Decentralized Identifiers (DIDs) v1.0

Core architecture, data model, and representations

W3C Proposed Recommendation 03 August 2021

This version:

https://www.w3.org/TR/2021/PR-did-core-20210803/

Latest published version:

https://www.w3.org/TR/did-core/

Latest editor's draft:

https://w3c.github.io/did-core/

Implementation report:

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

Previous version:

https://www.w3.org/TR/2021/CRD-did-core-20210730/

Editors:

Manu Sporny (Digital Bazaar)

Amy Guy (Digital Bazaar)

Markus Sabadello (Danube Tech)

Drummond Reed (Evernym)

Authors:

Manu Sporny (Digital Bazaar)

Dave Longley (Digital Bazaar)

Markus Sabadello (Danube Tech)

Drummond Reed (Evernym)

Orie Steele (Transmute)

Christopher Allen (Blockchain Commons)

Participate:

GitHub w3c/did-core

File an issue

Commit history

Pull requests

Related Documents

DID Use Cases and Requirements

DID Specification Registries

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. Other documents may supersede this document. A list of current W3C publications and the latest revision of this technical report can be found in the W3C technical reports index at https://www.w3.org/TR/.

The W3C Decentralized Identifier Working Group has published this document as a W3C Proposed Recommendation and is requesting that interested parties review this specification by August 31st, 2021.

At the time of publication, there existed 103 experimental DID Method specifications, 32 experimental DID Method driver implementations, a test suite that determines whether or not a given implementation is conformant with this specification and 46 implementations submitted to the conformance test suite. Readers are advised to heed the DID Core issues and DID Core Test Suite issues that each contain the latest list of concerns and proposed changes that might result in

alterations to this specification. At the time of publication, no additional substantive issues, changes, or modifications are expected.

Comments regarding this document are welcome. Please file issues directly on GitHub, or send them to public-did-wg@w3.org (subscribe, archives).

This document was published by the Decentralized Identifier Working Group as a Proposed Recommendation. This document is intended to become a W3C Recommendation.

GitHub Issues are preferred for discussion of this specification. Alternatively, you can send comments to our mailing list. Please send them to public-did-wg@w3.org (subscribe, archives).

Publication as a Proposed Recommendation does not imply endorsement by the W3C Membership.

This is a draft document and may be updated, replaced or obsoleted by other documents at any time. It is inappropriate to cite this document as other than work in progress.

The W3C Membership and other interested parties are invited to review the document and send comments through 31 August 2021. Advisory Committee Representatives should consult their WBS questionnaires. Note that substantive technical comments were expected during the Candidate Recommendation review period that ended 13 July 2021.

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 which 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 15 September 2020 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.

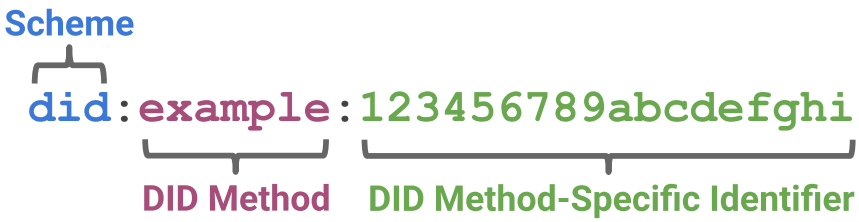

Figure 1 A simple example of a decentralized identifier (DID)

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",
    "https://w3id.org/security/suites/ed25519-2020/v1"
  ]
  "id": "did:example:123456789abcdefghi",
  "authentication": [{
    // used to authenticate as did:...fghi
    "id": "did:example:123456789abcdefghi#keys-1",
    "type": "Ed25519VerificationKey2020",
    "controller": "did:example:123456789abcdefghi",
    "publicKeyMultibase": "zH3C2AVvLMv6gmMNam3uVAjZpfkcJCwDwnZn6z3wXmqP
  }]
}
```

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 point 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.
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.

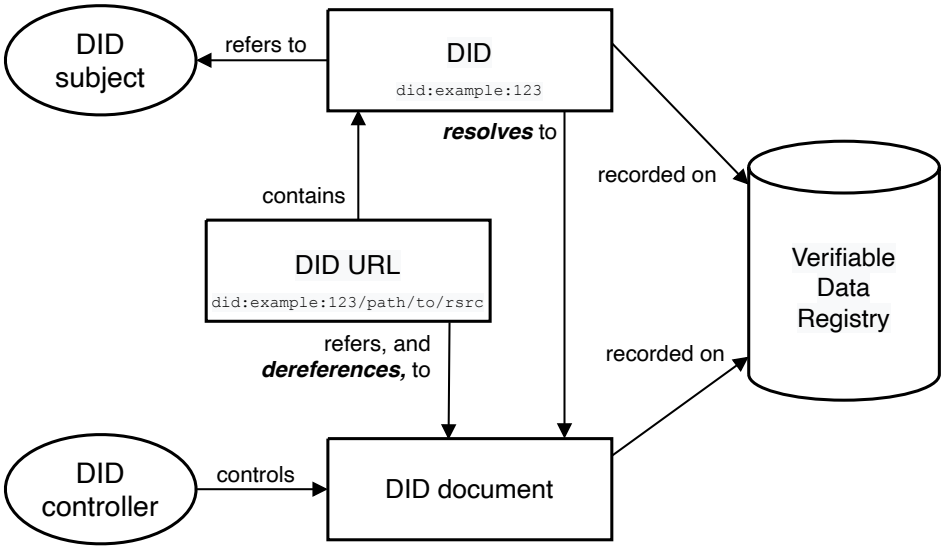

Figure 2 Overview of DID architecture and the relationship of the basic components. See also: narrative description.

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 § 8. 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 § 8. 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 § 8. 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 § 7. 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 § 7.2 DID URL Dereferencing.

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*, *OPTIONAL*, *RECOMMENDED*, *REQUIRED*, *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 DID Specification Registries [DID-SPEC-REGISTRIES] 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 resolver*** is any algorithm realized as software and/or hardware that complies with the relevant normative statements in § 7.1 DID Resolution.

A ***conforming DID URL dereferencer*** is any algorithm realized as software and/or hardware that complies with the relevant normative statements in § 7.2 DID URL Dereferencing.

A ***conforming DID method*** is any specification that complies with the relevant normative statements in § 8. Methods.

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.

authenticate

Authentication is a process by which an entity can prove it has a specific attribute or controls a specific secret using one or more verification methods. With DIDs, a common example would be proving control of the cryptographic private key associated with a public key published in a DID document.

cryptographic suite

A specification defining the usage of specific cryptographic primitives in order to achieve a particular security goal. These documents are often used to specify verification methods, digital signature types, their identifiers, and other related properties.

***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.

decentralized identity management

Identity management that is based on the use of decentralized identifiers. Decentralized identity management extends authority for identifier generation, registration, and assignment beyond traditional roots of trust such as X.500 directory services, the Domain Name System, and most national ID systems.

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 describing the DID subject, including mechanisms, such as cryptographic public keys, that the DID subject or a DID delegate can use to authenticate itself and prove its association with the DID. A DID document might have one or more different representations as defined in § 6. Representations or in the W3C DID Specification Registries [DID-SPEC-REGISTRIES].

DID fragment

The portion of a DID URL that follows the first hash sign character (#). 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 § 8. Methods.

DID path

The portion of a DID URL that begins with and includes the first forward slash (`/`) character and ends with either a question mark (`?`) character, a fragment hash sign (`#`) character, 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 question mark character (`?`). 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 § 7.1 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), optional DID query (with its leading `?` character), and optional DID fragment (with its leading `#` character).

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 § 7.2 DID URL Dereferencing.

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.

public key description

A data object contained inside a DID document that contains all the metadata necessary to use a public key or a verification key.

resource

As defined by [RFC3986]: "...the term 'resource' is used in a general sense for whatever might be identified by a URI." Similarly, any resource might serve as a DID subject identified by a DID.

representation

As defined for HTTP by [RFC7231]: "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, and that consists of a set of representation metadata and a potentially unbounded stream of representation data." A DID document is a representation of information describing a DID subject. See § 6. Representations.

services

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.

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 credential

A standard data model and representation format for cryptographically-verifiable digital credentials as defined by the W3C Verifiable Credentials specification [VC-DATA-MODEL].

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.

verification method
: A set of parameters that can be used together with a process to independently verify a proof. For example, a cryptographic public key can be used as a verification method with respect to a digital signature; in such usage, it verifies that the signer possessed the associated cryptographic private key.

  "Verification" and "proof" in this definition are intended to apply broadly. For example, a cryptographic public key might be used during Diffie-Hellman key exchange to negotiate a shared symmetric key for encryption. This guarantees the integrity of the key agreement process. It is thus another type of verification method, even though descriptions of the process might not use the words "verification" or "proof."

verification relationship
: An expression of the relationship between the DID subject and a verification method. An example of a verification relationship is § 5.3.1 Authentication.

Universally Unique Identifier **(UUID)**
: A type of globally unique identifier defined by [RFC4122]. UUIDs are similar to DIDs in that they do not require a centralized registration authority. UUIDs differ from DIDs in that they are not resolvable or cryptographically-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 § 8.2 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 § 8.1 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 § 8.1 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. This syntax feature is elaborated upon in § 3.2.1 DID Parameters.

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

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-0
> ```

> EXAMPLE 5: A unique service in a DID Document
>
> ```
> did:example:123#agent
> ```

> EXAMPLE 6: A resource external to a DID Document
>
> ```
> did:example:123?service=agent&relativeRef=/credentials#degree
> ```

> 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 described for JSON-LD representations in § E.2 application/did+ld+json. For information about how to dereference a DID fragment, see § 7.2 DID URL Dereferencing.

3.2.1 DID Parameters §

The DID URL syntax supports a simple format for parameters based on the `query` component described in § Query. Adding a DID parameter to a DID URL means that the parameter becomes part of the identifier for a resource.

> EXAMPLE 7: A DID URL with a 'versionTime' DID parameter
>
> ```
> did:example:123?versionTime=2021-05-10T17:00:00Z
> ```

> EXAMPLE 8: A DID URL with a 'service' and a 'relativeRef' DID parameter
>
> ```
> did:example:123?service=files&relativeRef=/resume.pdf
> ```

Some DID parameters are completely independent of of any specific DID method and function the same way for all DIDs. Other DID parameters are not supported by all DID methods. Where optional parameters are supported, they are expected to operate uniformly across the DID methods that do support them. The following table provides common DID parameters that function the same way across all DID methods. Support for all DID Parameters is *OPTIONAL*.

> NOTE
>
> It is generally expected that DID URL dereferencer implementations will reference [DID-RESOLUTION] for additional implementation details. The scope of this specification only defines the contract of the most common query parameters.

Parameter Name	Description

Parameter Name	Description
`service`	Identifies a service from the DID document by service ID. If present, the associated value *MUST* be an ASCII string.
`relativeRef`	A relative URI reference according to RFC3986 Section 4.2 that identifies a resource at a service endpoint, which is selected from a DID document by using the `service` parameter. If present, the associated value *MUST* be an ASCII string and *MUST* use percent-encoding for certain characters as specified in RFC3986 Section 2.1.
`versionId`	Identifies a specific version of a DID document to be resolved (the version ID could be sequential, or a UUID, or method-specific). If present, the associated value *MUST* be an ASCII string.
`versionTime`	Identifies a certain version timestamp of a DID document to be resolved. That is, the DID document that was valid for a DID at a certain time. If present, the associated value *MUST* be an ASCII string which is a valid XML datetime value, as defined in section 3.3.7 of W3C XML Schema Definition Language (XSD) 1.1 Part 2: Datatypes [XMLSCHEMA11-2]. This datetime value *MUST* be normalized to UTC 00:00:00 and without sub-second decimal precision. For example: `2020-12-20T19:17:47Z`.
`hl`	A resource hash of the DID document to add integrity protection, as specified in [HASHLINK]. This parameter is non-normative. If present, the associated value *MUST* be an ASCII string.

Implementers as well as DID method specification authors might use additional DID parameters that are not listed here. For maximum interoperability, it is *RECOMMENDED* that DID parameters use the DID Specification Registries mechanism [DID-SPEC-REGISTRIES], to avoid collision with other uses of the same DID parameter with different semantics.

DID parameters might be used if there is a clear use case where the parameter needs to be part of a URL that references a resource with more precision than using the DID alone. It is expected that DID parameters are *not* used if the same functionality can be expressed by passing input metadata to a DID resolver. Additional considerations for processing these parameters are discussed in [DID-RESOLUTION].

> NOTE: DID parameters and DID resolution
>
> The DID resolution and the DID URL dereferencing functions can be influenced by passing input metadata to a DID resolver that are not part of the DID URL (see § 7.1.1 DID Resolution Options). This is comparable to HTTP, where certain parameters could either be included in an HTTP URL, or alternatively passed as HTTP headers during the dereferencing process. The important distinction is that DID parameters that are part of the DID URL should be used to specify *what resource is being identified*, whereas input metadata that is not part of the DID URL should be use to control *how that resource is resolved or dereferenced*.

3.2.2 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 9: An example of a relative DID URL

