rule in RFC 3986, section 3.4.

> EXAMPLE 3
>
> ```
> did:example:123456?versionId=1
> ```

> NOTE: Avoid comparison of DID URLs containing multiple query parameters
>
> Implementers are urged to avoid comparing DID URLs for equivalence when they have more than one query parameter without a specification specifically designed for that purpose. This specification defines no normalization rules for query parameters, and any query parameter normalization rules defined at the DID Method specification or application layer are not universally recognized rules.

§ Fragment

DID fragment syntax and semantics are identical to a generic URI fragment and conforms to the `fragment` ABNF rule in RFC 3986, section 3.5.

A DID fragment is used as a method-independent reference into a DID document or external resource. Some examples of DID fragment identifiers are shown below.

EXAMPLE 4: A unique verification method in a DID Document

```
did:example:123#public-key-1
```

EXAMPLE 5: A unique service in a DID Document

```
did:example:123#service-5
```

NOTE: Fragment semantics across representations

In order to maximize interoperability, implementers are urged to ensure that DID fragments are interpreted in the same way across representations (see 6. Representations). For example, while JSON Pointer [*RFC6901*] can be used in a DID fragment, it will not be interpreted in the same way across non-JSON representations.

Additional semantics for fragment identifiers, which are compatible with and layered upon the semantics in this section, are specified in the media type description (see Section E.1 application/did). For information about how to dereference a DID fragment, see the *Decentralized Identifier Resolution (DID Resolution) v0.3* specification.

§ 3.2.1 Relative DID URLs

A relative DID URL is any URL value in a DID document that does not start with `did:<method-name>:<method-specific-id>`. More specifically, it is any URL value that does not start with the ABNF defined in 3.1 DID Syntax. The URL is expected to reference a resource in the same DID document. Relative DID URLs *MAY* contain relative path components, query parameters, and fragment identifiers.

When resolving a relative DID URL reference, the algorithm specified in RFC3986 Section 5: Reference Resolution *MUST* be used. The **base URI** value is the DID that is associated with the DID subject, see 5.1.1 DID Subject. The **scheme** is `did`. The **authority** is a combination of `<method-name>:<method-specific-id>`, and the **path**, **query**, and **fragment** values are those defined in Path, Query, and Fragment, respectively.

Relative DID URLs are often used to reference verification methods and services in a DID Document without having to use absolute URLs. DID methods where storage

size is a consideration might use relative URLs to reduce the storage size of DID documents.

EXAMPLE 6: An example of a relative DID URL

```
{
  "@context": "https://www.w3.org/ns/did/v1.1",
  "id": "did:example:123456789abcdefghi",
  "verificationMethod": [{
    "id": "did:example:123456789abcdefghi#key-1",
    "type": "Multikey", // external (property value)
    "controller": "did:example:123456789abcdefghi",
    "publicKeyMultibase": "z6MkmM42vxfqZQsv4ehtTjFFxQ4sQKS2w6WR7emozFAn
  }, ...],
  "authentication": [
    // a relative DID URL used to reference a verification method above
    "#key-1"
  ]
}
```

In the example above, the relative DID URL value will be transformed to an absolute DID URL value of `did:example:123456789abcdefghi#key-1`.

§ 4. Data Model

This specification defines a data model that can be used to express DID documents and their internal data structures, which can then be serialized into a concrete expression. This section provides a high-level description of the data model, descriptions of the ways different types of properties are expressed in the data model, and instructions for extending the data model.

A DID document consists of a map of entries, where each entry consists of a key/value pair. The DID document data model contains properties that are specified in section 5. Core Properties.

All entry keys in the DID document data model are strings. All entry values are expressed using one of the data types in the table below, and each representation specifies the concrete serialization format of each data type.

Data Type	Considerations
map	A finite ordered sequence of key/value pairs, with no key appearing twice as specified in the *Infra Standard*. A map is sometimes referred to as an ordered map in the *Infra Standard*.
list	A finite ordered sequence of items as specified in the *Infra Standard*.
set	A finite ordered sequence of items that does not contain the same item twice as specified in the *Infra Standard*. A set is sometimes referred to as an ordered set in the *Infra Standard*.
datetime	A date and time value that is capable of losslessly expressing all values expressible by a dateTime as specified in [*XMLSCHEMA11-2*].
string	A sequence of code units often used to represent human readable language as specified in the *Infra Standard*.
integer	A real number without a fractional component as specified in [*XMLSCHEMA11-2*]. To maximize interoperability, implementers are urged to heed the advice regarding integers in RFC8259, Section 6: Numbers.
double	A real number with a fractional component that is often used to approximate arbitrary real numbers as specified in [*XMLSCHEMA11-2*]. To maximize interoperability, implementers are urged to heed the advice regarding doubles in RFC8259, Section 6: Numbers.
boolean	A value that is either true or false as defined in the *Infra Standard*.
null	A value that is used to indicate the lack of a value as defined in the *Infra Standard*.

> As a result of the data model being defined using terminology from the *Infra Standard*, property values which can contain more than one item, such as lists, maps, and sets, are explicitly ordered. All list-like value structures in the *Infra Standard* are ordered, whether or not that order is significant. For the purposes of this specification, unless otherwise stated, ordering of a map or set is not important and implementations are not expected to produce or consume deterministically ordered values.

§ 4.1 Extensibility

The data model supports two types of extensibility.

1. For maximum interoperability, it is *RECOMMENDED* that extensions use the repository of *Decentralized Identifier Extensions*. The use of this mechanism for new properties or other extensions is the only specified mechanism that ensures that two different representations will be able to work together.
2. Representations *MAY* define other extensibility mechanisms, including ones that do not require the use of the *Decentralized Identifier Extensions* repository. Such extension mechanisms *SHOULD* support lossless conversion into any other conformant representation. Extension mechanisms for a representation *SHOULD* define a mapping of all properties and representation syntax into the DID document data model and its type system.

> NOTE: Unregistered extensions are less reliable
>
> It is always possible for two specific implementations to agree out-of-band to use a mutually understood extension or representation that is not recorded in the repository of *Decentralized Identifier Extensions*; interoperability between such implementations and the larger ecosystem will be less reliable.

§ 5. Core Properties

A DID is associated with a DID document, which is an extension of a controlled identifier document as defined in *Controlled Identifiers v1.0*. DID documents are expressed using the data model and can be serialized into a representation. The following sections define the properties in a DID document, including whether

these properties are required or optional. These properties describe relationships between the DID subject and the value of the property.

The following tables contain informative references for the core properties defined by this specification, with expected values, and whether or not they are required. The property names in the tables are linked to the normative definitions and more detailed descriptions of each property.

> NOTE: Property names used in maps of different types
>
> The property names `id`, `type`, and `controller` can be present in maps of different types with possible differences in constraints. For example, an `id` at the top-level of a DID document is required to be a DID, while an `id` in a `service` map can be a URL.

Property	Required?	Value constraints	Definition
`id`	yes	A string that conforms to rules defined in Section 3.1 DID Syntax.	Section 5.1.1 DID Subject.
`controller`	no	A string or a set of strings, each of which conforms to rules defined in Section 3.1 DID Syntax.	Section 5.1.2 DID Controller.
`alsoKnownAs`	no	A set of strings, each of which conforms to the URL syntax or Section 3.1 DID Syntax.	Section 2.1.3: Also Known As of the *Controlled Identifiers v1.0* specification.
`service`	no	A set of service maps.	Section 2.1.4: Services of the

Property	Required?	Value constraints	Definition
			Controlled Identifiers v1.0 specification.
`verificationMethod`	no	A set of verification method maps.	Section 5.2 Verification Methods.
`authentication`	no	A set of data where each element is either a string which conforms to rules defined in Section 3.1 DID Syntax, or a map that conforms to the rules for verification methods defined in Section 5.2 Verification Methods.	Section 2.3.1: Authentication of the *Controlled Identifiers v1.0* specification and Section 5.2 Verification Methods.
`assertionMethod`	no	A set of data where each element is either a string which conforms to rules defined in Section 3.1 DID Syntax, or a map that conforms to the rules for verification	Section 2.3.2: Assertion of the *Controlled Identifiers v1.0* specification and Section 5.2 Verification Methods.

Property	Required?	Value constraints	Definition
		methods defined in Section 5.2 Verification Methods.	
`keyAgreement`	no	A set of data where each element is either a string which conforms to rules defined in Section 3.1 DID Syntax, or a map that conforms to the rules for verification methods defined in Section 5.2 Verification Methods.	Section 2.3.3: Key Agreement of the *Controlled Identifiers v1.0* specification and Section 5.2 Verification Methods.
`capabilityInvocation`	no	A set of data where each element is either a string which conforms to rules defined in Section 3.1 DID Syntax, or a map that conforms to the rules for verification methods defined in	Section 2.3.4: Capability Invocation of the *Controlled Identifiers v1.0* specification and Section 5.2 Verification Methods.

Property	Required?	Value constraints	Definition
		Section 5.2 Verification Methods.	
`capabilityDelegation`	no	A set of data where each element is either a string which conforms to rules defined in Section 3.1 DID Syntax, or a map that conforms to the rules for verification methods defined in Section 5.2 Verification Methods.	Section 2.3.5: Capability Delegation of the *Controlled Identifiers v1.0* specification and Section 5.2 Verification Methods.

§ Verification Method properties

Property	Required?	Value constraints	Definition
`id`	yes	A string that conforms to the rules in 3.2 DID URL Syntax.	Section 2.1.1: Subjects of the *Controlled Identifiers v1.0* specification and Section 5.1 Identifiers.

Property	Required?	Value constraints	Definition
`type`	yes	A string.	Section 2.2: Verification Methods of the *Controlled Identifiers v1.0* specification.
`controller`	yes	A string that conforms to the rules in 3.1 DID Syntax.	Section 2.2: Verification Methods of the *Controlled Identifiers v1.0* specification.
`publicKeyMultibase`	no	A string that conforms to a Multibase-encoded public key.	Section 2.2.2: Multikey of the *Controlled Identifiers v1.0* specification.
`publicKeyJwk`	no	A map representing a JSON Web Key.	Section 2.2.3: JsonWebKey of the *Controlled Identifiers v1.0* specification.

§ Service properties

Property	Required?	Value constraints	Definition
`id`	yes	A string that conforms to the rules of either the *URL Standard* standard, or Section 3.1 DID Syntax.	Section 2.1.1: Subjects of the *Controlled Identifiers v1.0* specification.
`type`	yes	A string or a set of strings.	Section 2.1.4: Services of the *Controlled Identifiers v1.0* specification.
`serviceEndpoint`	yes	A single string, a single map, or a set composed of one or more strings and/or maps. Each string value *MUST* be a valid URL conforming to *URL Standard*.	Section 2.1.4: Services of the *Controlled Identifiers v1.0* specification.

§ 5.1 Identifiers

This section describes the mechanisms by which DID documents include identifiers for DID subjects and DID controllers.

§ 5.1.1 DID Subject

The DID for a particular DID subject is expressed using the `id` property in the DID document. This property is defined in Section 2.1.1: Subjects of the *Controlled Identifiers v1.0* specification and extended by this specification to include decentralized identifiers as defined in Section 3.1 DID Syntax.

EXAMPLE 7

```
{
  "id": "did:example:123456789abcdefghijk"
}
```

NOTE: Intermediate representations

DID method specifications can create intermediate representations of a DID document, such as when a decentralized identifier is being registered in a verifiable data registry or when a DID resolver is performing DID resolution. These intermediate representations might not contain `id` values, or might contain interim values for certain `id` properties. Once the DID document is fully resolved, the final `id` value will be determined and put in place of the interim values throughout the DID document.

§ 5.1.2 DID Controller

A DID controller is an entity that is authorized to make changes to a DID document. This property is defined in Section 2.1.2: Controllers of the *Controlled Identifiers v1.0* specification and extended by this specification to include DIDs as defined in Section 3.1 DID Syntax. The process of authorizing a DID controller is defined by the DID method.

EXAMPLE 8: DID document with a controller property

```
{
  "@context": "https://www.w3.org/ns/did/v1.1",
  "id": "did:example:123456789abcdefghi",
  "controller": "did:example:bcehfew7h32f32h7af3"
}
```

5.1.3 Identifier Restrictions

Identifiers used in a DID document to identify a DID subject or a DID Controller cannot use query parameters or fragment identifiers. Implementers are urged to pay particular attention to the list of allowable characters in Section 3.1 DID Syntax which makes this requirement clear; the syntax does not include the `?` character nor the `#` character. This is in contrast to identifiers used in a DID document to identify a verification method or a service, which follow the syntax rules in Section 3.2 DID URL Syntax, which do allow the use of query parameters and fragment identifiers. Even so, the use of query parameters in long-lived canonical identifiers made for DID ecosystems is discouraged as it can increase the complexity of DID resolution software and potentially lead to a larger security attack surface. Fragment identifiers are also expected to be unique within a particular DID document and implementers are discouraged from reusing them to refer to different resources over time, such as two different verification methods within the same DID document.