```
{
  "@context": [
    "https://www.w3.org/ns/did/v1",
    "https://w3id.org/security/suites/ed25519-2020/v1"
  ]
  "id": "did:example:123456789abcdefghi",
  "verificationMethod": [{
    "id": "did:example:123456789abcdefghi#key-1",
    "type": "Ed25519VerificationKey2020", // external (property value)
    "controller": "did:example:123456789abcdefghi",
    "publicKeyMultibase": "zH3C2AVvLMv6gmMNam3uVAjZpfkcJCwDwnZn6z3wXmqP
  }, ...],
  "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 DID document data structures, which can then be serialized into multiple concrete representations. 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 at least two different classes of entries. The first class of entries is called properties, and is specified in section § 5. Core Properties. The second class is made up of representation-specific entries, and is specified in section § 6. Representations.

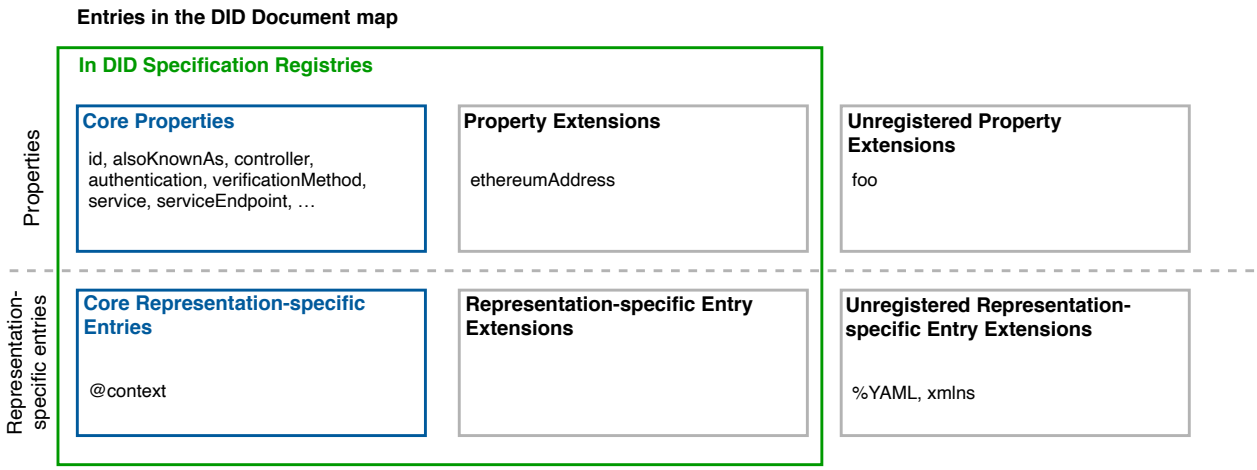

Figure 3 The entries in a DID document. See also: narrative description.

All entry keys in the DID document data model are strings. All entry values are expressed using one of the abstract 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 [INFRA]. A map is sometimes referred to as an ordered map in [INFRA].
list	A finite ordered sequence of items as specified in [INFRA].
set	A finite ordered sequence of items that does not contain the same item twice as specified in [INFRA]. A set is sometimes referred to as an ordered set in [INFRA].
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 [INFRA].
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 value 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 [INFRA].
null	A value that is used to indicate the lack of a value as defined in [INFRA].

As a result of the data model being defined using terminology from [INFRA], property values which can contain more than one item, such as lists, maps and sets, are explicitly ordered. All list-like value structures in [INFRA] are ordered, whether or not that order is significant. For the purposes of this specification, unless otherwise stated, map and set ordering 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 W3C DID Specification Registries mechanism [DID-SPEC-REGISTRIES]. 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 DID Specification Registries. 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 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 DID Specification Registries [DID-SPEC-REGISTRIES]; interoperability between such implementations and the larger ecosystem will be less reliable.

5. Core Properties §

A DID is associated with a DID document. 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.

DID Document properties §

Property	Required?	Value constraints
`id`	yes	A string that conforms to the rules in § 3.1 DID Syntax.
`alsoKnownAs`	no	A set of strings that conform to the rules of [RFC3986] for URIs.
`controller`	no	A string or a set of strings that conform to the rules in § 3.1 DID Syntax.
`verificationMethod`	no	A set of Verification Method maps that conform to the rules in § Verification Method properties.
`authentication`	no	A set of either Verification Method maps that conform to the rules in § Verification Method properties) or strings that conform to the rules in § 3.2 DID URL Syntax.
`assertionMethod`	no	
`keyAgreement`	no	
`capabilityInvocation`	no	
`capabilityDelegation`	no	
`service`	no	A set of Service Endpoint maps that conform to the rules in § Service properties.

Verification Method properties §

Property	Required?	Value constraints
`id`	yes	A string that conforms to the rules in § 3.2 DID URL Syntax.
`controller`	yes	A string that conforms to the rules in § 3.1 DID Syntax.
`type`	yes	A string.
`publicKeyJwk`	no	A map representing a JSON Web Key that conforms to [RFC7517]. See definition of publicKeyJwk for additional constraints.

Property	Required?	Value constraints
`publicKeyMultibase`	no	A string that conforms to a [MULTIBASE] encoded public key.

Service properties §

Property	Required?	Value constraints
`id`	yes	A string that conforms to the rules of [RFC3986] for URIs.
`type`	yes	A string or a set of strings.
`serviceEndpoint`	yes	A string that conforms to the rules of [RFC3986] for URIs, a map, or a set composed of a one or more strings that conform to the rules of [RFC3986] for URIs and/or maps.

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.

id

The value of `id` *MUST* be a string that conforms to the rules in § 3.1 DID Syntax and *MUST* exist in the root map of the data model for the DID document.

EXAMPLE 10

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

The `id` property only denotes the DID of the DID subject when it is present in the *topmost* map of the DID document.

NOTE: Intermediate representations

DID method specifications can create intermediate representations of a DID document that do not contain the `id` property, such as when a DID resolver is performing DID resolution. However, the fully resolved DID document always contains a valid `id` property.

5.1.2 DID Controller §

A DID controller is an entity that is authorized to make changes to a DID document. The process of authorizing a DID controller is defined by the DID method.

controller

The `controller` property is *OPTIONAL*. If present, the value *MUST* be a string or a set of strings that conform to the rules in § 3.1 DID Syntax. The corresponding DID document(s) *SHOULD* contain verification relationships that explicitly permit the use of certain verification methods for specific purposes.

When a `controller` property is present in a DID document, its value expresses one or more DIDs. Any verification methods contained in the DID documents for those DIDs *SHOULD* be accepted as authoritative, such that proofs that satisfy those verification methods are to be considered equivalent to proofs provided by the DID subject.

EXAMPLE 11: DID document with a controller property

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

NOTE: Authorization vs authentication

Note that authorization provided by the value of `controller` is separate from authentication as described in § 5.3.1 Authentication. This is particularly important for key recovery in the case of cryptographic key loss, where the DID subject no longer has access to their keys, or key compromise, where the DID controller's trusted third parties need to override malicious activity by an attacker. See § 9. Security Considerations for information related to threat models and attack vectors.

5.1.3 Also Known As §

A DID subject can have multiple identifiers for different purposes, or at different times. The assertion that two or more DIDs (or other types of URI) refer to the same DID subject can be made using the `alsoKnownAs` property.

alsoKnownAs

The `alsoKnownAs` property is *OPTIONAL*. If present, the value *MUST* be a set where each item in the set is a URI conforming to [RFC3986].

This relationship is a statement that the subject of this identifier is also identified by one or more other identifiers.

NOTE: Equivalence and alsoKnownAs

Applications might choose to consider two identifiers related by `alsoKnownAs` to be equivalent *if* the `alsoKnownAs` relationship is reciprocated in the reverse direction. It is best practice *not* to consider them equivalent in the absence of this inverse relationship. In other words, the presence of an `alsoKnownAs` assertion does not prove that this assertion is true. Therefore, it is strongly advised that a requesting party obtain independent verification of an `alsoKnownAs` assertion.

Given that the DID subject might use different identifiers for different purposes, an expectation of strong equivalence between the two identifiers, or merging the information of the two corresponding DID documents, is not necessarily appropriate, *even with* a reciprocal relationship.