5.2 Verification Methods

A DID document can express verification methods, as defined in Section 2.2: Verification Methods of *Controlled Identifiers v1.0* with the added restrictions that (a) the `id` value *MUST* conform to Section 3.2 DID URL Syntax or Section 3.2.1 Relative DID URLs and (b) the `controller` value *MUST* conform to Section 3.1 DID Syntax. See Section 2.2: Verification Methods of the *Controlled Identifiers v1.0* specification for a description of verification methods.

§ 5.3 Verification Relationships

A DID document can express verification relationships, as defined in Section 2.3: Verification Relationships of the *Controlled Identifiers v1.0* specification. See Section 2.3: Verification Relationships of the *Controlled Identifiers v1.0* specification for a description of verification methods.

§ 5.4 Services

A DID document can express services, as defined in Section 2.1.4: Services of the *Controlled Identifiers v1.0* specification. Identifiers used in services can be expressed according to Section 3.1 DID Syntax or Section 3.2 DID URL Syntax. See Section 2.1.4: Services of the *Controlled Identifiers v1.0* specification for a description of services.

§ 6. Representations

A concrete serialization of a DID document in this specification is called a representation. A representation is created by serializing the data model through a process called ***production***. A representation is transformed into the data model through a process called ***consumption***. The *production* and *consumption* processes enable the conversion of information from one representation to another. This specification defines a single representation for JSON, which is also compatible with processors that perform JSON-LD processing. The following sections define the general rules for production and consumption, as well as the JSON representation.

§ 6.1 Production and Consumption

In addition to the representations defined in this specification, implementers can use other representations, providing each such representation is properly specified (including rules for interoperable handling of properties not listed in the repository of DID Extensions [*DID-EXTENSIONS*]). See 4.1 Extensibility for more information.

The requirements for all representations are as follows:

1. A representation *MUST* define deterministic production and consumption rules for all data types specified in 4. Data Model.
2. A representation *MUST* be uniquely associated with an IANA-registered Media Type.
3. A representation *MUST* define fragment processing rules for its Media Type that are conformant with the fragment processing rules defined in Fragment.
4. A representation *SHOULD* use the lexical representation of data model data types. For example, JSON and JSON-LD use the XML Schema `dateTime` lexical serialization to represent datetimes. A representation *MAY* choose to serialize the data model data types using a different lexical serializations as long as the consumption process back into the data model is lossless. For example, some CBOR-based representations express datetime values using integers to represent the number of seconds since the Unix epoch.
5. A representation *MAY* define representation-specific entries that are stored in a representation-specific entries map for use during the production and consumption process. These entries are used when consuming or producing to aid in ensuring lossless conversion.
6. In order to maximize interoperability, representation specification authors *SHOULD* register their representation in the repository of DID Extensions [*DID-EXTENSIONS*].

The requirements for all conforming producers are as follows:

1. A conforming producer *MUST* take a DID document data model and a representation-specific entries map as input into the production process. The conforming producer *MAY* accept additional options as input into the production process.
2. A conforming producer *MUST* serialize all entries in the DID document data model, and the representation-specific entries map, that do not have explicit processing rules for the representation being produced using only the representation's data type processing rules and return the serialization after the production process completes.
3. A conforming producer *MUST* return the Media Type string associated with the representation after the production process completes.
4. A conforming producer *MUST NOT* produce non-conforming DIDs or DID documents.

The requirements for all conforming consumers are as follows:

1. A conforming consumer *MUST* take a representation and Media Type string as input into the consumption process. A conforming consumer *MAY* accept additional options as input into the consumption process.
2. A conforming consumer *MUST* determine the representation of a DID document using the Media Type input string.
3. A conforming consumer *MUST* detect any representation-specific entry across all known representations and place the entry into a representation-specific entries map which is returned after the consumption process completes. A list of all known representation-specific entries is available in the repository of DID Extensions [*DID-EXTENSIONS*].
4. A conforming consumer *MUST* add all non-representation-specific entries that do not have explicit processing rules for the representation being consumed to the DID document data model using only the representation's data type processing rules and return the DID document data model after the consumption process completes.
5. A conforming consumer *MUST* produce errors when consuming non-conforming DIDs or DID documents.

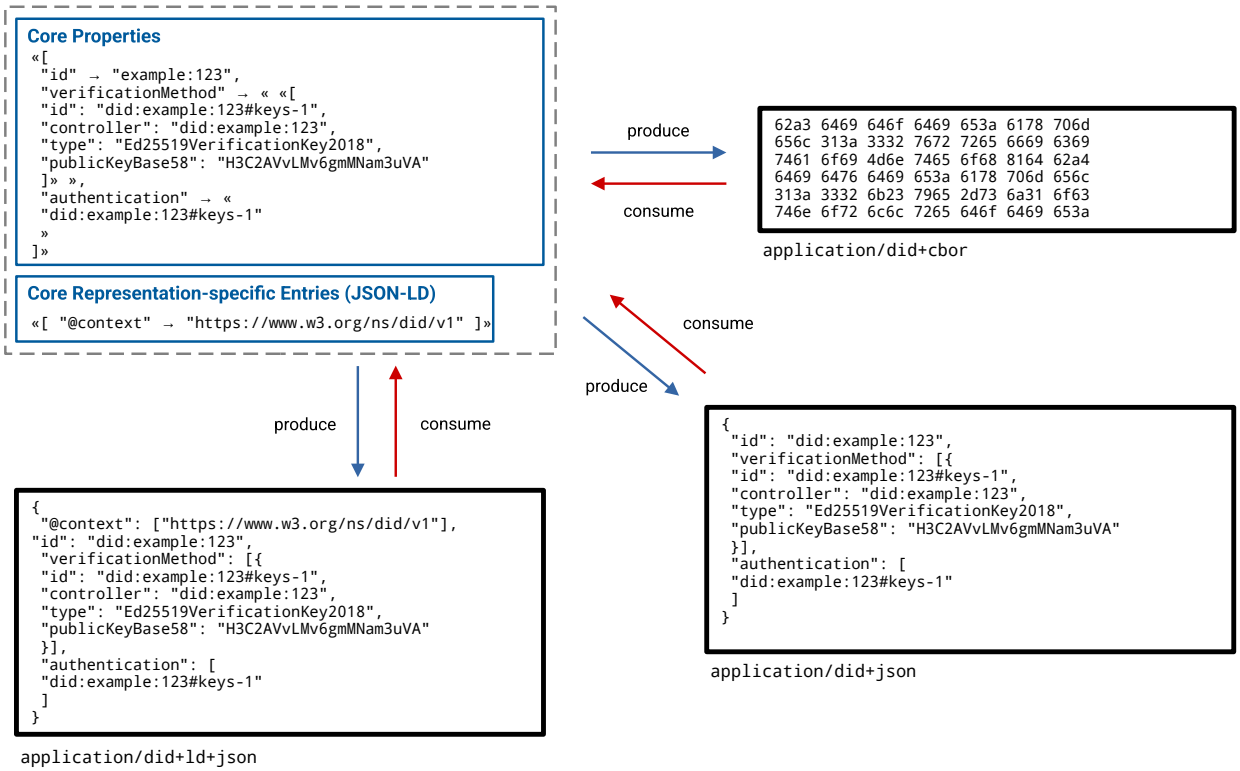

Figure 3 Production and consumption of representations. See also: narrative description.

> NOTE: Conversion between representations
>
> An implementation is expected to convert between representations by using the *consumption* rules on the source representation resulting in the data model and then using the *production* rules to serialize data model to the target representation, or any other mechanism that results in the same target representation.

§ 6.2 JSON

This section defines the production and consumption rules for the JSON representation.

§ 6.2.1 Production

The DID document, DID document data structures, and representation-specific entries map *MUST* be serialized to the JSON representation according to the following production rules:

Data Type	JSON Representation Type
map	A JSON Object, where each entry is serialized as a member of the JSON Object with the entry key as a JSON String member name and the entry value according to its type, as defined in this table.
list	A JSON Array, where each element of the list is serialized, in order, as a value of the array according to its type, as defined in this table.
set	A JSON Array, where each element of the set is added, in order, as a value of the array according to its type, as defined in this table.
datetime	A JSON String serialized as an XML Datetime normalized to UTC 00:00:00 and without sub-second decimal precision. For example: `2020-12-20T19:17:47Z`.

Data Type	JSON Representation Type
string	A JSON String.
integer	A JSON Number without a decimal or fractional component.
double	A JSON Number with a decimal and fractional component.
boolean	A JSON Boolean.
null	A JSON null literal.

> All implementers creating conforming producers that produce JSON representations are advised to ensure that their algorithms are aligned with the JSON serialization rules in the [*INFRA*] specification and the precision advisements regarding Numbers in the JSON [*RFC8259*] specification.

All entries of a DID document *MUST* be included in the root JSON Object. Entries *MAY* contain additional data substructures subject to the value representation rules in the list above. When serializing a DID document, a conforming producer *MUST* specify a media type of `application/did` to downstream applications.

EXAMPLE 9: Example DID document in JSON representation

```
{
  "@context": "https://www.w3.org/ns/did/v1.1",
  "id": "did:example:123456789abcdefghi",
  "authentication": [{
    "id": "did:example:123456789abcdefghi#keys-1",
    "type": "Multikey",
    "controller": "did:example:123456789abcdefghi",
    "publicKeyMultibase": "z6MkmM42vxfqZQsv4ehtTjFFxQ4sQKS2w6WR7emozFAn
  }]
}
```

§ 6.2.2 Consumption

The DID document and DID document data structures JSON representation *MUST* be deserialized into the data model according to the following consumption rules:

JSON Representation Type	Data Type
JSON Object	A map, where each member of the JSON Object is added as an entry to the map. Each entry key is set as the JSON Object member name. Each entry value is set by converting the JSON Object member value according to the JSON representation type as defined in this table. Since order is not specified by JSON Objects, no insertion order is guaranteed.
JSON Array where the data model entry value is a list or unknown	A list, where each value of the JSON Array is added to the list in order, converted based on the JSON representation type of the array value, as defined in this table.
JSON Array where the data model entry value is a set	A set, where each value of the JSON Array is added to the set in order, converted based on the JSON representation type of the array value, as defined in this table.
JSON String where data model entry value is a datetime	A datetime.
JSON String, where the data model entry value type is string or unknown	A string.
JSON Number without a decimal or fractional component	An integer.
JSON Number with a decimal and fractional component, or when entry value is a double regardless of inclusion of fractional component	A double.

JSON Representation Type	Data Type
JSON Boolean	A boolean.
JSON null literal	A null value.

> All implementers creating conforming producers or conforming consumers are advised to ensure that their algorithms are aligned with the JSON conversion rules in the [*INFRA*] specification and the precision advisements regarding Numbers in the JSON [*RFC8259*] specification.

If media type information is available to a conforming consumer and the media type value is `application/did`, then the data structure being consumed is a DID document, and the root element *MUST* be a JSON Object where all members of the object are entries of the DID document. A conforming consumer for a JSON representation that is consuming a DID document with a root element that is not a JSON Object *MUST* report an error.

§ 6.2.3 JSON-LD Processors

JSON-LD is a JSON-based format used to serialize Linked Data. Some implementations are expected to process DID documents using the standard JSON-LD Processing algorithms. To maximize interoperability, implementers are strongly advised to ensure that JSON-LD Processing using standards-compliant libraries is not prevented. The semantics of a DID document are the same whether JSON-LD Processing using standards-compliant libraries is performed or not. Any difference in semantics is an error in either implementation or DID Method specification.

For JSON-LD processing to occur, a `@context` property *MUST* be specified according to the rules specified in the *JSON-LD 1.1* specification.

@context
: The JSON-LD Context is either a string or a list containing any combination of strings and/or ordered maps.

The DID document, DID document data structures, and representation-specific entries map *MUST* be serialized to the JSON representation according to the JSON representation production rules as defined in 6.2 JSON.

In addition to using the JSON representation production rules, production *MUST* include the `@context` entry. The serialized value of `@context` *MUST* be the JSON String `https://www.w3.org/ns/did/v1.1`, or a JSON Array where the first item is the JSON String `https://www.w3.org/ns/did/v1.1` and the subsequent items are serialized according to the JSON production rules.

EXAMPLE 10: A valid serialization of a simple @context entry

```
{
  "@context": "https://www.w3.org/ns/did/v1.1",
  ...
}
```

EXAMPLE 11: A valid serialization of a layered @context entry

```
{
  "@context": [
    "https://www.w3.org/ns/did/v1.1",
    "https://did-method-extension.example/v1"
  ],
  ...
}
```

All implementers creating conforming consumers that produce or consume representations meant to be processed as JSON-LD are advised to ensure that their algorithms produce valid *JSON-LD* documents. Invalid JSON-LD documents will cause JSON-LD processors to halt and report errors.

To achieve interoperability across different representations, all JSON-LD Contexts and their terms *SHOULD* be registered in the repository of *Decentralized Identifier Extensions*.

A conforming producer that generates a JSON-LD representation *SHOULD NOT* produce a DID document that contains terms not defined via the `@context` as conforming consumers are expected to remove unknown terms. When serializing a JSON-LD representation of a DID document, a conforming producer *MUST* specify a media type of `application/did` to downstream applications.