5.2 Verification Methods §

A DID document can express verification methods, such as cryptographic public keys, which can be used to authenticate or authorize interactions with the DID subject or associated parties. For example, a cryptographic public key can be used as a verification method with respect to a digital signature; in such usage, it verifies that the signer could use the associated cryptographic private key. Verification methods might take many parameters. An example of this is a set of five cryptographic keys from which any three are required to contribute to a cryptographic threshold signature.

verificationMethod

The `verificationMethod` property is *OPTIONAL*. If present, the value *MUST* be a set of verification methods, where each verification method is expressed using a map. The verification method map *MUST* include the `id`, `type`, `controller`, and specific verification material properties that are determined by the value of `type` and are defined in § 5.2.1 Verification Material. A verification method *MAY* include additional properties. Verification

methods *SHOULD* be registered in the DID Specification Registries [DID-SPEC-REGISTRIES].

id

The value of the `id` property for a verification method *MUST* be a string that conforms to the rules in Section § 3.2 DID URL Syntax.

type

The value of the `type` property *MUST* be a string that references exactly one verification method type. In order to maximize global interoperability, the verification method type *SHOULD* be registered in the DID Specification Registries [DID-SPEC-REGISTRIES].

controller

The value of the `controller` property *MUST* be a string that conforms to the rules in § 3.1 DID Syntax.

EXAMPLE 12: Example verification method structure

```
{
  "@context": [
    "https://www.w3.org/ns/did/v1",
    "https://w3id.org/security/suites/jws-2020/v1"
    "https://w3id.org/security/suites/ed25519-2020/v1"
  ]
  "id": "did:example:123456789abcdefghi",
  ...
  "verificationMethod": [{
    "id": ...,
    "type": ...,
    "controller": ...,
    "publicKeyJwk": ...
  }, {
    "id": ...,
    "type": ...,
    "controller": ...,
    "publicKeyMultibase": ...
  }]
}
```

> NOTE: Verification method controller(s) and DID controller(s)
>
> The semantics of the `controller` property are the same when the subject of the relationship is the DID document as when the subject of the relationship is a verification method, such as a cryptographic public key. Since a key can't control itself, and the key controller cannot be inferred from the DID document, it is necessary to explicitly express the identity of the controller of the key. The difference is that the value of `controller` for a verification method is *not* necessarily a DID controller. DID controllers are expressed using the `controller` property at the highest level of the DID document (the topmost map in the data model); see § 5.1.2 DID Controller.

5.2.1 Verification Material §

Verification material is any information that is used by a process that applies a verification method. The `type` of a verification method is expected to be used to determine its compatibility with such processes. Examples of verification material properties are `publicKeyJwk` or `publicKeyMultibase`. A cryptographic suite specification is responsible for specifying the verification method `type` and its associated verification material. For example, see JSON Web Signature 2020 and Ed25519 Signature 2020. For all registered verification method types and associated verification material available for DIDs, please see the DID Specification Registries [DID-SPEC-REGISTRIES].

To increase the likelihood of interoperable implementations, this specification limits the number of formats for expressing verification material in a DID document. The fewer formats that implementers have to implement, the more likely it will be that they will support all of them. This approach attempts to strike a delicate balance between ease of implementation and supporting formats that have historically had broad deployment. Two supported verification material properties are listed below:

publicKeyJwk
: The `publicKeyJwk` property is *OPTIONAL*. If present, the value *MUST* be a map representing a JSON Web Key that conforms to [RFC7517]. The map *MUST NOT* contain "d", or any other members of the private information class as described in Registration Template. It is *RECOMMENDED* that verification methods that use JWKs [RFC7517] to represent their public keys use the value of `kid` as their fragment identifier. It is *RECOMMENDED* that JWK `kid` values are set to the public key fingerprint [RFC7638]. See the first key in Example 13 for an example of a public key with a compound key identifier.

publicKeyMultibase
: The `publicKeyMultibase` property is *OPTIONAL*. This feature is non-normative. If present, the value *MUST* be a string representation of a [MULTIBASE] encoded public key.

> Note that the [MULTIBASE] specification is not yet a standard and is subject to change. There might be some use cases for this data format where **`public`**`KeyMultibase` is defined, to allow for expression of public keys, but **`private`**`KeyMultibase` is not defined, to protect against accidental leakage of secret keys.

A verification method *MUST NOT* contain multiple verification material properties for the same material. For example, expressing key material in a verification method using both `publicKeyJwk` and `publicKeyMultibase` at the same time is prohibited.

An example of a DID document containing verification methods using both properties above is shown below.

EXAMPLE 13: Verification methods using publicKeyJwk and publicKeyMultibase

```
{
  "@context": [
    "https://www.w3.org/ns/did/v1",
    "https://w3id.org/security/suites/jws-2020/v1",
    "https://w3id.org/security/suites/ed25519-2020/v1"
  ]
  "id": "did:example:123456789abcdefghi",
  ...
  "verificationMethod": [{
    "id": "did:example:123#_Qq0UL2Fq651Q0Fjd6TvnYE-faHiOpRlPVQcY_-tA4A"
    "type": "JsonWebKey2020", // external (property value)
    "controller": "did:example:123",
    "publicKeyJwk": {
      "crv": "Ed25519", // external (property name)
      "x": "VCpo2LMLhn6iWku8MKvSLg2ZAoC-nlOyPVQaO3FxVeQ", // external (
      "kty": "OKP", // external (property name)
      "kid": "_Qq0UL2Fq651Q0Fjd6TvnYE-faHiOpRlPVQcY_-tA4A" // external
    }
  }, {
    "id": "did:example:123456789abcdefghi#keys-1",
    "type": "Ed25519VerificationKey2020", // external (property value)
    "controller": "did:example:pqrstuvwxyz0987654321",
    "publicKeyMultibase": "zH3C2AVvLMv6gmMNam3uVAjZpfkcJCwDwnZn6z3wXmqP
  }],
  ...
}
```

5.2.2 Referring to Verification Methods §

Verification methods can be embedded in or referenced from properties associated with various verification relationships as described in § 5.3 Verification Relationships. Referencing verification methods allows them to be used by more than one verification relationship.

If the value of a verification method property is a map, the verification method has been embedded and its properties can be accessed directly. However, if the value is a URL string, the verification method has been included by reference and its properties will need to be retrieved from elsewhere in the DID document or from another DID document. This is done by dereferencing the URL and searching the resulting resource for a verification method map with an `id` property whose value matches the URL.

EXAMPLE 14: Embedding and referencing verification methods

```
{
...

  "authentication": [
    // this key is referenced and might be used by
    // more than one verification relationship
    "did:example:123456789abcdefghi#keys-1",
    // this key is embedded and may *only* be used for authentication
    {
      "id": "did:example:123456789abcdefghi#keys-2",
      "type": "Ed25519VerificationKey2020", // external (property value
      "controller": "did:example:123456789abcdefghi",
      "publicKeyMultibase": "zH3C2AVvLMv6gmMNam3uVAjZpfkcJCwDwnZn6z3wXm
    }
  ],

...
}
```

5.3 Verification Relationships §

A verification relationship expresses the relationship between the DID subject and a verification method.

Different verification relationships enable the associated verification methods to be used for different purposes. It is up to a *verifier* to ascertain the validity of a verification attempt by checking that the verification method used is contained in the appropriate verification relationship property of the DID Document.

The verification relationship between the DID subject and the verification method is explicit in the DID document. Verification methods that are not associated with a particular verification relationship cannot be used for that verification relationship. For example, a verification method in the value of the `authentication` property cannot be used to engage in key agreement protocols with the DID subject—the value of the `keyAgreement` property needs to be used for that.

The DID document does not express revoked keys using a verification relationship. If a referenced verification method is not in the latest DID Document used to dereference it, then that verification method is considered invalid or revoked. Each DID method specification is expected to detail how revocation is performed and tracked.

The following sections define several useful verification relationships. A DID document *MAY* include any of these, or other properties, to express a specific verification relationship. In order to maximize global interoperability, any such properties used *SHOULD* be registered in the DID Specification Registries [DID-SPEC-REGISTRIES].

5.3.1 Authentication §

The `authentication` verification relationship is used to specify how the DID subject is expected to be authenticated, for purposes such as logging into a website or engaging in any sort of challenge-response protocol.

authentication

The `authentication` property is *OPTIONAL*. If present, the associated value *MUST* be a set of one or more verification methods. Each verification method *MAY* be embedded or referenced.

EXAMPLE 15: Authentication property containing three verification methods

```
{
  "@context": [
    "https://www.w3.org/ns/did/v1",
    "https://w3id.org/security/suites/ed25519-2020/v1"
  ],
  "id": "did:example:123456789abcdefghi",
  ...
  "authentication": [
    // this method can be used to authenticate as did:...fghi
    "did:example:123456789abcdefghi#keys-1",
    // this method is *only* approved for authentication, it may not
    // be used for any other proof purpose, so its full description is
    // embedded here rather than using only a reference
    {
      "id": "did:example:123456789abcdefghi#keys-2",
      "type": "Ed25519VerificationKey2020",
      "controller": "did:example:123456789abcdefghi",
      "publicKeyMultibase": "zH3C2AVvLMv6gmMNam3uVAjZpfkcJCwDwnZn6z3wXm
    }
  ],
  ...
}
```

If authentication is established, it is up to the DID method or other application to decide what to do with that information. A particular DID method could decide that authenticating as a DID controller is sufficient to, for example, update or delete the DID document. Another DID method could require different keys, or a different verification method entirely, to be presented in order to update or delete the DID document than that used to authenticate. In other words, what is done *after* the authentication check is out of scope for the data model; DID methods and applications are expected to define this themselves.

This is useful to any *authentication verifier* that needs to check to see if an entity that is attempting to authenticate is, in fact, presenting a valid proof of authentication. When a *verifier* receives some data (in some protocol-specific format) that contains a proof that was made for the purpose of "authentication", and that says that an entity is identified by the DID, then that *verifier* checks to ensure that the proof can be verified using a verification method (e.g., public key) listed under `authentication` in the DID Document.

Note that the verification method indicated by the `authentication` property of a DID document can only be used to authenticate the DID subject. To authenticate a different DID controller, the entity associated with the value of `controller`, as defined in § 5.1.2 DID Controller, needs to

authenticate with its *own* DID document and associated `authentication` verification relationship.

5.3.2 Assertion §

The `assertionMethod` verification relationship is used to specify how the DID subject is expected to express claims, such as for the purposes of issuing a Verifiable Credential [VC-DATA-MODEL].

assertionMethod

The `assertionMethod` property is *OPTIONAL*. If present, the associated value *MUST* be a set of one or more verification methods. Each verification method *MAY* be embedded or referenced.

This property is useful, for example, during the processing of a verifiable credential by a verifier. During verification, a verifier checks to see if a verifiable credential contains a proof created by the DID subject by checking that the verification method used to assert the proof is associated with the `assertionMethod` property in the corresponding DID document.

EXAMPLE 16: Assertion method property containing two verification methods

```
{
  "@context": [
    "https://www.w3.org/ns/did/v1",
    "https://w3id.org/security/suites/ed25519-2020/v1"
  ],
  "id": "did:example:123456789abcdefghi",
  ...
  "assertionMethod": [
    // this method can be used to assert statements as did:...fghi
    "did:example:123456789abcdefghi#keys-1",
    // this method is *only* approved for assertion of statements, it i
    // used for any other verification relationship, so its full descri
    // embedded here rather than using a reference
    {
      "id": "did:example:123456789abcdefghi#keys-2",
      "type": "Ed25519VerificationKey2020", // external (property value
      "controller": "did:example:123456789abcdefghi",
      "publicKeyMultibase": "zH3C2AVvLMv6gmMNam3uVAjZpfkcJCwDwnZn6z3wXm
    }
  ],
  ...
}
```

5.3.3 Key Agreement §

The `keyAgreement` verification relationship is used to specify how an entity can generate encryption material in order to transmit confidential information intended for the DID subject, such as for the purposes of establishing a secure communication channel with the recipient.

keyAgreement

> The `keyAgreement` property is *OPTIONAL*. If present, the associated value *MUST* be a set of one or more verification methods. Each verification method *MAY* be embedded or referenced.

An example of when this property is useful is when encrypting a message intended for the DID subject. In this case, the counterparty uses the cryptographic public key information in the verification method to wrap a decryption key for the recipient.

EXAMPLE 17: Key agreement property containing two verification methods

```
{
  "@context": "https://www.w3.org/ns/did/v1",
  "id": "did:example:123456789abcdefghi",
  ...
  "keyAgreement": [
    // this method can be used to perform key agreement as did:...fghi
    "did:example:123456789abcdefghi#keys-1",
    // this method is *only* approved for key agreement usage, it will
    // be used for any other verification relationship, so its full des
    // embedded here rather than using only a reference
    {
      "id": "did:example:123#zC9ByQ8aJs8vrNXyDhPHHNNMSHPcaSgNpjjsBYpMMj
      "type": "X25519KeyAgreementKey2019", // external (property value)
      "controller": "did:example:123",
      "publicKeyMultibase": "z9hFgmPVfmBZwRvFEyniQDBkz9LmV7gDEqytWyGZLm
    }
  ],
  ...
}
```

5.3.4 Capability Invocation §

The `capabilityInvocation` verification relationship is used to specify a verification method that might be used by the DID subject to invoke a cryptographic capability, such as the authorization to update the DID Document.

capabilityInvocation

> The `capabilityInvocation` property is *OPTIONAL*. If present, the associated value *MUST* be a set of one or more verification methods. Each verification method *MAY* be embedded or referenced.

An example of when this property is useful is when a DID subject needs to access a protected HTTP API that requires authorization in order to use it. In order to authorize when using the HTTP API, the DID subject uses a capability that is associated with a particular URL that is exposed via the HTTP API. The invocation of the capability could be expressed in a number of ways, e.g., as a digitally signed message that is placed into the HTTP Headers.

The server providing the HTTP API is the *verifier* of the capability and it would need to verify that the verification method referred to by the invoked capability exists in the `capabilityInvocation` property of the DID document. The verifier would also check to make sure that the action being performed is valid and the capability is appropriate for the resource being accessed. If the verification is successful, the server has cryptographically determined that the invoker is authorized to access the protected resource.

EXAMPLE 18: Capability invocation property containing two verification methods

```
{
  "@context": [
    "https://www.w3.org/ns/did/v1",
    "https://w3id.org/security/suites/ed25519-2020/v1"
  ],
  "id": "did:example:123456789abcdefghi",
  ...
  "capabilityInvocation": [
    // this method can be used to invoke capabilities as did:...fghi
    "did:example:123456789abcdefghi#keys-1",
    // this method is *only* approved for capability invocation usage,
    // be used for any other verification relationship, so its full des
    // embedded here rather than using only a reference
    {
    "id": "did:example:123456789abcdefghi#keys-2",
    "type": "Ed25519VerificationKey2020", // external (property value)
    "controller": "did:example:123456789abcdefghi",
    "publicKeyMultibase": "zH3C2AVvLMv6gmMNam3uVAjZpfkcJCwDwnZn6z3wXmqP
    }
  ],
  ...
}
```

5.3.5 Capability Delegation §

The `capabilityDelegation` verification relationship is used to specify a mechanism that might be used by the DID subject to delegate a cryptographic capability to another party, such as delegating the authority to access a specific HTTP API to a subordinate.

capabilityDelegation

The `capabilityDelegation` property is *OPTIONAL*. If present, the associated value *MUST* be a set of one or more verification methods. Each verification method *MAY* be embedded or referenced.

An example of when this property is useful is when a DID controller chooses to delegate their capability to access a protected HTTP API to a party other than themselves. In order to delegate the capability, the DID subject would use a verification method associated with the `capabilityDelegation` verification relationship to cryptographically sign the capability over to another DID subject. The delegate would then use the capability in a manner that is similar to the example described in § 5.3.4 Capability Invocation.

EXAMPLE 19: Capability Delegation property containing two verification methods

```
{
  "@context": [
    "https://www.w3.org/ns/did/v1",
    "https://w3id.org/security/suites/ed25519-2020/v1"
  ],
  "id": "did:example:123456789abcdefghi",
  ...
  "capabilityDelegation": [
    // this method can be used to perform capability delegation as did:
    "did:example:123456789abcdefghi#keys-1",
    // this method is *only* approved for granting capabilities; it wil
    // be used for any other verification relationship, so its full des
    // embedded here rather than using only a reference
    {
    "id": "did:example:123456789abcdefghi#keys-2",
    "type": "Ed25519VerificationKey2020", // external (property value)
    "controller": "did:example:123456789abcdefghi",
    "publicKeyMultibase": "zH3C2AVvLMv6gmMNam3uVAjZpfkcJCwDwnZn6z3wXmqP
    }
  ],
  ...
}
```

5.4 Services §

Services are used in DID documents to express ways of communicating with the DID subject or associated entities. A service can be any type of service the DID subject wants to advertise, including decentralized identity management services for further discovery, authentication, authorization, or interaction.

Due to privacy concerns, revealing public information through services, such as social media accounts, personal websites, and email addresses, is discouraged. Further exploration of privacy concerns can be found in § 10.1 Keep Personal Data Private and § 10.6 Service Privacy. The information associated with services is often service specific. For example, the information associated with an encrypted messaging service can express how to initiate the encrypted link before messaging begins.

Services are expressed using the `service` property, which is described below:

service

The `service` property is *OPTIONAL*. If present, the associated value *MUST* be a set of services, where each service is described by a map. Each service map *MUST* contain `id`, `type`, and `serviceEndpoint` properties. Each service extension *MAY* include additional properties and *MAY* further restrict the properties associated with the extension.

id

The value of the `id` property *MUST* be a URI conforming to [RFC3986]. A conforming producer *MUST NOT* produce multiple `service` entries with the same `id`. A conforming consumer *MUST* produce an error if it detects multiple `service` entries with the same `id`.

type

The value of the `type` property *MUST* be a string or a set of strings. In order to maximize interoperability, the service type and its associated properties *SHOULD* be registered in the DID Specification Registries [DID-SPEC-REGISTRIES].

serviceEndpoint

The value of the `serviceEndpoint` property *MUST* be a string, a map, or a set composed of one or more strings and/or maps. All string values *MUST* be valid URIs conforming to [RFC3986] and normalized according to the Normalization and Comparison rules in RFC3986 and to any normalization rules in its applicable URI scheme specification.

For more information regarding privacy and security considerations related to services see § 10.6 Service Privacy, § 10.1 Keep Personal Data Private, § 10.3 DID Document Correlation Risks, and § 9.3 Authentication Service Endpoints.

EXAMPLE 20: Usage of the service property

```
{
  "service": [{
    "id":"did:example:123#linked-domain",
    "type": "LinkedDomains", // external (property value)
    "serviceEndpoint": "https://bar.example.com"
  }]
}
```

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 representations for JSON and JSON-LD, and developers can use any other representation, such as XML or YAML, that is capable of expressing the data model. The following sections define the general rules for production and consumption, as well as the JSON and JSON-LD representations.

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 DID Specification Registries [DID-SPEC-REGISTRIES]). 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 DID Specification Registries [DID-SPEC-REGISTRIES].

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 DID Specification Registries [DID-SPEC-REGISTRIES].
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.

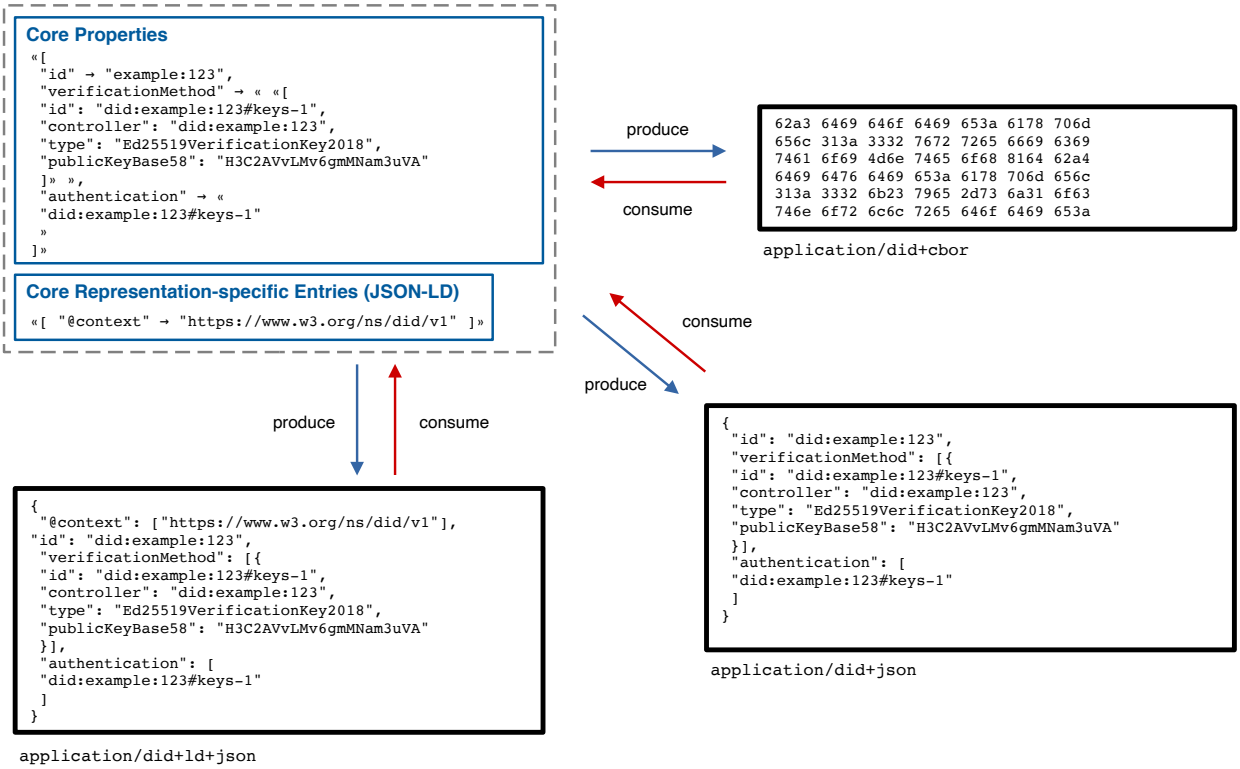

Figure 4 Production and consumption of representations. See also: narrative description.