Conforming consumers that process a JSON-LD representation *SHOULD* drop all terms from a DID document that are not defined via the `@context`.

§ 6.2.4 Remote Resource Integrity

Implementations *MUST* treat the base context value, located at `https://www.w3.org/ns/did/v1.1`, as already retrieved; the following value is the hexadecimal encoded SHA2-256 digest value of the base context file: `ea216ecc1cb02cd39b693dba2250141e270ba0bf95890be107dd9a9e8e43de85`. It is possible to confirm the cryptographic digest above by running the following command from a modern Unix command line interface: `curl -s https://www.w3.org/ns/did/v1.1 | openssl dgst -sha256`.

Implementers are warned that other data that is referenced from within a DID document, such as resources that are linked to via URLs, are not cryptographically protected by default. It is considered a best practice to ensure that the same sorts of protections are provided for any URL that is critical to the security of the DID document through the use of permanently cached files and/or cryptographic hashes. Ultimately, knowing the cryptographic digest of any linked external content enables an application to confirm that the content is the same as what the DID controller intended.

> NOTE: See errata if hash value changes are detected
>
> It is extremely unlikely that the files that have associated cryptographic hash values in this specification will change. However, if critical errata are found in the specification and corrections are required to ensure ecosystem stability, the cryptographic hash values might change. As such, the HTTP cache times for the files are not set to infinity and implementers are advised to check for errata if a change is detected in a cryptographic hash value.

§ 6.3 Media Types

Media types, as defined in [*RFC6838*], identify the syntax used to express a DID document as well as other useful processing guidelines.

Syntaxes used to express the data model in this specification *SHOULD* be identified by a media type, and conventions outlined in this section *SHOULD* be followed when defining or using media types with DID documents.

There is one media type associated with the core data model, which is listed in Section E. IANA Considerations: `application/did`.

§ 6.3.1 Media Type Precision

This section is non-normative.

At times, developers or systems might use lower-precision media types to convey DID documents. Some of the reasons for use of lower-precision media types include:

- A web server defaults to `text/plain` or `application/octet-stream` when a file extension is not available and it cannot determine the media type.
- A developer adds a file extension that leads to a media type that is less specific than the content of the file. For example, `.json` could result in a media type of `application/json` and `.jsonld` might result in a media type of `application/ld+json`.
- A protocol requires a less precise media type for a particular transaction; for example, `application/json` instead of `application/did`,

Implementers are discouraged from raising errors when it is possible to determine the intended media type from the payload, provided that the media type used is acceptable in the given protocol. For example, if an application only accepts payloads that conform to the rules associated with the `application/did` media type, but the payload is tagged with the lower-precision `application/json` or `application/ld+json`, the application might perform the following steps to determine whether the payload also conforms to the higher-precision media type:

1. Parse the payload as a JSON document.
2. Ensure that the first element of the `@context` property matches `https://www.w3.org/ns/did/v1.1`.
3. Assume an `application/did` media type if the JSON document contains a top-level `id` property containing an identifier that conforms to the rules in Section 3.1 DID Syntax.

Whenever possible, implementers are advised to use the most precise (the highest-precision) media type for all payloads defined by this specification. Implementers are also advised to recognize that a payload tagged with a lower- precision media type does not mean that the payload does not meet the rules necessary to tag it with a higher-precision type. Similarly, a payload tagged with a higher-precision media type does not mean that the payload will meet the requirements associated with the media type. Receivers of payloads, regardless of their associated media

type, are expected to perform appropriate checks to ensure that payloads conform with the requirements for their use in a given system.

HTTP clients and servers use media types associated with DID documents in `Accept:` headers and when indicating content types. Implementers are warned that HTTP servers might ignore the `Accept:` header and return another content type, or return an error code such as `415 Unsupported Media Type`.

7. Methods

A conforming DID method defines how implementers can realize the features described by this specification. DID methods are often associated with a particular verifiable data registry. New DID methods are defined in their own specifications to enable interoperability between different implementations of the same DID method.

Conceptually, the relationship between this specification and a DID method specification is similar to the relationship between the IETF generic URI specification [*RFC3986*] and a specific URI scheme [*IANA-URI-SCHEMES*], such as the `http` scheme [*RFC9110*]. In addition to defining a specific DID scheme, a DID method specification also defines the mechanisms for creating, resolving, updating, and deactivating DIDs and DID documents using a specific type of verifiable data registry. It also documents all implementation considerations related to DIDs as well as Security and Privacy Considerations.

This section specifies the requirements for authoring DID method specifications.

7.1 Method Syntax

The requirements for all DID method specifications when defining the method-specific DID Syntax are as follows:

1. A DID method specification *MUST* define exactly one method-specific DID scheme that is identified by exactly one method name as specified by the `method-name` rule in 3.1 DID Syntax.
2. The DID method specification *MUST* specify how to generate the `method-specific-id` component of a DID.
3. The DID method specification *MUST* define sensitivity and normalization of the value of the `method-specific-id`.

4. The `method-specific-id` value *MUST* be unique within a DID method. The `method-specific-id` value itself might be globally unique.
5. Any DID generated by a DID method *MUST* be globally unique.
6. To reduce the chances of `method-name` conflicts, a DID method specification *SHOULD* be registered in the repository of DID Extensions [*DID-EXTENSIONS*].
7. A DID method *MAY* define multiple `method-specific-id` formats.
8. The `method-specific-id` format *MAY* include colons. The use of colons *MUST* comply syntactically with the `method-specific-id` ABNF rule.
9. A DID method specification *MAY* specify ABNF rules for DID paths that are more restrictive than the generic rules in Path.
10. A DID method specification *MAY* specify ABNF rules for DID queries that are more restrictive than the generic rules in this section.
11. A DID method specification *MAY* specify ABNF rules for DID fragments that are more restrictive than the generic rules in this section.

> NOTE: Colons in method-specific-id
>
> The meaning of colons in the `method-specific-id` is entirely method-specific. Colons might be used by DID methods for establishing hierarchically partitioned namespaces, for identifying specific instances or parts of the verifiable data registry, or for other purposes. Implementers are advised to avoid assuming any meanings or behaviors associated with a colon that are generically applicable to all DID methods.

§ 7.2 Method Operations

The requirements for all DID method specifications when defining the method operations are as follows:

1. A DID method specification *MUST* define how authorization is performed to execute all operations, including any necessary cryptographic processes.
2. A DID method specification *MUST* specify how a DID controller creates a DID and its associated DID document.
3. A DID method specification *MUST* specify how a DID resolver uses a DID to resolve a DID document, including how the DID resolver can verify the authenticity of the response.

4. A DID method specification *MUST* specify what constitutes an update to a DID document and how a DID controller can update a DID document *or* state that updates are not possible.
5. The DID method specification *MUST* specify how a DID controller can deactivate a DID *or* state that deactivation is not possible.

The authority of a party that is performing authorization to carry out the operations is specific to a DID method. For example, a DID method might —

- make use of the `controller` property.
- use the verification methods listed under `authentication`.
- use other constructs in the DID Document such as the verification method specified via the `capabilityInvocation` verification relationship.
- not use the DID document for this decision at all, and depend on an out-of-band mechanism, instead.

When executing method operations, a DID method can use any data structures it requires, including intermediate, partial, or temporary DID documents, as long as they are kept internal to the DID method, and the DID documents returned by the method operations are fully conformant, as defined in 1.4 Conformance.

§ 7.3 Security Requirements

The requirements for all DID method specifications when authoring the *Security Considerations* section are as follows:

1. A DID method specifications *MUST* follow all guidelines and normative language provided in RFC3552: Writing Security Considerations Sections for the DID operations defined in the DID method specification.
2. The Security Considerations section *MUST* document the following forms of attack for the DID operations defined in the DID method specification: eavesdropping, replay, message insertion, deletion, modification, denial of service, amplification, and man-in-the-middle. Other known forms of attack *SHOULD* also be documented.
3. The Security Considerations section *MUST* discuss residual risks, such as the risks from compromise in a related protocol, incorrect implementation, or cipher after threat mitigation was deployed.
4. The Security Considerations section *MUST* provide integrity protection and update authentication for all operations required by Section 7.2 Method

Operations.

5. If authentication is involved, particularly user-host authentication, the security characteristics of the authentication method *MUST* be clearly documented.
6. The Security Considerations section *MUST* discuss the policy mechanism by which DIDs are proven to be uniquely assigned.
7. Method-specific endpoint authentication *MUST* be discussed. Where DID methods make use of DLTs with varying network topology, sometimes offered as *light node* or *thin client* implementations to reduce required computing resources, the security assumptions of the topology available to implementations of the DID method *MUST* be discussed.
8. If a protocol incorporates cryptographic protection mechanisms, the DID method specification *MUST* clearly indicate which portions of the data are protected and by what protections, and it *SHOULD* give an indication of the sorts of attacks to which the cryptographic protection is susceptible. Some examples are integrity only, confidentiality, and endpoint authentication.
9. Data which is to be held secret (keying material, random seeds, and so on) *SHOULD* be clearly labeled.
10. DID method specifications *SHOULD* explain and specify the implementation of signatures on DID documents, if applicable.
11. Where DID methods use peer-to-peer computing resources, such as with all known DLTs, the expected burdens of those resources *SHOULD* be discussed in relation to denial of service.
12. DID methods that introduce new authentication service types, as described in 5.4 Services, *SHOULD* consider the security requirements of the supported authentication protocol.

§ 7.4 Privacy Requirements

The requirements for all DID method specifications when authoring the *Privacy Considerations* section are:

1. The DID method specification's Privacy Considerations section *MUST* discuss any subsection of Section 5 of [*RFC6973*] that could apply in a method-specific manner. The subsections to consider are: surveillance, stored data compromise, unsolicited traffic, misattribution, correlation, identification, secondary use, disclosure, and exclusion.

8. Security Considerations

This section is non-normative.

This section contains a variety of security considerations that people using Decentralized Identifiers are advised to consider before deploying this technology in a production setting. Readers are urged to read the Security Considerations section of the *Controlled Identifiers v1.0* specification before reading this section. DIDs are designed to operate under the threat model used by many IETF standards and documented in [*RFC3552*]. This section elaborates upon a number of the considerations in [*RFC3552*], as well as other considerations that are unique to DID architecture.

8.1 Choosing DID Resolvers

The repository of DID Extensions [*DID-EXTENSIONS*] contains an informative list of DID method names and their corresponding DID method specifications. Implementers need to bear in mind that there is no central authority to mandate which DID method specification is to be used with any specific DID method name. If there is doubt on whether or not a specific DID resolver implements a DID method correctly, the DID Specification Registries can be used to look up the registered specification and make an informed decision regarding which DID resolver implementation to use.

8.2 Non-Repudiation

Non-repudiation of DIDs and DID document updates is supported if:

- The verifiable data registry supports verifiable timestamps. See "DID Document Metadata" in [*DID-RESOLUTION*] for further information on useful timestamps that can be used during the DID resolution process.
- The subject has had adequate opportunity to revert malicious updates according to the authorization mechanism for the DID method.

§ 8.3 Revocation in Trustless Systems

Trustless systems are those where all trust is derived from cryptographically provable assertions, and more specifically, where no metadata outside of the cryptographic system is factored into the determination of trust in the system. To verify a signature of proof for a verification method which has been revoked in a trustless system, a DID method needs to support either or both of the `versionId` or `versionTime`, as well as both the `updated` and `nextUpdate`, DID document metadata properties. A verifier can validate a signature or proof of a revoked key if and only if all of the following are true:

- The proof or signature includes the `versionId` or `versionTime` of the DID document that was used at the point in time at which the signature or proof was made.
- The verifier can determine the point in time at which the signature or proof was made; for example, it was anchored on a blockchain.
- In the resolved DID document metadata, the `updated` timestamp is before, and the `nextUpdate` timestamp is after, the point in time at which the signature or proof was made.

Similar trust may be achieved in systems that are willing to accept metadata beyond that which constitutes cryptographic input -- but this always requires a careful judgment about whether a DID document's content included the expected content at the moment of a signing event.

§ 8.4 DID Recovery

Recovery is a reactive security measure whereby a controller that has lost the ability to perform DID operations, such as through the loss of a device, is able to regain the ability to perform DID operations.

The following considerations might be of use when contemplating the use of DID recovery:

- Proactively performing recovery, on an infrequent but regular basis, can help to prevent loss of control.

- It is considered a best practice to never reuse cryptographic material associated with recovery for any other purposes.
- Recovery is commonly performed in conjunction with verification method rotation and verification method revocation.
- Recovery is advised when a controller or any service trusted to act on their behalf no longer has the exclusive ability to perform DID operations as described in 7.2 Method Operations.
- DID method specifications might choose to enable support for a quorum of trusted parties to facilitate recovery. Some of the facilities to do so are suggested in 5.1.2 DID Controller.
- Not all DID method specifications will recognize control from DIDs registered using other DID methods and they might restrict third-party control to DIDs that use the same method.
- Access control and recovery in a DID method specification can also include a time lock feature to protect against key compromise by maintaining a second track of control for recovery.
- There are currently no common recovery mechanisms that apply to all DID methods.