> 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.

Data Type	JSON Representation Type
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`.
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+json` to downstream applications such as described in § 7.1.2 DID Resolution Metadata.

EXAMPLE 21: Example DID document in JSON representation

```
{
  "id": "did:example:123456789abcdefghi",
  "authentication": [{
    "id": "did:example:123456789abcdefghi#keys-1",
    "type": "Ed25519VerificationKey2018",
    "controller": "did:example:123456789abcdefghi",
    "publicKeyBase58": "H3C2AVvLMv6gmMNam3uVAjZpfkcJCwDwnZn6z3wXmqPV"
  }]
}
```

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 Boolean	A boolean.
JSON null literal	A null value.

> All implementers creating conforming consumers that produce JSON representations 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+json`, 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.

6.3 JSON-LD §

JSON-LD [JSON-LD11] is a JSON-based format used to serialize Linked Data. This section defines the production and consumption rules for the JSON-LD representation.

The JSON-LD representation defines the following representation-specific entries:

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

6.3.1 Production §

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

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

EXAMPLE 22: A valid serialization of a simple @context entry

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

EXAMPLE 23: A valid serialization of a layered @context entry

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

All implementers creating conforming producers that produce JSON-LD representations are advised to ensure that their algorithms produce valid JSON-LD [JSON-LD11] documents. Invalid JSON-LD documents will cause JSON-LD processors to halt and report errors.

In order to achieve interoperability across different representations, all JSON-LD Contexts and their terms *SHOULD* be registered in the DID Specification Registries [DID-SPEC-REGISTRIES].

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+ld+json` to downstream applications such as described in § 7.1.2 DID Resolution Metadata.

6.3.2 Consumption §

The DID document and any DID document data structures expressed by a JSON-LD representation *MUST* be deserialized into the data model according to the JSON representation consumption rules as defined in § 6.2 JSON.

All implementers creating conforming consumers that consume JSON-LD representations are advised to ensure that their algorithms only accept valid JSON-LD [JSON-LD11] documents. Invalid JSON-LD documents will cause JSON-LD processors to halt and report errors.

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

7. Resolution §

This section defines the inputs and outputs of DID resolution and DID URL dereferencing. Their exact implementation is out of scope for this specification, but some considerations for

implementers are discussed in [DID-RESOLUTION].

All conformant DID resolvers *MUST* implement the DID resolution functions for at least one DID method and *MUST* be able to return a DID document in at least one conformant representation.

7.1 DID Resolution §

The DID resolution functions resolve a DID into a DID document by using the "Read" operation of the applicable DID method as described in § 8.2 Method Operations. The details of how this process is accomplished are outside the scope of this specification, but all conforming DID resolvers implement the functions below, which have the following abstract forms:

```
resolve(did, resolutionOptions) →
   « didResolutionMetadata, didDocument, didDocumentMetadata »

resolveRepresentation(did, resolutionOptions) →
   « didResolutionMetadata, didDocumentStream, didDocumentMetadata »
```

The `resolve` function returns the DID document in its abstract form (a map). The `resolveRepresentation` function returns a byte stream of the DID Document formatted in the corresponding representation.

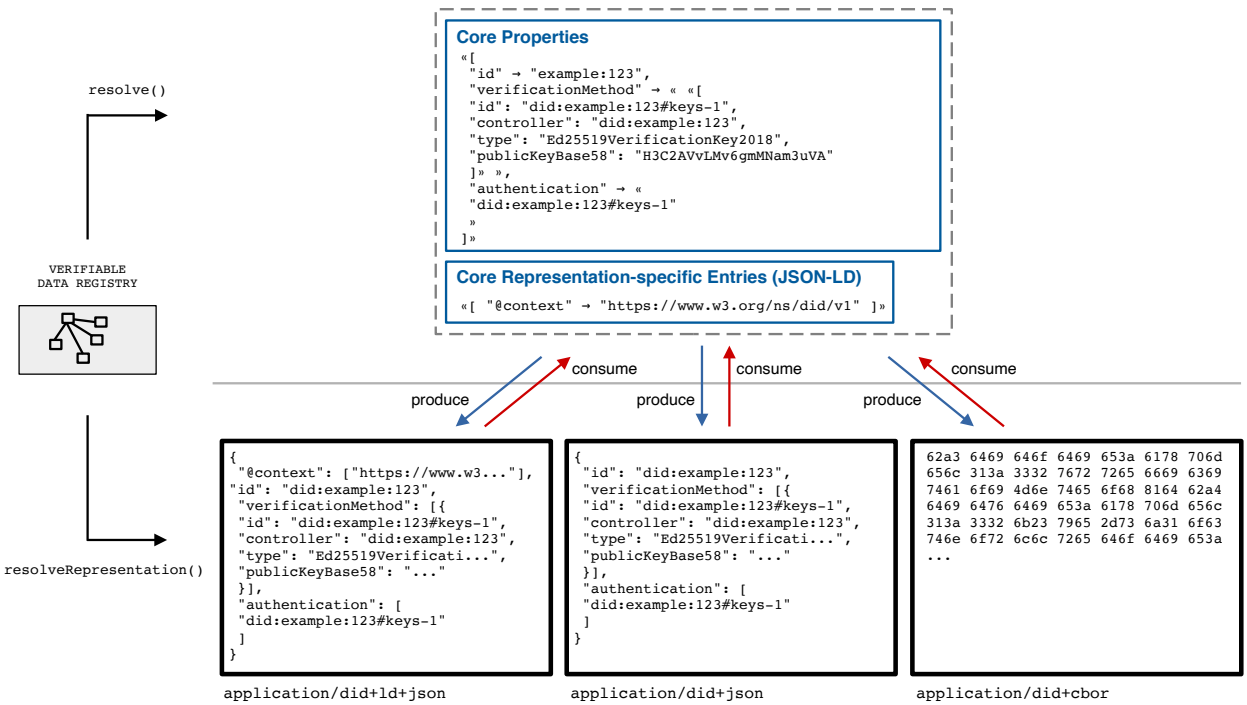

Figure 5 Functions resolve() and resolveRepresentation(). See also: narrative description.

The input variables of the `resolve` and `resolveRepresentation` functions are as follows:

did

This is the DID to resolve. This input is *REQUIRED* and the value *MUST* be a conformant DID as defined in § 3.1 DID Syntax.

resolutionOptions

A metadata structure containing properties defined in § 7.1.1 DID Resolution Options. This input is *REQUIRED*, but the structure *MAY* be empty.

These functions each return multiple values, and no limitations are placed on how these values are returned together. The return values of `resolve` are didResolutionMetadata, didDocument, and didDocumentMetadata. The return values of `resolveRepresentation` are didResolutionMetadata, didDocumentStream, and didDocumentMetadata. These values are described below:

didResolutionMetadata

A metadata structure consisting of values relating to the results of the DID resolution process which typically changes between invocations of the `resolve` and `resolveRepresentation` functions, as it represents data about the resolution process itself. This structure is *REQUIRED*, and in the case of an error in the resolution process, this *MUST NOT* be empty. This metadata is defined by § 7.1.2 DID Resolution Metadata. If `resolveRepresentation` was called, this structure *MUST* contain a `contentType` property containing the Media Type of the representation found in the `didDocumentStream`. If the resolution is not successful, this structure *MUST* contain an `error` property describing the error.

didDocument

If the resolution is successful, and if the `resolve` function was called, this *MUST* be a DID document abstract data model (a map) as described in § 4. Data Model that is capable of being transformed into a conforming DID Document (representation), using the production rules specified by the representation. The value of `id` in the resolved DID document *MUST* match the DID that was resolved. If the resolution is unsuccessful, this value *MUST* be empty.

didDocumentStream

If the resolution is successful, and if the `resolveRepresentation` function was called, this *MUST* be a byte stream of the resolved DID document in one of the conformant representations. The byte stream might then be parsed by the caller of the `resolveRepresentation` function into a data model, which can in turn be validated and processed. If the resolution is unsuccessful, this value *MUST* be an empty stream.

didDocumentMetadata

If the resolution is successful, this *MUST* be a metadata structure. This structure contains metadata about the DID document contained in the `didDocument` property. This metadata typically does not change between invocations of the `resolve` and `resolveRepresentation` functions unless the DID document changes, as it represents metadata about the DID document. If the resolution is unsuccessful, this output *MUST* be an

empty metadata structure. Properties defined by this specification are in § 7.1.3 DID Document Metadata.

Conforming DID resolver implementations do not alter the signature of these functions in any way. DID resolver implementations might map the `resolve` and `resolveRepresentation` functions to a method-specific internal function to perform the actual DID resolution process. DID resolver implementations might implement and expose additional functions with different signatures in addition to the `resolve` and `resolveRepresentation` functions specified here.

7.1.1 DID Resolution Options §

The possible properties within this structure and their possible values are registered in the DID Specification Registries [DID-SPEC-REGISTRIES]. This specification defines the following common properties.

accept
: The Media Type of the caller's preferred representation of the DID document. The Media Type *MUST* be expressed as an ASCII string. The DID resolver implementation *SHOULD* use this value to determine the representation contained in the returned `didDocumentStream` if such a representation is supported and available. This property is *OPTIONAL* for the `resolveRepresentation` function and *MUST NOT* be used with the `resolve` function.

7.1.2 DID Resolution Metadata §

The possible properties within this structure and their possible values are registered in the DID Specification Registries [DID-SPEC-REGISTRIES]. This specification defines the following DID resolution metadata properties:

contentType
: The Media Type of the returned `didDocumentStream`. This property is *REQUIRED* if resolution is successful and if the `resolveRepresentation` function was called. This property *MUST NOT* be present if the `resolve` function was called. The value of this property *MUST* be an ASCII string that is the Media Type of the conformant representations. The caller of the `resolveRepresentation` function *MUST* use this value when determining how to parse and process the `didDocumentStream` returned by this function into the data model.

error
: The error code from the resolution process. This property is *REQUIRED* when there is an error in the resolution process. The value of this property *MUST* be a single keyword ASCII string. The possible property values of this field *SHOULD* be registered in the DID

Specification Registries [DID-SPEC-REGISTRIES]. This specification defines the following common error values:

invalidDid
: The DID supplied to the DID resolution function does not conform to valid syntax. (See § 3.1 DID Syntax.)

notFound
: The DID resolver was unable to find the DID document resulting from this resolution request.

representationNotSupported
: This error code is returned if the representation requested via the `accept` input metadata property is not supported by the DID method and/or DID resolver implementation.

7.1.3 DID Document Metadata §

The possible properties within this structure and their possible values *SHOULD* be registered in the DID Specification Registries [DID-SPEC-REGISTRIES]. This specification defines the following common properties.

created
: DID document metadata *SHOULD* include a `created` property to indicate the timestamp of the Create operation. The value of the property *MUST* be a string formatted as an XML Datetime normalized to UTC 00:00:00 and without sub-second decimal precision. For example: `2020-12-20T19:17:47Z`.

updated
: DID document metadata *SHOULD* include an `updated` property to indicate the timestamp of the last Update operation for the document version which was resolved. The value of the property *MUST* follow the same formatting rules as the `created` property. The `updated` property is omitted if an Update operation has never been performed on the DID document. If an `updated` property exists, it can be the same value as the `created` property when the difference between the two timestamps is less than one second.

deactivated
: If a DID has been deactivated, DID document metadata *MUST* include this property with the boolean value `true`. If a DID has not been deactivated, this property is *OPTIONAL*, but if included, *MUST* have the boolean value `false`.

nextUpdate
: DID document metadata *MAY* include a `nextUpdate` property if the resolved document version is not the latest version of the document. It indicates the timestamp of the next Update operation. The value of the property *MUST* follow the same formatting rules as the `created` property.

versionId

DID document metadata *SHOULD* include a `versionId` property to indicate the version of the last Update operation for the document version which was resolved. The value of the property *MUST* be an ASCII string.

nextVersionId

DID document metadata *MAY* include a `nextVersionId` property if the resolved document version is not the latest version of the document. It indicates the version of the next Update operation. The value of the property *MUST* be an ASCII string.

equivalentId

A DID method can define different forms of a DID that are logically equivalent. An example is when a DID takes one form prior to registration in a verifiable data registry and another form after such registration. In this case, the DID method specification might need to express one or more DIDs that are logically equivalent to the resolved DID as a property of the DID document. This is the purpose of the `equivalentId` property.

DID document metadata *MAY* include an `equivalentId` property. If present, the value *MUST* be a set where each item is a string that conforms to the rules in Section § 3.1 DID Syntax. The relationship is a statement that each `equivalentId` value is logically equivalent to the `id` property value and thus refers to the same DID subject. Each `equivalentId` DID value *MUST* be produced by, and a form of, the same DID method as the `id` property value. (e.g., `did:example:abc` == `did:example:ABC`)

A conforming DID method specification *MUST* guarantee that each `equivalentId` value is logically equivalent to the `id` property value.

A requesting party is expected to retain the values from the `id` and `equivalentId` properties to ensure any subsequent interactions with any of the values they contain are correctly handled as logically equivalent (e.g., retain all variants in a database so an interaction with any one maps to the same underlying account).

> NOTE: Stronger equivalence
>
> `equivalentId` is a much stronger form of equivalence than `alsoKnownAs` because the equivalence *MUST* be guaranteed by the governing DID method. `equivalentId` represents a full graph merge because the same DID document describes both the `equivalentId` DID and the `id` property DID.

If a requesting party does not retain the values from the `id` and `equivalentId` properties and ensure any subsequent interactions with any of the values they contain are correctly handled as logically equivalent, there might be negative or unexpected issues that arise. Implementers are strongly advised to observe the directives related to this metadata property.

canonicalId

The `canonicalId` property is identical to the `equivalentId` property except: a) it is associated with a single value rather than a set, and b) the DID is defined to be the canonical ID for the DID subject within the scope of the containing DID document.

DID document metadata *MAY* include a `canonicalId` property. If present, the value *MUST* be a string that conforms to the rules in Section § 3.1 DID Syntax. The relationship is a statement that the `canonicalId` value is logically equivalent to the `id` property value and that the `canonicalId` value is defined by the DID method to be the canonical ID for the DID subject in the scope of the containing DID document. A `canonicalId` value *MUST* be produced by, and a form of, the same DID method as the `id` property value. (e.g., `did:example:abc` == `did:example:ABC`).

A conforming DID method specification *MUST* guarantee that the `canonicalId` value is logically equivalent to the `id` property value.

A requesting party is expected to use the `canonicalId` value as its primary ID value for the DID subject and treat all other equivalent values as secondary aliases (e.g., update corresponding primary references in their systems to reflect the new canonical ID directive).

> NOTE: Canonical equivalence
>
> `canonicalId` is the same statement of equivalence as `equivalentId` except it is constrained to a single value that is defined to be canonical for the DID subject in the scope of the DID document. Like `equivalentId`, `canonicalId` represents a full graph merge because the same DID document describes both the `canonicalId` DID and the `id` property DID.

If a resolving party does not use the `canonicalId` value as its primary ID value for the DID subject and treat all other equivalent values as secondary aliases, there might be negative or unexpected issues that arise related to user experience. Implementers are strongly advised to observe the directives related to this metadata property.

7.2 DID URL Dereferencing §

The DID URL dereferencing function dereferences a DID URL into a resource with contents depending on the DID URL's components, including the DID method, method-specific identifier, path, query, and fragment. This process depends on DID resolution of the DID contained in the DID URL. DID URL dereferencing might involve multiple steps (e.g., when the DID URL being dereferenced includes a fragment), and the function is defined to return the final resource after all steps are completed. The details of how this process is accomplished are outside the scope of this specification. The following figure depicts the relationship described above.

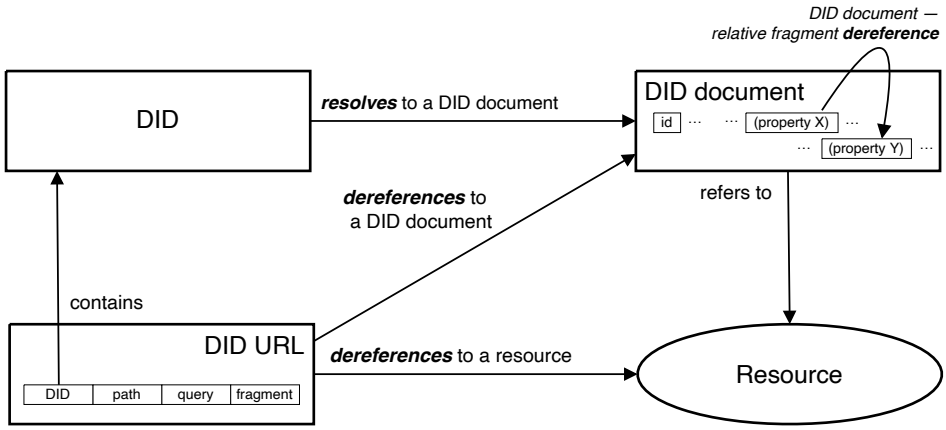

Figure 6 Overview of DID URL dereference See also: narrative description.

All conforming DID resolvers implement the following function which has the following abstract form:

```
dereference(didUrl, dereferenceOptions) →
   « dereferencingMetadata, contentStream, contentMetadata »
```

The input variables of the `dereference` function are as follows:

didUrl

A conformant DID URL as a single string. This is the DID URL to dereference. To dereference a DID fragment, the complete DID URL including the DID fragment *MUST* be used. This input is *REQUIRED*.

> NOTE: DID URL dereferencer patterns
>
> While it is valid for any `didUrl` to be passed to a DID URL dereferencer, implementers are expected to refer to [DID-RESOLUTION] to further understand common patterns for how a DID URL is expected to be dereferenced.

dereferencingOptions

A metadata structure consisting of input options to the `dereference` function in addition to the `didUrl` itself. Properties defined by this specification are in § 7.2.1 DID URL Dereferencing Options. This input is *REQUIRED*, but the structure *MAY* be empty.

This function returns multiple values, and no limitations are placed on how these values are returned together. The return values of the `dereference` include `dereferencingMetadata`, `contentStream`, and `contentMetadata`:

dereferencingMetadata

A metadata structure consisting of values relating to the results of the DID URL dereferencing process. This structure is *REQUIRED*, and in the case of an error in the dereferencing process, this *MUST NOT* be empty. Properties defined by this specification are in § 7.2.2 DID URL

Dereferencing Metadata. If the dereferencing is not successful, this structure *MUST* contain an `error` property describing the error.

contentStream
: If the `dereferencing` function was called and successful, this *MUST* contain a resource corresponding to the DID URL. The `contentStream` *MAY* be a resource such as a DID document that is serializable in one of the conformant representations, a Verification Method, a service, or any other resource format that can be identified via a Media Type and obtained through the resolution process. If the dereferencing is unsuccessful, this value *MUST* be empty.

contentMetadata
: If the dereferencing is successful, this *MUST* be a metadata structure, but the structure *MAY* be empty. This structure contains metadata about the `contentStream`. If the `contentStream` is a DID document, this *MUST* be a didDocumentMetadata structure as described in DID Resolution. If the dereferencing is unsuccessful, this output *MUST* be an empty metadata structure.

Conforming DID URL dereferencing implementations do not alter the signature of these functions in any way. DID URL dereferencing implementations might map the `dereference` function to a method-specific internal function to perform the actual DID URL dereferencing process. DID URL dereferencing implementations might implement and expose additional functions with different signatures in addition to the `dereference` function specified here.

7.2.1 DID URL Dereferencing Options §

The possible properties within this structure and their possible values *SHOULD* be registered in the DID Specification Registries [DID-SPEC-REGISTRIES]. This specification defines the following common properties for dereferencing options:

accept
: The Media Type that the caller prefers for `contentStream`. The Media Type *MUST* be expressed as an ASCII string. The DID URL dereferencing implementation *SHOULD* use this value to determine the `contentType` of the representation contained in the returned value if such a representation is supported and available.

7.2.2 DID URL Dereferencing Metadata §

The possible properties within this structure and their possible values are registered in the DID Specification Registries [DID-SPEC-REGISTRIES]. This specification defines the following common properties.

contentType

The Media Type of the returned `contentStream` *SHOULD* be expressed using this property if dereferencing is successful. The Media Type value *MUST* be expressed as an ASCII string.

error

The error code from the dereferencing process. This property is *REQUIRED* when there is an error in the dereferencing process. The value of this property *MUST* be a single keyword expressed as an ASCII string. The possible property values of this field *SHOULD* be registered in the DID Specification Registries [DID-SPEC-REGISTRIES]. This specification defines the following common error values:

invalidDidUrl

The DID URL supplied to the DID URL dereferencing function does not conform to valid syntax. (See § 3.2 DID URL Syntax.)

notFound

The DID URL dereferencer was unable to find the `contentStream` resulting from this dereferencing request.

7.3 Metadata Structure §

Input and output metadata is often involved during the DID Resolution, DID URL dereferencing, and other DID-related processes. The structure used to communicate this metadata *MUST* be a map of properties. Each property name *MUST* be a string. Each property value *MUST* be a string, map, list, set, boolean, or null. The values within any complex data structures such as maps and lists *MUST* be one of these data types as well. All metadata property definitions registered in the DID Specification Registries [DID-SPEC-REGISTRIES] *MUST* define the value type, including any additional formats or restrictions to that value (for example, a string formatted as a date or as a decimal integer). It is *RECOMMENDED* that property definitions use strings for values. The entire metadata structure *MUST* be serializable according to the JSON serialization rules in the [INFRA] specification. Implementations *MAY* serialize the metadata structure to other data formats.