§ 8.5 The Role of Human-Friendly Identifiers

DIDs achieve global uniqueness without the need for a central registration authority. This comes at the cost of human memorability. Algorithms capable of generating globally unambiguous identifiers produce random strings of characters that have no human meaning. This trade-off is often referred to as Zooko's Triangle.

There are use cases where it is desirable to discover a DID when starting from a human-friendly identifier. For example, a natural language name, a domain name, or a conventional address for a DID controller, such as a mobile telephone number, email address, social media username, or blog URL. However, the problem of mapping human-friendly identifiers to DIDs, and doing so in a way that can be verified and trusted, is outside the scope of this specification.

Solutions to this problem are defined in separate specifications, such as [*DNS-DID*], that reference this specification. It is strongly recommended that such specifications carefully consider the:

- Numerous security attacks based on deceiving users about the true human-friendly identifier for a target entity.
- Privacy consequences of using human-friendly identifiers that are inherently correlatable, especially if they are globally unique.

§ 8.6 DIDs as Enhanced URNs

If desired by a DID controller, a DID or a DID URL is capable of acting as persistent, location-independent resource identifier. These sorts of identifiers are classified as Uniform Resource Names (URNs) and are defined in [*RFC8141*]. DIDs are an enhanced form of URN that provide a cryptographically secure, location-independent identifier for a digital resource, while also providing metadata that enables retrieval. Due to the indirection between the DID document and the DID itself, the DID controller can adjust the actual location of the resource — or even provide the resource directly — without adjusting the DID. DIDs of this type can definitively verify that the resource retrieved is, in fact, the resource identified.

A DID controller who intends to use a DID for this purpose is advised to follow the security considerations in [*RFC8141*]. In particular:

- The DID controller is expected to choose a DID method that supports the controller's requirements for persistence. The Decentralized Characteristics Rubric [*DID-RUBRIC*] is one tool available to help implementers decide upon the most suitable DID method.
- The DID controller is expected to publish its operational policies so requesting parties can determine the degree to which they can rely on the persistence of a DID controlled by that DID controller. In the absence of such policies, requesting parties are not expected to make any assumption about whether a DID is a persistent identifier for the same DID subject.

§ 8.7 Immutability

Many cybersecurity abuses hinge on exploiting gaps between reality and the assumptions of rational, good-faith actors. Immutability of DID documents can provide some security benefits. Individual DID methods ought to consider constraints that would eliminate behaviors or semantics they do not need. The more *locked down* a DID method is, while providing the same set of features, the less it can be manipulated by malicious actors.

As an example, consider that a single edit to a DID document can change anything except the root `id` property of the document. But is it actually desirable for a service to change its `type` after it is defined? Or for a key to change its value? Or would it be better to require a new `id` when certain fundamental properties of an object change? Malicious takeovers of a website often aim for an outcome where the site keeps its host name identifier, but is subtly changed underneath. If certain properties of the site, such as the ASN associated with its IP address, were required by the specification to be immutable, anomaly detection would be easier, and attacks would be much harder and more expensive to carry out.

For DID methods tied to a global source of truth, a direct, just-in-time lookup of the latest version of a DID document is always possible. However, it seems likely that layers of cache might eventually sit between a DID resolver and that source of truth. If they do, believing the attributes of an object in the DID document to have a given state when they are actually subtly different might invite exploits. This is particularly true if some lookups are of a full DID document, and others are of partial data where the larger context is assumed.

§ 8.8 Encrypted Data in DID Documents

Encryption algorithms have been known to fail due to advances in cryptography and computing power. Implementers are advised to assume that any encrypted data placed in a DID document might eventually be made available in clear text to the same audience to which the encrypted data is available. This is particularly pertinent if the DID document is public.

Encrypting all or parts of a DID document is *not* an appropriate means to protect data in the long term. Similarly, placing encrypted data in a DID document is not an appropriate means to protect personal data.

Given the caveats above, if encrypted data is included in a DID document, implementers are advised to not associate any correlatable information that could be used to infer a relationship between the encrypted data and an associated party. Examples of correlatable information include public keys of a receiving party, identifiers to digital assets known to be under the control of a receiving party, or human readable descriptions of a receiving party.

§ 8.9 Equivalence Properties

Given the `equivalentId` and `canonicalId` properties are generated by DID methods themselves, the same security and accuracy guarantees that apply to the resolved DID present in the `id` field of a DID document also apply to these properties. The `alsoKnownAs` property is not guaranteed to be an accurate statement of equivalence, and should not be relied upon without performing validation steps beyond the resolution of the DID document.

The `equivalentId` and `canonicalId` properties express equivalence assertions to variants of a single DID produced by the same DID method and can be trusted to the extent the requesting party trusts the DID method and a conforming producer and resolver.

The `alsoKnownAs` property permits an equivalence assertion to URIs that are not governed by the same DID method and cannot be trusted without performing verification steps outside of the governing DID method. See additional guidance in Section 2.1.3: Also Known As of the *Controlled Identifiers v1.0* specification.

As with any other security-related properties in the DID document, parties relying on any equivalence statement in a DID document should guard against the values of these properties being substituted by an attacker after the proper verification has been performed. Any write access to a DID document stored in memory or disk after verification has been performed is an attack vector that might circumvent verification unless the DID document is re-verified.

§ 8.10 Persistence

DIDs are designed to be persistent such that a controller need not rely upon a single trusted third party or administrator to maintain their identifiers. In an ideal case, no administrator can take control away from the controller, nor can an administrator prevent their identifiers' use for any particular purpose such as authentication, authorization, and attestation. No third party can act on behalf of a controller to remove or render inoperable an entity's identifier without the controller's consent.

However, it is important to note that in all DID methods that enable cryptographic proof-of-control, the means of proving control can always be transferred to another

party by transferring the secret cryptographic material. Therefore, it is vital that systems relying on the persistence of an identifier over time regularly check to ensure that the identifier is, in fact, still under the control of the intended party.

Unfortunately, it is impossible to determine from the cryptography alone whether or not the secret cryptographic material associated with a given verification method has been compromised. It might well be that the expected controller still has access to the secret cryptographic material — and as such can execute a proof-of-control as part of a verification process — while at the same time, a bad actor also has access to those same keys, or to a copy thereof.

As such, cryptographic proof-of-control is expected to only be used as one factor in evaluating the level of identity assurance required for high-stakes scenarios. DID-based authentication provides much greater assurance than a username and password, thanks to the ability to determine control over a cryptographic secret without transmitting that secret between systems. However, it is not infallible. Scenarios that involve sensitive, high value, or life-critical operations are expected to use additional factors as appropriate.

In addition to potential ambiguity from use by different controllers, it is impossible to guarantee, in general, that a given DID is being used in reference to the same subject at any given point in time. It is technically possible for the controller to reuse a DID for different subjects and, more subtly, for the precise definition of the subject to either change over time or be misunderstood.

For example, consider a DID used for a sole proprietorship, receiving various credentials used for financial transactions. To the controller, that identifier referred to the business. As the business grows, it eventually gets incorporated as a Limited Liability Company. The controller continues using that same DID, because to **them** the DID refers to the business. However, to the state, the tax authority, and the local municipality, the DID no longer refers to the same entity. Whether or not the subtle shift in meaning matters to a credit provider or supplier is necessarily up to them to decide. In many cases, as long as the bills get paid and collections can be enforced, the shift is immaterial.

Due to these potential ambiguities, DIDs are to be considered valid *contextually* rather than absolutely. Their persistence does not imply that they refer to the exact same subject, nor that they are under the control of the same controller. Instead, one needs to understand the context in which the DID was created, how it is used, and consider the likely shifts in their meaning, and adopt procedures and policies to address both potential and inevitable semantic drift.

§ 8.11 Evaluating Competing Considerations

This section is non-normative.

This specification does not require or suggest the use of any specific type of verifiable data registry. Different use cases might result in different requirements. Different requirements might suggest different considerations with different trade-offs. For example, trade-offs between computation (energy usage), trust (deference to authority), coordination (network bandwidth), or memory (physical storage) might or might not be appropriate for any given use case. Other use cases might not make the same trade-offs. Those that need to consider different criteria for their use case are directed to the DID Method Rubric, which provides evaluation criteria to help decision makers determine whether or not a particular DID Method is appropriate for their use cases.

§ 9. Privacy Considerations

This section is non-normative.

Since DIDs and DID documents are designed to be administered directly by the DID controller(s), it is critically important to apply the principles of Privacy by Design [*PRIVACY-BY-DESIGN*] to all aspects of the decentralized identifier architecture. All seven of these principles have been applied throughout the development of this specification. The design used in this specification does not assume that there is a registrar, hosting company, nor other intermediate service provider to recommend or apply additional privacy safeguards. Privacy in this specification is preventive, not remedial, and is an embedded default. Before reading this section, readers are urged to read the Privacy Considerations section of the *Controlled Identifiers v1.0* specification, as it contains more general privacy considerations that also apply to DIDs. The rest of this section covers privacy considerations that are specific to decentralized identifiers and are in addition to the guidance provided in the *Controlled Identifiers v1.0* specification.

§ 9.1 Group Privacy

When a DID subject is indistinguishable from others in the group, privacy is available. When the act of engaging privately with another party is by itself a recognizable flag, privacy is greatly diminished.

DIDs and DID methods need to work to improve group privacy, particularly for those who legitimately need it most. Choose technologies and human interfaces that default to preserving anonymity and pseudonymity. To reduce digital fingerprints, share common settings across requesting party implementations, keep negotiated options to a minimum on wire protocols, use encrypted transport layers, and pad messages to standard lengths.

A. Examples

This section is non-normative.

A.1 DID Documents

This section is non-normative.

See Verification Method Types [*DID-EXTENSIONS*] for optional extensions and other verification method types.

> NOTE
>
> These examples are for information purposes only, it is considered a best practice to avoid using the same verification method for multiple purposes.

EXAMPLE 12: DID Document with 1 verification method type

```
  {
    "@context": "https://www.w3.org/ns/did/v1.1",
    "id": "did:example:123",
    "authentication": [
      {
        "id": "did:example:123#z6MkmM42vxfqZQsv4ehtTjFFxQ4sQKS2w6WR7em
        "type": "Multikey", // external (property value)
        "controller": "did:example:123",
        "publicKeyMultibase": "z6MkmM42vxfqZQsv4ehtTjFFxQ4sQKS2w6WR7em
      }
    ],
    "capabilityInvocation": [
      {
        "id": "did:example:123#z6Mkvtac9bidSz9bBttzn7Yg3oCDHvMY2FtkFLs
        "type": "Multikey", // external (property value)
        "controller": "did:example:123",
        "publicKeyMultibase": "z6Mkvtac9bidSz9bBttzn7Yg3oCDHvMY2FtkFLs
      }
    ],
    "capabilityDelegation": [
      {
        "id": "did:example:123#z6MknxsdF4CGVxhRNsx6TvXPFczaHEkajKBBwu7
        "type": "Multikey", // external (property value)
        "controller": "did:example:123",
        "publicKeyMultibase": "z6MknxsdF4CGVxhRNsx6TvXPFczaHEkajKBBwu7
      }
    ],
    "assertionMethod": [
      {
        "id": "did:example:123#z6MkgYhVuWq4hyc7ZKBGhsY7pb5Bc8V6VPXGPG3
        "type": "Multikey", // external (property value)
        "controller": "did:example:123",
        "publicKeyMultibase": "z6MkgYhVuWq4hyc7ZKBGhsY7pb5Bc8V6VPXGPG3
      }
    ]
}
```

EXAMPLE 13: DID Document with many different key types

```
{
  "@context": "https://www.w3.org/ns/did/v1.1",
  "id": "did:example:123",
  "verificationMethod": [
    {
      "id": "did:example:123#key-0",
      "type": "JsonWebKey",
      "controller": "did:example:123",
      "publicKeyJwk": {
        "kty": "OKP", // external (property name)
        "crv": "Ed25519", // external (property name)
        "x": "VCpo2LMLhn6iWku8MKvSLg2ZAoC-nlOyPVQaO3FxVeQ" // external
      }
    },
    {
      "id": "did:example:123#key-1",
      "type": "JsonWebKey",
      "controller": "did:example:123",
      "publicKeyJwk": {
        "kty": "OKP", // external (property name)
        "crv": "X25519", // external (property name)
        "x": "pE_mG098rdQjY3MKK2D5SUQ6ZOEW3a6Z6T7Z4SgnzCE" // external
      }
    },
    {
      "id": "did:example:123#key-2",
      "type": "JsonWebKey",
      "controller": "did:example:123",
      "publicKeyJwk": {
        "kty": "EC", // external (property name)
        "crv": "secp256k1", // external (property name)
        "x": "Z4Y3NNOxv0J6tCgqOBFnHnaZhJF6LdulT7z8A-2D5_8", // externa
        "y": "i5a2NtJoUKXkLm6q8nOEu9WOkso1Ag6FTUT6k_LMnGk" // external
      }
    },
    {
      "id": "did:example:123#key-3",
      "type": "JsonWebKey",
      "controller": "did:example:123",
      "publicKeyJwk": {
        "kty": "EC", // external (property name)
        "crv": "secp256k1", // external (property name)
        "x": "U1V4TVZVMUpUaOZVU1NBcU9CRm5IbmFaaEpGNkxkdWx", // externa
        "y": "i5a2NtJoUKXkLm6q8nOEu9WOkso1Ag6FTUT6k_LMnGk" // external
```

```
    }
  },
  {
    "id": "did:example:123#key-4",
    "type": "JsonWebKey",
    "controller": "did:example:123",
    "publicKeyJwk": {
      "kty": "EC", // external (property name)
      "crv": "P-256", // external (property name)
      "x": "Ums5WVgwRkRTVVFnU3k5c2xvZllMbEcwM3NPRW91ZzN", // externa
      "y": "nDQW6XZ7b_u2Sy9slofYLlG03sOEoug3I0aAPQ0exs4" // external
    }
  },
  {
    "id": "did:example:123#key-5",
    "type": "JsonWebKey",
    "controller": "did:example:123",
    "publicKeyJwk": {
      "kty": "EC", // external (property name)
      "crv": "P-384", // external (property name)
      "x": "VUZKSlUwMGdpSXplekRwODhzX2N4U1BYdHVYWUZsaXVDR25kZ1U0UXA4
      "y": "jq4QoAHKiIzezDp88s_cxSPXtuXYFliuCGndgU4Qp8l91xzD1spCmFIz
    }
  },
  {
    "id": "did:example:123#key-6",
    "type": "JsonWebKey",
    "controller": "did:example:123",
    "publicKeyJwk": {
      "kty": "EC", // external (property name)
      "crv": "P-521", // external (property name)
      "x": "VTI5c1lYSmZWMmx1WkhNZ0dQTXhaYkhtSnBEU3UtSXZwdUtpZ0VOMnB6
      "y": "UW5WNVgwSnBkR052YVc0Z1VqY1B6LVpoZWNaRnliT3FMSUpqVk9sTEVU
    }
  },
  {
    "id": "did:example:123#key-7",
    "type": "JsonWebKey",
    "controller": "did:example:123",
    "publicKeyJwk": {
      "kty": "RSA", // external (property name)
      "e": "AQAB", // external (property name)
      "n": "UkhWaGJGOUZRMTlFVWtKSElBdENGV2hlU1F2djFNRXh1NVJMQ01UNGpW
    }
```

```
    }
  ]
}
```

EXAMPLE 14: DID Document with different verification method types

```
{
  "@context": "https://www.w3.org/ns/did/v1.1",
  "id": "did:example:123",
  "verificationMethod": [{
    "id": "did:example:123#key-0",
    "type": "Multikey",
    "controller": "did:example:123",
    "publicKeyMultibase": "z6MkmM42vxfqZQsv4ehtTjFFxQ4sQKS2w6WR7emozFA
  }, {
    "id": "did:example:123#key-1",
    "type": "Multikey",
    "controller": "did:example:123",
    "publicKeyMultibase": "z6MtTjFFxQ4sQKS2wmozFAn5cxukmM46WR7e2vxfqZQ
  }, {
    "id": "did:example:123#key-2",
    "type": "EcdsaSecp256k1VerificationKey2019",
    "controller": "did:example:123",
    "publicKeyMultibase": "zns2aFDq25fEV1NUd3wZ65sgtht4j5QjFW8JCAHdUJf
  }, {
    "id": "did:example:123#key-3",
    "type": "JsonWebKey",
    "controller": "did:example:123",
    "publicKeyJwk": {
      "kty": "EC", // external (property name)
      "crv": "P-256", // external (property name)
      "x": "Er6KSSnAjI7OObRWhlaMgqyIOQYrDJTE94ej5hybQ2M",
      "y": "pPVzCOTJwgikPjuUE6UebfZySqEJ0ZtsWFpj7YSPGEk"
    }
  }]
}
```

EXAMPLE 15: DID Document that uses relative DID URLs

```
{
  "@context": "https://www.w3.org/ns/did/v1.1",
  "id": "did:example:123",
  "verificationMethod": [
    {
      // A relative DID URL, that will be transformed to the absolut
      "id": "#key-1",
      "type": "Ed25519VerificationKey2020",
      "controller": "did:example:123",
      "publicKeyMultibase": "z6MkmM42vxfqZQsv4ehtTjFFxQ4sQKS2w6WR7em
    }
  ],
  "authentication": [
    "#key-1"
  ],
  "capabilityInvocation": [
    "#key-1"
  ],
  "capabilityDelegation": [
    "#key-1"
  ],
  "assertionMethod": [
    // Using relative DID URL #key-1 is equivalent to using the abso
    "did:example:123#key-1"
  ]
}
```

A.2 Proving

This section is non-normative.

NOTE

These examples are for information purposes only. See *Verifiable Credentials Data Model v2.0* for additional examples.

EXAMPLE 16: Verifiable Credential linked to a verification method of type Multikey


```
{
  // external (all terms in this example)
  "@context": [
    "https://www.w3.org/ns/credentials/v2",
    "https://w3id.org/citizenship/v4rc1"
  ],
  "type": [
    "VerifiableCredential",
    "PermanentResidentCardCredential"
  ],
  "issuer": {
    "id": "did:key:zDnaeYGXycLmAn5m9akGtdL6rqBspGQPM7QZXW2CvJ3k9c2Bz",
    "image": "...5CYII="
  },
  "name": "Permanent Resident Card",
  "description": "Government of Utopia Permanent Resident Card.",
  "credentialSubject": {
    "type": [
      "PermanentResident",
      "Person"
    ],
    "givenName": "JANE",
    "familyName": "SMITH",
    "gender": "Female",
    "image": "...kJggg==",
    "residentSince": "2015-01-01",
    "commuterClassification": "C1",
    "birthCountry": "Arcadia",
    "birthDate": "1978-07-17",
    "permanentResidentCard": {
      "type": "PermanentResidentCard",
      "identifier": "83627465",
      "lprCategory": "C09",
      "lprNumber": "999-999-999"
    }
  },
  "validFrom": "2025-01-04T00:00:00Z",
  "validUntil": "2026-01-04T23:59:59Z",
  "proof": {
    "type": "DataIntegrityProof",
    "created": "2025-01-04T15:02:36Z",
    "verificationMethod": "did:key:zDnaeYGXycLmAn5m9akGtdL6rqBspGQPM7Q
    "cryptosuite": "ecdsa-rdfc-2019",
```

```
    "proofPurpose": "assertionMethod",
    "proofValue": "z5CK4DPN7...Jpqwp"
  }
}
```

EXAMPLE 17: Verifiable Credential linked to a verification method of type JsonWebKey


```
{  // external (all terms in this example)
  "@context": [
    "https://www.w3.org/ns/credentials/v2",
    "https://w3id.org/citizenship/v4rc1"
  ],
  "type": [
    "VerifiableCredential",
    "PermanentResidentCardCredential"
  ],
  "issuer": {
    "id": "did:example:123#key-1",
    "image": "...5CYII="
  },
  "name": "Permanent Resident Card",
  "description": "Government of Utopia Permanent Resident Card.",
  "credentialSubject": {
    "type": [
      "PermanentResident",
      "Person"
    ],
    "givenName": "JANE",
    "familyName": "SMITH",
    "gender": "Female",
    "image": "...kJggg==",
    "residentSince": "2015-01-01",
    "commuterClassification": "C1",
    "birthCountry": "Arcadia",
    "birthDate": "1978-07-17",
    "permanentResidentCard": {
      "type": "PermanentResidentCard",
      "identifier": "83627465",
      "lprCategory": "C09",
      "lprNumber": "999-999-999"
    }
  },
  "validFrom": "2025-01-04T00:00:00Z",
  "validUntil": "2026-01-04T23:59:59Z",
  "proof": {
    "type": "DataIntegrityProof",
    "created": "2025-01-04T15:02:36Z",
    "verificationMethod": "did:example:123#key-1",
    "cryptosuite": "ecdsa-jcs-2019",
    "proofPurpose": "assertionMethod",
```

```
    "proofValue": "z5m9akGtdL...6rqBspGQP"
  }
}
```

EXAMPLE 18: Verifiable Credential linked to a bls12381 verification method


```
{  // external (all terms in this example)
  "@context": [
    "https://www.w3.org/ns/credentials/v2",
    "https://w3id.org/citizenship/v4rc1"
  ],
  "type": [
    "VerifiableCredential",
    "PermanentResidentCardCredential"
  ],
  "issuer": {
    "id": "did:key:zUC7DojAAkoD8WpSS87KG6iuMSBd4wH1fZzmcwmakx4JfaXN7RL
    "image": "...CYII="
  },
  "name": "Permanent Resident Card",
  "description": "Government of Utopia Permanent Resident Card.",
  "credentialSubject": {
    "type": [
      "PermanentResident",
      "Person"
    ],
    "givenName": "JANE",
    "familyName": "SMITH",
    "gender": "Female",
    "image": "...3dgg==",
    "residentSince": "2015-01-01",
    "commuterClassification": "C1",
    "birthCountry": "Arcadia",
    "birthDate": "1978-07-17",
    "permanentResidentCard": {
      "type": "PermanentResidentCard",
      "identifier": "83627465",
      "lprCategory": "C09",
      "lprNumber": "999-999-999"
    }
  },
  "validFrom": "2025-01-04T00:00:00Z",
  "validUntil": "2026-01-04T23:59:59Z",
  "proof": {
    "type": "DataIntegrityProof",
    "verificationMethod": "did:key:zUC7DojAAkoD8WpSS87KG6iuMSBd4wH1fZz
    "cryptosuite": "bbs-2023",
    "proofPurpose": "assertionMethod",
```

```
    "proofValue": "u2V0ChVhQik2d4...pc3N1ZXI"
  }
}
```

EXAMPLE 19: Verifiable Credential selective disclosure zero knowledge proof linked to a bls12381 verification method


```
{
  // external (all terms in this example)
  "@context": "https://www.w3.org/ns/credentials/v2",
  "type": "VerifiablePresentation",
  // holder did:key is pairwise to the domain to avoid correlation
  "holder": "did:key:z6MkveKdpgkQ1pwNktQ5Lc19epBrzFjMUeNMUZGFvezFF2dX"
  "verifiableCredential": {
    "@context": [
      "https://www.w3.org/ns/credentials/v2",
      "https://w3id.org/citizenship/v4rc1"
    ],
    "type": [
      "VerifiableCredential",
      "PermanentResidentCardCredential"
    ],
    "issuer": {
      "id": "did:web:unlinkable.example",
      "image": "...CYII="
    },
    "credentialSubject": {
      "type": ["PermanentResident", "Person"],
      // only country is selectively disclosed
      "birthCountry": "Arcadia"
    },
    "proof": {
      "type": "DataIntegrityProof",
      "verificationMethod": "did:web:vcplayground.org#zUC7EwMqo9vCjFmj
      "cryptosuite": "bbs-2023",
      "proofPurpose": "assertionMethod",
      "proofValue": "u2V0DhV...3JnIn0"
    }
  },
  "proof": {
    "type": "DataIntegrityProof",
    "created": "2025-01-04T15:10:39Z",
    "verificationMethod": "did:key:z6MkveKdpgkQ1pwNktQ5Lc19epBrzFjMUeN
    "proofPurpose": "authentication",
    "challenge": "QZVVFcXlMPStFmpXTSktv",
    "domain": "https://unlinkable.example",
```

```
    "proofValue": "z5tXmHk...x2GvTt3bF"
  }
}
```

EXAMPLE 20: Verifiable Credential as Decoded JWT

```
{ // external (all terms in this example)
  "protected": {
    "kid": "did:example:123#_Qq0UL2Fq651Q0Fjd6TvnYE-faHiOpRlPVQcY_-tA4
    "alg": "EdDSA"
  },
  "payload": {
    "@context": [
      "https://www.w3.org/ns/credentials/v2",
      "https://w3id.org/citizenship/v4rc1"
    ],
    "type": [
      "VerifiableCredential",
      "PermanentResidentCardCredential"
    ],
    "issuer": {
      "id": "did:key:zUC7Do...oAVE",
      "image": "...CYII="
    },
    "name": "Permanent Resident Card",
    "description": "Government of Utopia Permanent Resident Card.",
    "credentialSubject": {
      "type": [
        "PermanentResident",
        "Person"
      ],
      "givenName": "JANE",
      "familyName": "SMITH",
      "gender": "Female",
      "image": "...3dgg==",
      "residentSince": "2015-01-01",
      "commuterClassification": "C1",
      "birthCountry": "Arcadia",
      "birthDate": "1978-07-17",
      "permanentResidentCard": {
        "type": "PermanentResidentCard",
        "identifier": "83627465",
        "lprCategory": "C09",
        "lprNumber": "999-999-999"
      }
    },
    "validFrom": "2025-01-04T00:00:00Z",
    "validUntil": "2026-01-04T23:59:59Z",
```

```
  },
  "signature": "qSv6d...bJRAw"
}
```

§ A.3 Encrypting

This section is non-normative.

> NOTE
>
> These examples are for information purposes only, it is considered a best practice to avoid dislosing unnecessary information in JWE headers.