All implementations of functions that use metadata structures as either input or output are able to fully represent all data types described here in a deterministic fashion. As inputs and outputs using metadata structures are defined in terms of data types and not their serialization, the method for representation is internal to the implementation of the function and is out of scope of this specification.

The following example demonstrates a JSON-encoded metadata structure that might be used as DID resolution input metadata.

EXAMPLE 24: JSON-encoded DID resolution input metadata example

```
{
  "accept": "application/did+ld+json"
}
```

This example corresponds to a metadata structure of the following format:

EXAMPLE 25: DID resolution input metadata example

```
«[
  "accept" → "application/did+ld+json"
]»
```

The next example demonstrates a JSON-encoded metadata structure that might be used as DID resolution metadata if a DID was not found.

EXAMPLE 26: JSON-encoded DID resolution metadata example

```
{
  "error": "notFound"
}
```

This example corresponds to a metadata structure of the following format:

EXAMPLE 27: DID resolution metadata example

```
«[
  "error" → "notFound"
]»
```

The next example demonstrates a JSON-encoded metadata structure that might be used as DID document metadata to describe timestamps associated with the DID document.

EXAMPLE 28: JSON-encoded DID document metadata example

```
{
  "created": "2019-03-23T06:35:22Z",
  "updated": "2023-08-10T13:40:06Z"
}
```

This example corresponds to a metadata structure of the following format:

EXAMPLE 29: DID document metadata example

```
«[
  "created" → "2019-03-23T06:35:22Z",
  "updated" → "2023-08-10T13:40:06Z"
]»
```

8. Methods §

A 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 [RFC7230]. 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.

8.1 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 DID Specification Registries [DID-SPEC-REGISTRIES].
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.

8.2 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.

8.3 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 § 8.2 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.

8.4 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.

9. 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. 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.

9.1 Choosing DID Resolvers §

The DID Specification Registries [DID-SPEC-REGISTRIES] 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.

9.2 Proving Control and Binding §

Binding an entity in the digital world or the physical world to a DID, to a DID document, or to cryptographic material requires, the use of security protocols contemplated by this specification. The following sections describe some possible scenarios and how an entity therein might prove control over a DID or a DID document for the purposes of authentication or authorization.

Proving Control of a DID and/or DID Document §

Proving control over a DID and/or a DID Document is useful when updating either in a verifiable data registry or authenticating with remote systems. Cryptographic digital signatures and verifiable timestamps enable certain security protocols related to DID documents to be cryptographically verifiable. For these purposes, this specification defines useful verification relationships in § 5.3.1 Authentication and § 5.3.4 Capability Invocation. The secret cryptographic material associated with the verification methods can be used to generate a cryptographic digital signature as a part of an authentication or authorization security protocol.

> NOTE: Signed DID documents
>
> Some DID methods allow digital signatures and other proofs to be included in the DID document or a § 7.3 Metadata Structure. However, such proofs by themselves do not necessarily prove control over a DID, or guarantee that the DID document is the correct one for the DID. In order to obtain the correct DID document and verify control over a DID, it is necessary to perform the DID resolution process as defined by the DID method.

Binding to Physical Identity §

A DID and DID document do not inherently carry any personal data and it is strongly advised that non-public entities do not publish personal data in DID documents.

It can be useful to express a binding of a DID to a person's or organization's physical identity in a way that is provably asserted by a trusted authority, such as a government. This specification

provides the § 5.3.2 Assertion verification relationship for these purposes. This feature can enable interactions that are private and can be considered legally enforceable under one or more jurisdictions; establishing such bindings has to be carefully balanced against privacy considerations (see § 10. Privacy Considerations).

The process of binding a DID to something in the physical world, such as a person or an organization — for example, by using verifiable credentials with the same subject as that DID — is contemplated by this specification and further defined in the Verifiable Credentials Data Model [VC-DATA-MODEL].

9.3 Authentication Service Endpoints §

If a DID document publishes a service intended for authentication or authorization of the DID subject (see Section § 5.4 Services), it is the responsibility of the service endpoint provider, subject, or requesting party to comply with the requirements of the authentication protocols supported at that service endpoint.

9.4 Non-Repudiation §

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

- The verifiable data registry supports verifiable timestamps. See § 7.1.3 DID Document Metadata for further information on useful timestamps that can be used during the DID resolution process.
- The subject is monitoring for unauthorized updates as elaborated upon in § 9.5 Notification of DID Document Changes.
- The subject has had adequate opportunity to revert malicious updates according to the authorization mechanism for the DID method.

9.5 Notification of DID Document Changes §

One mitigation against unauthorized changes to a DID document is monitoring and actively notifying the DID subject when there are changes. This is analogous to helping prevent account takeover on conventional username/password accounts by sending password reset notifications to the email addresses on file.

In the case of a DID, there is no intermediary registrar or account provider to generate such notifications. However, if the verifiable data registry on which the DID is registered directly supports change notifications, a subscription service can be offered to DID controllers. Notifications could be sent directly to the relevant service endpoints listed in an existing DID.

If a DID controller chooses to rely on a third-party monitoring service (other than the verifiable data registry itself), this introduces another vector of attack.

9.6 Key and Signature Expiration §

In a decentralized identifier architecture, there might not be centralized authorities to enforce cryptographic material or cryptographic digital signature expiration policies. Therefore, it is with supporting software such as DID resolvers and verification libraries that requesting parties validate that cryptographic material were not expired at the time they were used. Requesting parties might employ their own expiration policies in addition to inputs into their verification processes. For example, some requesting parties might accept authentications from five minutes in the past, while others with access to high precision time sources might require authentications to be time stamped within the last 500 milliseconds.

There are some requesting parties that have legitimate needs to extend the use of already-expired cryptographic material, such as verifying legacy cryptographic digital signatures. In these scenarios, a requesting party might instruct their verification software to ignore cryptographic key material expiration or determine if the cryptographic key material was expired at the time it was used.

9.7 Verification Method Rotation §

Rotation is a management process that enables the secret cryptographic material associated with an existing verification method to be deactivated or destroyed once a new verification method has been added to the DID document. Going forward, any new proofs that a controller would have generated using the old secret cryptographic material can now instead be generated using the new cryptographic material and can be verified using the new verification method.

Rotation is a useful mechanism for protecting against verification method compromise, since frequent rotation of a verification method by the controller reduces the value of a single compromised verification method to an attacker. Performing revocation immediately after rotation is useful for verification methods that a controller designates for short-lived verifications, such as those involved in encrypting messages and authentication.

The following considerations might be of use when contemplating the use of verification method rotation:

- Verification method rotation is a proactive security measure.
- It is generally considered a best practice to perform verification method rotation on a regular basis.
- Higher security environments tend to employ more frequent verification method rotation.

- Verification method rotation manifests only as changes to the current or latest version of a DID document.
- When a verification method has been active for a long time, or used for many operations, a controller might wish to perform a rotation.
- Frequent rotation of a verification method might be frustrating for parties that are forced to continuously renew or refresh associated credentials.
- Proofs or signatures that rely on verification methods that are not present in the latest version of a DID document are not impacted by rotation. In these cases, verification software might require additional information, such as when a particular verification method was expected to be valid as well as access to a verifiable data registry containing a historical record, to determine the validity of the proof or signature. This option might not be available in all DID methods.
- The section on DID method operations specifies the DID operations to be supported by a DID method specification, including update which is expected to be used to perform a verification method rotation.
- A controller performs a rotation when they add a new verification method that is meant to replace an existing verification method after some time.
- Not all DID methods support verification method rotation.

9.8 Verification Method Revocation §

Revocation is a management process that enables the secret cryptographic material associated with an existing verification method to be deactivated such that it ceases to be a valid form of creating new proofs of digital signatures.

Revocation is a useful mechanism for reacting to a verification method compromise. Performing revocation immediately after rotation is useful for verification methods that a controller designates for short-lived verifications, such as those involved in encrypting messages and authentication.

Compromise of the secrets associated with a verification method allows the attacker to use them according to the verification relationship expressed by controller in the DID document, for example, for authentication. The attacker's use of the secrets might be indistinguishable from the legitimate controller's use starting from the time the verification method was registered, to the time it was revoked.

The following considerations might be of use when contemplating the use of verification method revocation:

- Verification method revocation is a reactive security measure.
- It is considered a best practice to support key revocation.

- A controller is expected to immediately revoke any verification method that is known to be compromised.
- Verification method revocation can only be embodied in changes to the latest version of a DID Document; it cannot retroactively adjust previous versions.
- As described in § 5.2.1 Verification Material, absence of a verification method is the only form of revocation that applies to all DID Methods that support revocation.
- If a verification method is no longer exclusively accessible to the controller or parties trusted to act on behalf of the controller, it is expected to be revoked immediately to reduce the risk of compromises such as masquerading, theft, and fraud.
- Revocation is expected to be understood as a controller expressing that proofs or signatures associated with a revoked verification method created after its revocation should be treated as invalid. It could also imply a concern that existing proofs or signatures might have been created by an attacker, but this is not necessarily the case. Verifiers, however, might still choose to accept or reject any such proofs or signatures at their own discretion.
- The section on DID method operations specifies the DID operations to be supported by a DID method specification, including update and deactivate, which might be used to remove a verification method from a DID document.
- Not all DID methods support verification method revocation.
- Even if a verification method is present in a DID document, additional information, such as a public key revocation certificate, or an external allow or deny list, could be used to determine whether a verification method has been revoked.
- The day-to-day operation of any software relying on a compromised verification method, such as an individual's operating system, antivirus, or endpoint protection software, could be impacted when the verification method is publicly revoked.

Revocation Semantics §

Although verifiers might choose not to accept proofs or signatures from a revoked verification method, knowing whether a verification was made with a revoked verification method is trickier than it might seem. Some DID methods provide the ability to look back at the state of a DID at a point in time, or at a particular version of the DID document. When such a feature is combined with a reliable way to determine the time or DID version that existed when a cryptographically verifiable statement was made, then revocation does not undo that statement. This can be the basis for using DIDs to make binding commitments; for example, to sign a mortgage.

If these conditions are met, revocation is not retroactive; it only nullifies future use of the method.

However, in order for such semantics to be safe, the second condition — an ability to know what the state of the DID document was at the time the assertion was made — is expected to apply.

Without that guarantee, someone could discover a revoked key and use it to make cryptographically verifiable statements with a simulated date in the past.

Some DID methods only allow the retrieval of the current state of a DID. When this is true, or when the state of a DID at the time of a cryptographically verifiable statement cannot be reliably determined, then the only safe course is to disallow any consideration of DID state with respect to time, except the present moment. DID ecosystems that take this approach essentially provide cryptographically verifiable statements as ephemeral tokens that can be invalidated at any time by the DID controller.

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 the signature or proof was created.
- The verifier can determine the point in time at which the signature or proof was made; for example, it was anchored on a blockchain.
- For 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.

In systems that are willing to admit metadata other than those constituting cryptographic input, similar trust may be achieved -- but always on the same basis where a careful judgment is made about whether a DID document's content at the moment of a signing event contained the expected content.