EXAMPLE 21: JWE linked to a verification method via kid

```
{ // external (all terms in this example)
  "ciphertext": "3SHQQJajNH6q0fyAHmw...",
  "iv": "QldSPLVnFf2-VXcNLza6mbylYwphW57Q",
  "protected": "eyJlbmMiOiJYQzIwUCJ9",
  "recipients": [
    {
      "encrypted_key": "BMJ19zK12YHftJ4sr6Pz1rX1HtYni_L9DZvO1cEZfRWDN2
      "header": {
        "alg": "ECDH-ES+A256KW",
        "apu": "Tx9qG69ZfodhRos-8qfhTPc6ZFnNUcgNDVdHqX1UR3s",
        "apv": "ZGlkOmVsZW06cm9wc3RlbjpFa...",
        "epk": {
          "crv": "X25519",
          "kty": "OKP",
          "x": "Tx9qG69ZfodhRos-8qfhTPc6ZFnNUcgNDVdHqX1UR3s"
        },
        "kid": "did:example:123#zC1Rnuvw9rVa6E5TKF4uQVRuQuaCpVgB81Um2u
      }
    }
  ],
  "tag": "xbfwwDkzOAJfSVem0jr1bA"
}
```

§ B. Architectural Considerations

This section is non-normative.

§ B.1 Detailed Architecture Diagram

This section is non-normative.

Following is a diagram showing the relationships among 4. Data Model, 5. Core Properties, and 7. Methods, and [*DID-RESOLUTION*].

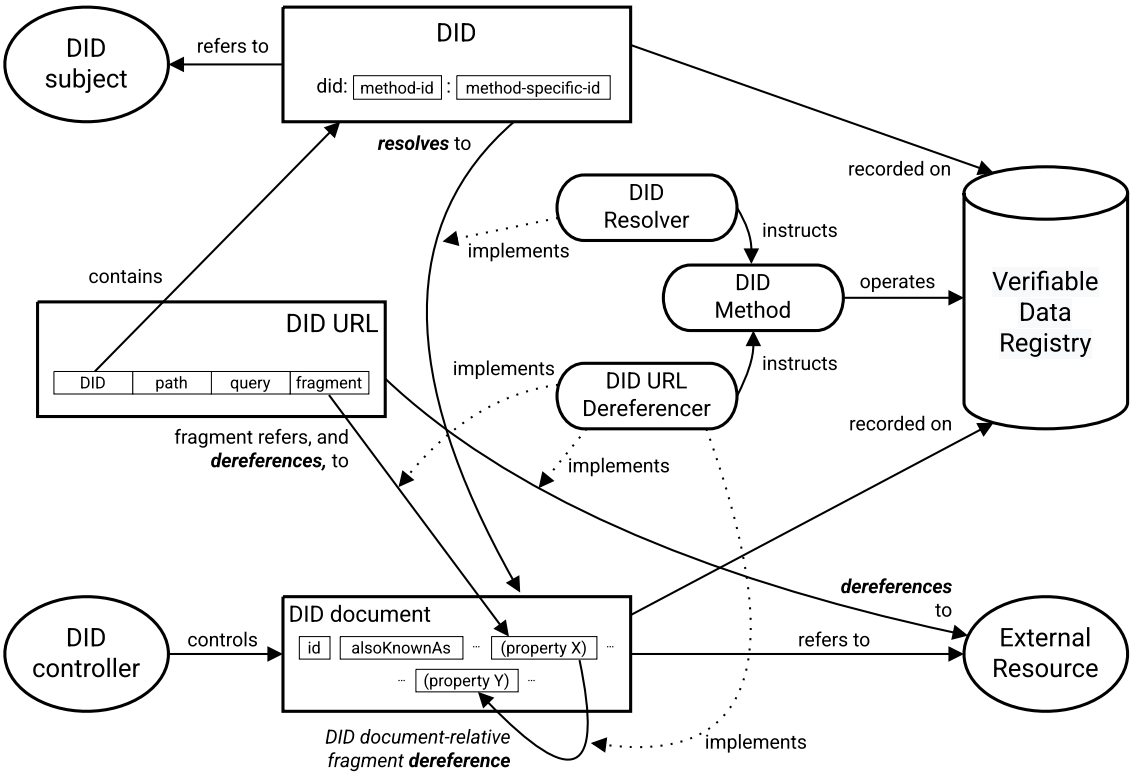

Figure 4 Detailed overview of DID architecture and the relationship of the basic components.

§ B.2 Creation of a DID

This section is non-normative.

The creation of a DID is a process that is defined by each DID Method. Some DID Methods, such as `did:key`, are purely generative, such that a DID and a DID document are generated by transforming a single piece of cryptographic material into a conformant representation. Other DID methods might require the use of a verifiable data registry, where the DID and DID document are recognized to exist by third parties only when the registration has been completed, as defined by the respective DID method. Other processes might be defined by the respective DID method.

§ B.3 Determining the DID subject

This section is non-normative.

A DID is a specific type of URI (Uniform Resource Identifier), so a DID can refer to any resource. Per [*RFC3986*]:

> the term "resource" is used in a general sense for whatever might be identified by a URI. [...] A resource is not necessarily accessible via the Internet.

Resources can be digital or physical, abstract or concrete. Any resource that can be assigned a URI can be assigned a DID. The resource referred to by the DID is the DID subject.

The DID controller determines the DID subject. It is not expected to be possible to determine the DID subject from looking at the DID itself, as DIDs are generally only meaningful to machines, not human. A DID is unlikely to contain any information about the DID subject, so further information about the DID subject is only discoverable by resolving the DID to the DID document, obtaining a verifiable credential about the DID, or via some other description of the DID.

While the value of the `id` property in the retrieved DID document must always match the DID being resolved, whether or not the actual resource to which the DID refers can change over time is dependent upon the DID method. For example, a DID method that permits the DID subject to change could be used to generate a DID for the current occupant of a particular role—such as the CEO of a company—where the actual person occupying the role can be different depending on when the DID is resolved.

§ B.4 Referring to the DID document

This section is non-normative.

The DID refers to the DID subject and *resolves to* the DID document (by following the protocol specified by the DID method). The DID document is not a separate resource from the DID subject and does not have a URI separate from the DID. Rather the DID document is an artifact of DID resolution controlled by the DID controller for the purpose of enabling cryptographically verifiable interactions with a DID subject.

This distinction is illustrated by the graph model shown below.

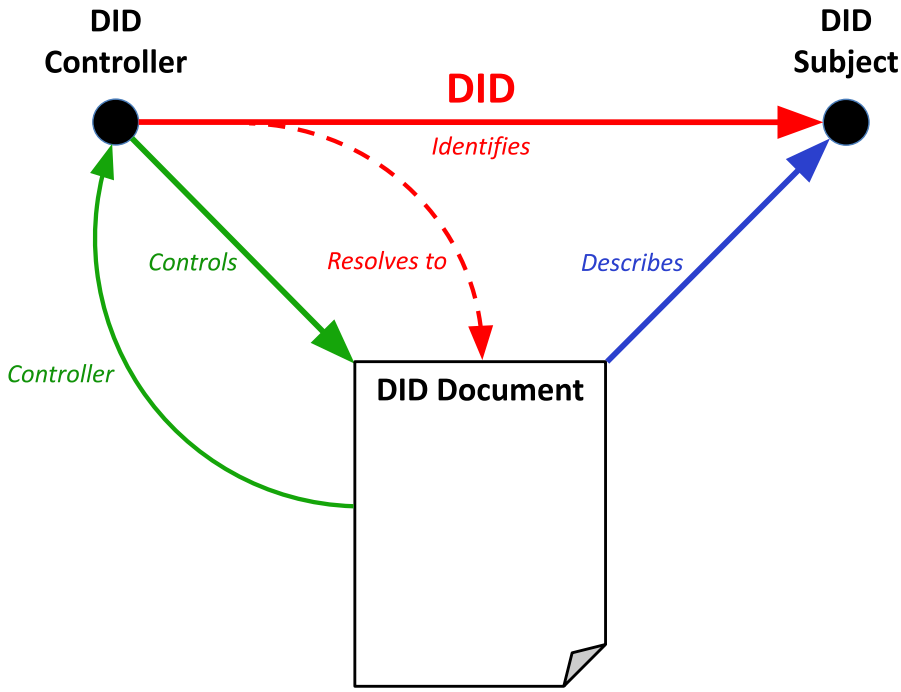

Figure 5 A DID is an identifier assigned by a DID controller to refer to a DID subject and resolve to a DID document that describes the DID subject. The DID document is an artifact of DID resolution and not a separate resource distinct from the DID subject. See also: narrative description.

§ B.5 Statements in the DID document

This section is non-normative.

Each property in a DID document is a statement by the DID controller that describes:

- The string of characters defining identifiers for the DID subject (e.g., the `id` and `alsoKnownAs` properties)
- How to interact with the DID subject (e.g., the `verificationMethod` and `service` properties).
- How to interpret the specific representation of the DID document (e.g., the `@context` property for a JSON-LD representation).

The only required property in a DID document is `id`, so that is the only statement guaranteed to be in a DID document. That statement is illustrated in Figure 5 with a direct link between the DID and the DID subject.

§ B.6 Discovering more information about the DID subject

This section is non-normative.

Options for discovering more information about the DID subject depend on the properties present in the DID document. If the `service` property is present, more information can be requested from a service endpoint. For example, by querying a service endpoint that supports verifiable credentials for one or more claims (attributes) describing the DID subject.

Another option is to use the `alsoKnownAs` property if it is present in the DID document. The DID controller can use it to provide a list of other URIs (including other DIDs) that refer to the same DID subject. Resolving or dereferencing these URIs might yield other descriptions or representations of the DID subject as illustrated in the figure below.

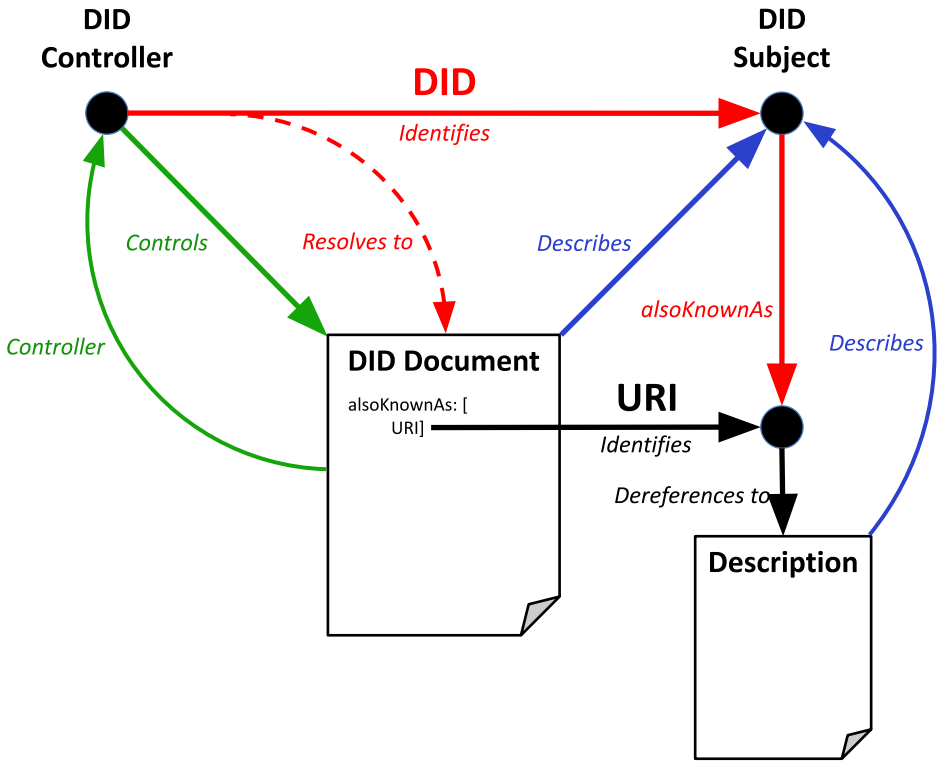

Figure 6 A DID document can use the alsoKnownAs property to assert that another URI (including, but not necessarily, another DID) refers to the same DID subject. See also: narrative description.

§ B.7 Serving a representation of the DID subject

This section is non-normative.

If the DID subject is a digital resource that can be retrieved from the internet, a DID method can choose to construct a DID URL which returns a representation of the DID subject itself. For example, a data schema that needs a persistent, cryptographically verifiable identifier could be assigned a DID, and passing a specified Path or Query could be used as a standard way to retrieve a representation of that schema.

Similarly, a DID can be used to refer to a digital resource (such as an image) that can be returned directly from a verifiable data registry if that functionality is supported by the applicable DID method.

§ B.8 Assigning DIDs to existing web resources

This section is non-normative.

If the controller of a web page or any other web resource wants to assign it a persistent, cryptographically verifiable identifier, the controller can give it a DID. For example, the author of a blog hosted by a blog hosting company (under that hosting company's domain) could create a DID for the blog. In the DID document, the author can include the `alsoKnownAs` property pointing to the current URL of the blog, e.g.:

```
"alsoKnownAs": ["https://myblog.blogging-host.example/home"]
```

If the author subsequently moves the blog to a different hosting company (or to the author's own domain), the author can update the DID document to point to the new URL for the blog, e.g.:

```
"alsoKnownAs": ["https://myblog.example/"]
```

The DID effectively adds a layer of indirection for the blog URL. This layer of indirection is under the control of the author instead of under the control of an external administrative authority such as the blog hosting company. This is how a DID can effectively function as an enhanced URN (Uniform Resource Name)—a persistent identifier for an information resource whose network location might change over time.

§ B.9 The relationship between DID controllers and DID subjects

This section is non-normative.

To avoid confusion, it is helpful to classify DID subjects into two disjoint sets based on their relationship to the DID controller.

§ B.9.1 Set #1: The DID subject *is* the DID controller

This section is non-normative.

The first case, shown in Figure 7, is the common scenario where the DID subject is also the DID controller. This is the case when an individual or organization creates a DID to self-identify.

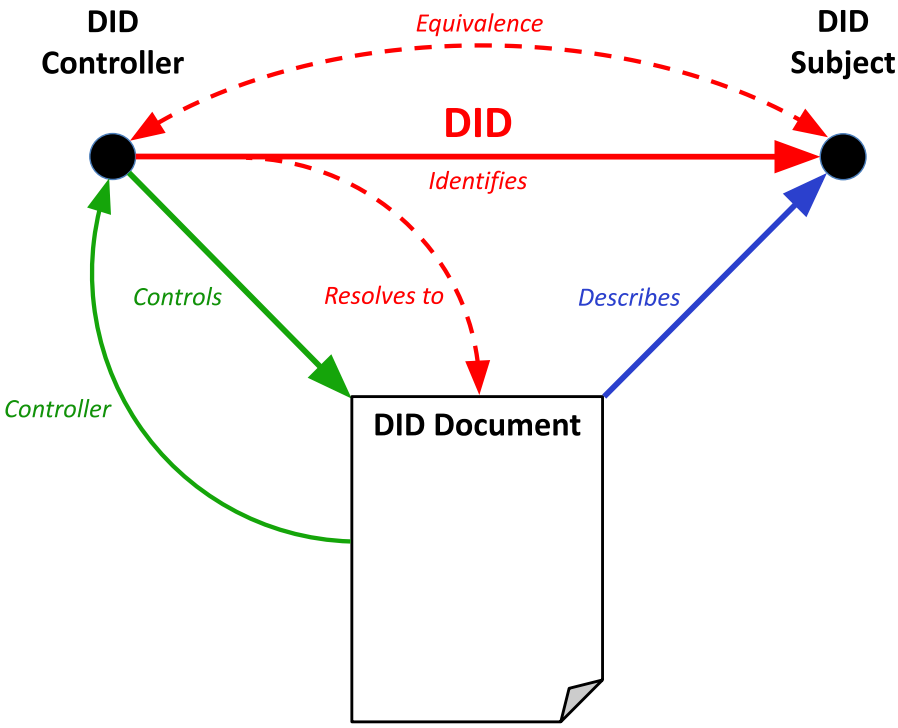

Figure 7 The DID subject is the same entity as the DID controller. See also: narrative description.

From a graph model perspective, even though the nodes identified as the DID controller and DID subject in Figure 7 are distinct, there is a logical arc connecting them to express a semantic equivalence relationship.

§ B.9.2 Set #2: The DID subject is *not* the DID controller

This section is non-normative.

The second case is when the DID subject is a separate entity from the DID controller. This is the case when, for example, a parent creates and maintains control of a DID for a child; a corporation creates and maintains control of a DID for a subsidiary; or a manufacturer creates and maintains control of a DID for a product, an IoT device, or a digital file.

From a graph model perspective, the only difference from Set 1 that there is no equivalence arc relationship between the DID subject and DID controller nodes.

§ B.10 Multiple DID controllers

This section is non-normative.

A DID document might have more than one DID controller. This can happen in one of two ways.

§ B.10.1 Independent Control

This section is non-normative.

In this case, each of the DID controllers might act on its own, i.e., each one has full power to update the DID document independently. From a graph model perspective, in this configuration:

- Each additional DID controller is another distinct graph node (which might be identified by its own DID).
- The same arcs ("controls" and "controller") exist between each DID controller and the DID document.

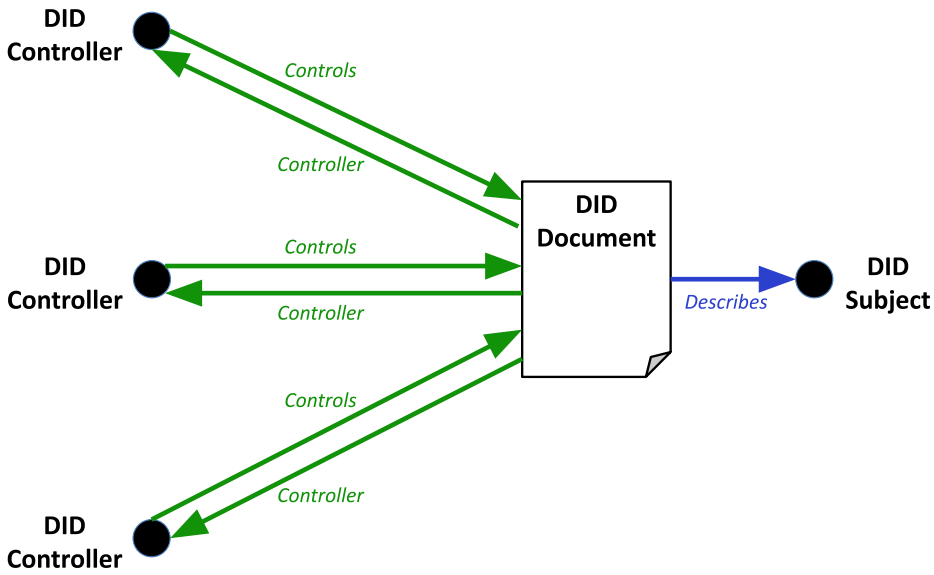

Figure 8 Multiple independent DID controllers that can each act independently. See also: Text Description

§ B.10.2 Group Control

This section is non-normative.

In the case of group control, the DID controllers are expected to act together in some fashion, such as when using a cryptographic algorithm that requires multiple

digital signatures ("multi-sig") or a threshold number of digital signatures ("m-of-n"). These additional thresholds for verifying a proof can be expressed in a verification method as described in Section 5.2 Verification Methods or can be an intrinsic part of the verification material of the verification method, where the number of DID controllers that participated in the generation of a particular digital signature are hidden for privacy reasons. Verification methods that require a proof be produced by a combination of cryptographic operations performed by members of a group can be used to control the contents of a DID document; exactly how this is realized depends on individual DID method specifications.

From a functional standpoint, this option is similar to a single DID controller because, although each of the DID controllers in the DID controller group has its own graph node, the actual control collapses into a single logical graph node representing the DID controller group as shown in Figure 9.

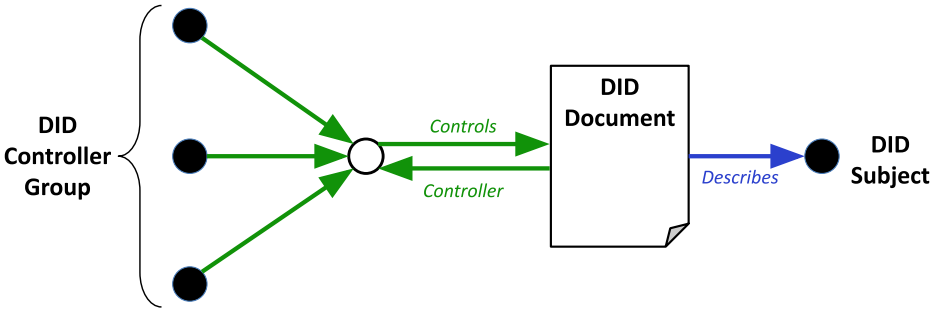

Figure 9 Multiple DID controllers who are expected to act together as a DID controller group. See also: narrative description.

This configuration will often apply when the DID subject is an organization, corporation, government agency, community, or other group that is not controlled by a single individual.

§ B.11 Changing the DID subject

This section is non-normative.

A DID document has exactly one DID which refers to the DID subject. The DID is expressed as the value of the `id` property. This property value is immutable for the lifetime of the DID document.

However, it is possible that the resource *identified* by the DID, the DID subject, may change over time. This is under the exclusive authority of the DID controller. For more details, see section 8.10 Persistence.

§ B.12 Changing the DID controller

This section is non-normative.

The DID controller for a DID document might change over time. However, depending on how it is implemented, a change in the DID controller might not be made apparent by changes to the DID document itself. For example, if the change is implemented through a shift in ownership of the underlying cryptographic keys or other controls used for one or more of the verification methods in the DID document, it might be indistinguishable from a standard key rotation.

On the other hand, if the change is implemented by changing the value of the `controller` property, it will be transparent.

If it is important to verify a change of DID controller, implementers are advised to authenticate the new DID controller against the verification methods in the revised DID document.

§ C. Revision History

This section is non-normative.

This section contains the changes that have been made since the publication of this specification as a W3C First Public Working Draft.

Changes since the DID v1.0 Recommendation include:

- Editorial updates to explanations, grammar, and examples, to improve readability of the specification.
- Update references about HTTP semantics to the *HTTP Semantics* specification.
- Clarified fragment resolution algorithm.
- Consolidated media types to `application/did` after IANA registration process completed.
- Added JSON-LD Context for v1.1.
- Moved sections on resolution to the *Decentralized Identifier Resolution (DID Resolution) v0.3* specification.
- Refactored specification to layer on top of the *Controlled Identifiers v1.0* specification.

Changes since the DID v1.0 Second Candidate Recommendation include:

- Non-normatively refer to the DID Resolution specification to guide implementers toward common DID URL implementation patterns.
- Elaborate upon when DID Documents are understood to start existing.
- Convert PNG diagrams to SVG diagrams.
- Rearrange order of Appendices to improve readability.
- Update the IANA guidance as a result of the IETF Media Type Maintenance Working Group efforts.
- Add links to use cases document.
- Add warning related to Multibase and `publicKeyMultibase` [*CID*].
- Remove at risk issue markers for features that gained enough implementation experience.
- Finalize the Editors, Authors, and Acknowledgements information.

Changes since the DID v1.0 First Candidate Recommendation include:

- Addition of at risk markers to most of the DID Parameters, the data model datatypes that are expected to not be implemented, and the application/did media type. This change resulted in the DID WG's decision to perform a second Candidate Recommendation phase. All other changes were either editorial or predicted in "at risk" issue markers.
- Removal of the at risk issue marker for the `method-specific-id` ABNF rule and for `nextUpdate` and `nextVersionId`.
- Clarification that `equivalentId` and `canonicalId` are optional.
- Addition of a definitions for "amplification attack" and "cryptographic suite".
- Replacement of `publicKeyBase58` with `publicKeyMultibase`.
- Updates to the DID Document examples section.
- A large number of editorial clean ups to the Security Considerations section.

Changes since the DID v1.0 First Public Working Draft include:

- The introduction of an abstract data model that can be serialized to multiple representations including JSON and JSON-LD.
- The introduction of a repository of DID Extensions for the purposes of registering extension properties, representations, DID Resolution input metadata and output metadata, DID Document metadata, DID parameters, and DID Methods.

- Separation of DID Document metadata, such as created and updated values, from DID Document properties.
- The removal of embedded proofs in the DID Document.
- The addition of verification relationships for the purposes of authentication, assertion, key agreement, capability invocation and capability delegation.
- The ability to support relating multiple identifiers with the DID Document, such as the DID controller, also known as, equivalent IDs, and canonical IDs.
- Enhancing privacy by reducing information that could contain personally identifiable information in the DID Document.
- The addition of a large section on security considerations and privacy considerations.
- A Representations section that details how the abstract data model can be produced and consumed in a variety of different formats along with general rules for all representations, producers, and consumers.
- A section detailing the DID Resolution and DID URL Dereferencing interface definition that all DID resolvers are expected to expose as well as inputs and outputs to those processes.
- DID Document examples in an appendix that provide more complex examples of DID Document serializations.
- IANA Considerations for multiple representations specified in DID Core.
- Removal of the Future Work section as much of the work has now been accomplished.
- An acknowledgements section.

§ D. Acknowledgements

This section is non-normative.

The Working Group extends deep appreciation and heartfelt thanks to our Chairs Brent Zundel and Dan Burnett, as well as our W3C Staff Contact, Ivan Herman, for their tireless work in keeping the Working Group headed in a productive direction and navigating the deep and dangerous waters of the standards process.

The Working Group gratefully acknowledges the work that led to the creation of this specification, and extends sincere appreciation to those individuals that worked on technologies and specifications that deeply influenced our work. In particular, this includes the work of Phil Zimmerman, Jon Callas, Lutz

Donnerhacke, Hal Finney, David Shaw, and Rodney Thayer on Pretty Good Privacy (PGP) in the 1990s and 2000s.

In the mid-2010s, preliminary implementations of what would become Decentralized Identifiers were built in collaboration with Jeremie Miller's Telehash project and the W3C Web Payments Community Group's work led by Dave Longley and Manu Sporny. Around a year later, the XDI.org Registry Working Group began exploring decentralized technologies for replacing its existing identifier registry. Some of the first written papers exploring the concept of Decentralized Identifiers can be traced back to the first several Rebooting the Web of Trust workshops convened by Christopher Allen. That work led to a key collaboration between Christopher Allen, Drummond Reed, Les Chasen, Manu Sporny, and Anil John. Anil saw promise in the technology and allocated the initial set of government funding to explore the space. Without the support of Anil John and his guidance through the years, it is unlikely that Decentralized Identifiers would be where they are today. Further refinement at the Rebooting the Web of Trust workshops led to the first implementers documentation, edited by Drummond Reed, Les Chasen, Christopher Allen, and Ryan Grant. Contributors included Manu Sporny, Dave Longley, Jason Law, Daniel Hardman, Markus Sabadello, Christian Lundkvist, and Jonathan Endersby. This initial work was then merged into the W3C Credentials Community Group, incubated further, and then transitioned to the W3C Decentralized Identifiers Working Group for global standardization.

Portions of the work on this specification have been funded by the United States Department of Homeland Security's (US DHS) Science and Technology Directorate under contracts HSHQDC-16-R00012-H-SB2016-1-002, and HSHQDC-17-C-00019, as well as the US DHS Silicon Valley Innovation Program under contracts 70RSAT20T00000010, 70RSAT20T00000029, 70RSAT20T00000030, 70RSAT20T00000045, 70RSAT20T00000003, and 70RSAT20T00000033. The content of this specification does not necessarily reflect the position or the policy of the U.S. Government and no official endorsement should be inferred.

Portions of the work on this specification have also been funded by the European Union's StandICT.eu program under sub-grantee contract number CALL05/19. The content of this specification does not necessarily reflect the position or the policy of the European Union and no official endorsement should be inferred.

Work on this specification has also been supported by the Rebooting the Web of Trust community facilitated by Christopher Allen, Shannon Appelcline, Kiara Robles, Brian Weller, Betty Dhamers, Kaliya Young, Kim Hamilton Duffy, Manu Sporny, Drummond Reed, Joe Andrieu, and Heather Vescent. Development of this specification has also been supported by the W3C Credentials Community Group, which has been Chaired by Kim Hamilton Duffy, Joe Andrieu, Christopher Allen,

Heather Vescent, and Wayne Chang. The participants in the Internet Identity Workshop, facilitated by Phil Windley, Kaliya Young, Doc Searls, and Heidi Nobantu Saul, also supported this work through numerous working sessions designed to debate, improve, and educate participants about this specification.

The Working Group thanks the following individuals for their contributions to this specification (in alphabetical order, Github handles start with @ and are sorted as last names): Denis Ah-Kang, Nacho Alamillo, Christopher Allen, Joe Andrieu, Antonio, Phil Archer, George Aristy, Baha, Juan Benet, BigBlueHat, Dan Bolser, Chris Boscolo, Pelle Braendgaard, Daniel Buchner, Daniel Burnett, Juan Caballero, @cabo, Tim Cappalli, Melvin Carvalho, David Chadwick, Wayne Chang, Sam Curren, Hai Dang, Tim Daubenschütz, Oskar van Deventer, Kim Hamilton Duffy, Arnaud Durand, Ken Ebert, Veikko Eeva, @ewagner70, Carson Farmer, Nikos Fotiou, Gabe, Gayan, @gimly-jack, @gjgd, Ryan Grant, Peter Grassberger, Adrian Gropper, Amy Guy, Daniel Hardman, Kyle Den Hartog, Philippe Le Hegaret, Ivan Herman, Michael Herman, Alen Horvat, Dave Huseby, Marcel Jackisch, Mike Jones, Andrew Jones, Tom Jones, jonnycrunch, Gregg Kellogg, Michael Klein, @kdenhartog-sybil1, Paul Knowles, @ktobich, David I. Lehn, Charles E. Lehner, Michael Lodder, @mooreT1881, Dave Longley, Tobias Looker, Wolf McNally, Robert Mitwicki, Mircea Nistor, Grant Noble, Mark Nottingham, @oare, Darrell O'Donnell, Vinod Panicker, Dirk Porsche, Praveen, Mike Prorock, @pukkamustard, Drummond Reed, Julian Reschke, Yancy Ribbens, Justin Richer, Rieks, @rknobloch, Mikeal Rogers, Evstifeev Roman, Troy Ronda, Leonard Rosenthol, Michael Ruminer, Markus Sabadello, Cihan Saglam, Samu, Rob Sanderson, Wendy Seltzer, Mehran Shakeri, Jaehoon (Ace) Shim, Samuel Smith, James M Snell, SondreB, Manu Sporny, @ssstolk, Orie Steele, Shigeya Suzuki, Sammotic Switchyarn, @tahpot, Oliver Terbu, Ted Thibodeau Jr., Joel Thorstensson, Tralcan, Henry Tsai, Rod Vagg, Mike Varley, Kaliya "Identity Woman" Young, Eric Welton, Fuqiao Xue, @Yue, Dmitri Zagidulin, @zhanb, and Brent Zundel.

§ E. IANA Considerations

This section will be submitted to the Internet Engineering Steering Group (IESG) for review, approval, and registration with IANA when this specification becomes a W3C Proposed Recommendation.

§ E.1 application/did

Type name:
application

Subtype name:
did

Required parameters:
None

Optional parameters:
None

Encoding considerations:
See RFC 8259, section 11.

Security considerations:
See DID Core v1.1, Security Considerations.

Interoperability considerations:
Not Applicable

Published specification:
https://www.w3.org/TR/did/

Applications that use this media type:
Any application that requires an identifier that is decentralized, persistent, cryptographically verifiable, and resolvable. Applications typically consist of cryptographic identity systems, decentralized networks of devices, and websites that issue or verify verifiable credentials.

Additional information:

Magic number(s):
Not Applicable

File extension(s):
.did

Macintosh file type code(s):
TEXT

Email address to contact for further information:
W3C Decentralized Identifiers Working Group public-did-wg@w3.org

Intended usage:
Common

Restrictions on usage:
None

Author(s):

Drummond Reed, Manu Sporny, Markus Sabadello, Dave Longley, Christopher Allen

Change controller:

W3C

Fragment identifiers used in JSON-LD environments are treated according to the rules associated with the JSON-LD 1.1: application/ld+json media type [*JSON-LD11*]. Fragment identifiers used in JSON environments have the same semantic interpretation as those in JSON-LD environments. An algorithm for performing fragment resolution is provided in Section 3.4: Fragment Resolution of the *Controlled Identifiers v1.0* specification which is extended by the *Decentralized Identifiers (DIDs) v1.1* specification.

F. References

F.1 Normative references

[CID]

Controlled Identifiers v1.0. Michael Jones; Manu Sporny. W3C. 15 May 2025. W3C Recommendation. URL: https://www.w3.org/TR/cid-1.0/

[DID-CORE]

Decentralized Identifiers (DIDs) v1.0. Manu Sporny; Amy Guy; Markus Sabadello; Drummond Reed. W3C. 19 July 2022. W3C Recommendation. URL: https://www.w3.org/TR/did-core/

[INFRA]

Infra Standard. Anne van Kesteren; Domenic Denicola. WHATWG. Living Standard. URL: https://infra.spec.whatwg.org/

[JSON-LD11]

JSON-LD 1.1. Gregg Kellogg; Pierre-Antoine Champin; Dave Longley. W3C. 16 July 2020. W3C Recommendation. URL: https://www.w3.org/TR/json-ld11/

[RFC2119]

Key words for use in RFCs to Indicate Requirement Levels. S. Bradner. IETF. March 1997. Best Current Practice. URL: https://www.rfc-editor.org/rfc/rfc2119

[RFC3552]

Guidelines for Writing RFC Text on Security Considerations. E. Rescorla; B. Korver. IETF. July 2003. Best Current Practice. URL: https://www.rfc-editor.org/rfc/rfc3552

[RFC3986]
Uniform Resource Identifier (URI): Generic Syntax. T. Berners-Lee; R. Fielding; L. Masinter. IETF. January 2005. Internet Standard. URL: https://www.rfc-editor.org/rfc/rfc3986

[RFC5234]
Augmented BNF for Syntax Specifications: ABNF. D. Crocker, Ed.; P. Overell. IETF. January 2008. Internet Standard. URL: https://www.rfc-editor.org/rfc/rfc5234

[RFC6838]
Media Type Specifications and Registration Procedures. N. Freed; J. Klensin; T. Hansen. IETF. January 2013. Best Current Practice. URL: https://www.rfc-editor.org/rfc/rfc6838

[RFC8174]
Ambiguity of Uppercase vs Lowercase in RFC 2119 Key Words. B. Leiba. IETF. May 2017. Best Current Practice. URL: https://www.rfc-editor.org/rfc/rfc8174

[RFC8259]
The JavaScript Object Notation (JSON) Data Interchange Format. T. Bray, Ed. IETF. December 2017. Internet Standard. URL: https://www.rfc-editor.org/rfc/rfc8259

[URL]
URL Standard. Anne van Kesteren. WHATWG. Living Standard. URL: https://url.spec.whatwg.org/

[XMLSCHEMA11-2]
W3C XML Schema Definition Language (XSD) 1.1 Part 2: Datatypes. David Peterson; Sandy Gao; Ashok Malhotra; Michael Sperberg-McQueen; Henry Thompson; Paul V. Biron et al. W3C. 5 April 2012. W3C Recommendation. URL: https://www.w3.org/TR/xmlschema11-2/

§ F.2 Informative references

[DID-1.1]
Decentralized Identifiers (DIDs) v1.1. Manu Sporny; Dmitri Zagidulin. W3C. 5 March 2026. W3C Candidate Recommendation. URL: https://www.w3.org/TR/did-1.1/

[DID-EXTENSIONS]
Decentralized Identifier Extensions. Manu Sporny; Markus Sabadello. W3C. 11 December 2025. W3C Working Group Note. URL: https://www.w3.org/TR/did-extensions/

[DID-RESOLUTION]

Decentralized Identifier Resolution (DID Resolution) v0.3. Markus Sabadello; Dmitri Zagidulin. W3C. 9 March 2026. W3C Working Draft. URL: https://www.w3.org/TR/did-resolution/

[DID-RUBRIC]

DID Method Rubric v1.0. Joe Andrieu; Daniel Hardman. W3C. 19 November 2021. W3C Working Group Note. URL: https://www.w3.org/TR/did-rubric/

[DID-USE-CASES]

Use Cases and Requirements for Decentralized Identifiers. Joe Andrieu; Phil Archer; Kim Duffy; Ryan Grant; Adrian Gropper. W3C. 17 March 2021. W3C Working Group Note. URL: https://www.w3.org/TR/did-use-cases/

[DNS-DID]

The Decentralized Identifier (DID) in the DNS. Alexander Mayrhofer; Dimitrij Klesev; Markus Sabadello. February 2019. Internet-Draft. URL: https://datatracker.ietf.org/doc/draft-mayrhofer-did-dns/

[IANA-URI-SCHEMES]

Uniform Resource Identifier (URI) Schemes. IANA. URL: https://www.iana.org/assignments/uri-schemes/uri-schemes.xhtml

[MATRIX-URIS]

Matrix URIs - Ideas about Web Architecture. Tim Berners-Lee. December 1996. Personal View. URL: https://www.w3.org/DesignIssues/MatrixURIs.html

[PRIVACY-BY-DESIGN]

Privacy by Design. Ann Cavoukian. Information and Privacy Commissioner. 2011. URL: https://iapp.org/media/pdf/resource_center/pbd_implement_7found_principles.pdf

[RFC6901]

JavaScript Object Notation (JSON) Pointer. P. Bryan, Ed.; K. Zyp; M. Nottingham, Ed. IETF. April 2013. Proposed Standard. URL: https://www.rfc-editor.org/rfc/rfc6901

[RFC6973]

Privacy Considerations for Internet Protocols. A. Cooper; H. Tschofenig; B. Aboba; J. Peterson; J. Morris; M. Hansen; R. Smith. IETF. July 2013. Informational. URL: https://www.rfc-editor.org/rfc/rfc6973

[RFC8141]

Uniform Resource Names (URNs). P. Saint-Andre; J. Klensin. IETF. April 2017. Proposed Standard. URL: https://www.rfc-editor.org/rfc/rfc8141

[RFC9110]

HTTP Semantics. R. Fielding, Ed.; M. Nottingham, Ed.; J. Reschke, Ed. IETF. June 2022. Internet Standard. URL: https://httpwg.org/specs/rfc9110.html

[VC-DATA-MODEL]

Verifiable Credentials Data Model v2.0. Ivan Herman; Michael Jones; Manu Sporny; Ted Thibodeau Jr; Gabe Cohen. W3C. 15 May 2025. W3C Recommendation. URL: https://www.w3.org/TR/vc-data-model-2.0/

↥