9.9 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:

- Performing recovery proactively on an infrequent but regular basis, can help to ensure that control has not been lost.
- 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 services trusted to act on their behalf no longer have the exclusive ability to perform DID operations as described in § 8.2 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.

9.10 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.

9.11 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.

9.12 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.

9.13 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.

9.14 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 § 5.1.3 Also Known As.

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.

9.15 Content Integrity Protection §

DID documents which include links to external machine-readable content such as images, web pages, or schemas are vulnerable to tampering. It is strongly advised that external links are integrity protected using solutions such as a hashlink [HASHLINK]. External links are to be avoided if they cannot be integrity protected and the DID document's integrity is dependent on the external link.

One example of an external link where the integrity of the DID document itself could be affected is the JSON-LD Context [JSON-LD11]. To protect against compromise, DID document consumers are advised to cache local static copies of JSON-LD contexts and/or verify the integrity of external contexts against a cryptographic hash that is known to be associated with a safe version of the external JSON-LD Context.

9.16 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.

9.17 Level of Assurance §

Additional information about the security context of authentication events is often required for compliance reasons, especially in regulated areas such as the financial and public sectors. This information is often referred to as a Level of Assurance (LOA). Examples include the protection of secret cryptographic material, the identity proofing process, and the form-factor of the authenticator.

Payment services (PSD 2) and eIDAS introduce such requirements to the security context. Level of assurance frameworks are classified and defined by regulations and standards such as eIDAS, NIST 800-63-3 and ISO/IEC 29115:2013, including their requirements for the security context, and making recommendations on how to achieve them. This might include strong user authentication where FIDO2/WebAuthn can fulfill the requirement.

Some regulated scenarios require the implementation of a specific level of assurance. Since verification relationships such as `assertionMethod` and `authentication` might be used in some of these situations, information about the applied security context might need to be expressed and provided to a *verifier*. Whether and how to encode this information in the DID document data model is out of scope for this specification. Interested readers might note that 1) the information could be transmitted using Verifiable Credentials [VC-DATA-MODEL], and 2) the DID document data model can be extended to incorporate this information as described in § 4.1 Extensibility, and where § 10. Privacy Considerations is applicable for such extensions.

10. 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. The following sections cover privacy considerations that implementers might find useful when building systems that utilize decentralized identifiers.

10.1 Keep Personal Data Private §

If a DID method specification is written for a public-facing verifiable data registry where corresponding DIDs and DID documents might be made publicly available, it is *critical* that those DID documents contain no personal data. Personal data can instead be transmitted through other means such as 1) Verifiable Credentials [VC-DATA-MODEL], or 2) service endpoints under control of the DID subject or DID controller.

Due diligence is expected to be taken around the use of URLs in service endpoints to prevent leakage of personal data or correlation within a URL of a service endpoint. For example, a URL that contains a username is dangerous to include in a DID Document because the username is likely to be human-meaningful in a way that can reveal information that the DID subject did not consent to sharing. With the privacy architecture suggested by this specification, personal data can be exchanged on a private, peer-to-peer basis using communication channels identified and secured by verification methods in DID documents. This also enables DID subjects and requesting parties to implement the GDPR right to be forgotten, because no personal data is written to an immutable distributed ledger.

10.2 DID Correlation Risks §

Like any type of globally unambiguous identifier, DIDs might be used for correlation. DID controllers can mitigate this privacy risk by using pairwise DIDs that are unique to each relationship; in effect, each DID acts as a pseudonym. A pairwise DID need only be shared with more than one party when correlation is explicitly desired. If pairwise DIDs are the default, then the only need to publish a DID openly, or to share it with multiple parties, is when the DID controller(s) and/or DID subject explicitly desires public identification and correlation.

10.3 DID Document Correlation Risks §

The anti-correlation protections of pairwise DIDs are easily defeated if the data in the corresponding DID documents can be correlated. For example, using identical verification methods or bespoke service endpoints in multiple DID documents can provide as much correlation information as using the same DID. Therefore, the DID document for a pairwise DID also needs to use pairwise unique information, such as ensuring that verification methods are unique to the pairwise relationship.

It might seem natural to also use pairwise unique service endpoints in the DID document for a pairwise DID. However, unique endpoints allow all traffic between two DIDs to be isolated perfectly into unique buckets, where timing correlation and similar analysis is easy. Therefore, a better strategy for endpoint privacy might be to share an endpoint among a large number of DIDs controlled by many different subjects (see § 10.5 Herd Privacy).

10.4 DID Subject Classification §

It is dangerous to add properties to the DID document that can be used to indicate, explicitly or through inference, what *type* or nature of thing the DID subject is, particularly if the DID subject is a person.

Not only do such properties potentially result in personal data (see § 10.1 Keep Personal Data Private) or correlatable data (see § 10.2 DID Correlation Risks and § 10.3 DID Document Correlation Risks) being present in the DID document, but they can be used for grouping particular DIDs in such a way that they are included in or excluded from certain operations or functionalities.

Including *type* information in a DID Document can result in personal privacy harms even for DID Subjects that are non-person entities, such as IoT devices. The aggregation of such information around a DID Controller could serve as a form of digital fingerprint and this is best avoided.

To minimize these risks, all properties in a DID document ought to be for expressing cryptographic material, endpoints, or verification methods related to using the DID.

10.5 Herd Privacy §

When a DID subject is indistinguishable from others in the herd, 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 herd 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.

10.6 Service Privacy §

The ability for a controller to optionally express at least one service endpoint in the DID document increases their control and agency. Each additional endpoint in the DID document adds privacy risk either due to correlation, such as across endpoint descriptions, or because the services are not protected by an authorization mechanism, or both.

DID documents are often public and, since they are standardized, will be stored and indexed efficiently by their very standards-based nature. This risk is worse if DID documents are published to immutable verifiable data registries. Access to a history of the DID documents referenced by a DID represents a form of traffic analysis made more efficient through the use of standards.

The degree of additional privacy risk caused by using multiple service endpoints in one DID document can be difficult to estimate. Privacy harms are typically unintended consequences. DIDs can refer to documents, services, schemas, and other things that might be associated with individual people, households, clubs, and employers — and correlation of their service endpoints could become a powerful surveillance and inference tool. An example of this potential harm can be seen when multiple common country-level top level domains such as `https://example.co.uk` might be used to infer the approximate location of the DID subject with a greater degree of probability.

Maintaining Herd Privacy §

The variety of possible endpoints makes it particularly challenging to maintain herd privacy, in which no information about the DID subject is leaked (see § 10.5 Herd Privacy).

First, because service endpoints might be specified as URIs, they could unintentionally leak personal information because of the architecture of the service. For example, a service endpoint of `http://example.com/MyFirstName` is leaking the term `MyFirstName` to everyone who can access the DID document. When linking to legacy systems, this is an unavoidable risk, and care is expected to be taken in such cases. This specification encourages new, DID-aware endpoints to use nothing more than the DID itself for any identification necessary. For example, if a service description were to include `http://example.com/did%3Aexample%3Aabc123`, no harm would be done because `did:example:abc123` is already exposed in the DID Document; it leaks no additional information.

Second, because a DID document can list multiple service endpoints, it is possible to irreversibly associate services that are not associated in any other context. This correlation on its own may lead

to privacy harms by revealing information about the DID subject, even if the URIs used did not contain any sensitive information.

Third, because some types of DID subjects might be more or less likely to list specific endpoints, the listing of a given service could, by itself, leak information that can be used to infer something about the DID subject. For example, a DID for an automobile might include a pointer to a public title record at the Department of Motor Vehicles, while a DID for an individual would not include that information.

It is the goal of herd privacy to ensure that the nature of specific DID subjects is obscured by the population of the whole. To maximize herd privacy, implementers need to rely on one — and only one — service endpoint, with that endpoint providing a proxy or mediator service that the controller is willing to depend on, to protect such associations and to blind requests to the ultimate service.

Service Endpoint Alternatives §

Given the concerns in the previous section, implementers are urged to consider any of the following service endpoint approaches:

- **Negotiator Endpoint** — Service for negotiating mutually agreeable communications channels, preferably using private set intersection. The output of negotiation is a communication channel and whatever credentials might be needed to access it.
- **Tor Endpoint** (Tor Onion Router) — Provide a privacy-respecting address for reaching service endpoints. Any service that can be provided online can be provided through TOR for additional privacy.
- **Mediator Endpoint** — Mediators provide a generic endpoint, for multiple parties, receive encrypted messages on behalf of those parties, and forward them to the intended recipient. This avoids the need to have a specific endpoint per subject, which could create a correlation risk. This approach is also called a proxy.
- **Confidential Storage** — Proprietary or confidential personal information might need to be kept off of a verifiable data registry to provide additional privacy and/or security guarantees, especially for those DID methods where DID documents are published on a public ledger. Pointing to external resource services provides a means for authorization checks and deletion.
- **Polymorphic Proxy** — A proxy endpoint that can act as any number of services, depending on how it is called. For example, the same URL could be used for both negotiator and mediator functions, depending on a mechanism for re-routing.

These service endpoint types continue to be an area of innovation and exploration.

A. Examples §

A.1 DID Documents §

This section is non-normative.

See Verification Method Types [DID-SPEC-REGISTRIES] 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 30: DID Document with 1 verification method type

```
{
  "@context": [
    "https://www.w3.org/ns/did/v1",
    "https://w3id.org/security/suites/ed25519-2020/v1"
  ],
  "id": "did:example:123",
  "authentication": [
    {
      "id": "did:example:123#z6MkecaLyHuYWkayBDLw5ihndj3T1m6zKTGqau3
      "type": "Ed25519VerificationKey2020", // external (property va
      "controller": "did:example:123",
      "publicKeyMultibase": "zAKJP3f7BD6W4iWEQ9jwndVTCBq8ua2Utt8EEjJ
    }
  ],
  "capabilityInvocation": [
    {
      "id": "did:example:123#z6MkhdmzFu659ZJ4XKj31vtEDmjvsi5yDZG5L7C
      "type": "Ed25519VerificationKey2020", // external (property va
      "controller": "did:example:123",
      "publicKeyMultibase": "z4BWwfeqdp1obQptLLMvPNgBw48p7og1ie6Hf9p
    }
  ],
  "capabilityDelegation": [
    {
      "id": "did:example:123#z6Mkw94ByR26zMSkNdCUi6FNRsWnc2DFEeDXyBG
      "type": "Ed25519VerificationKey2020", // external (property va
      "controller": "did:example:123",
      "publicKeyMultibase": "zHgo9PAmfeoxHG8Mn2XHXamxnnSwPpkyBHAMNF3
    }
  ],
  "assertionMethod": [
    {
      "id": "did:example:123#z6MkiukuAuQAE8ozxvmahnQGzApvtW7KT5XXKfo
      "type": "Ed25519VerificationKey2020", // external (property va
      "controller": "did:example:123",
      "publicKeyMultibase": "z5TVraf9itbKXrRvt2DSS95Gw4vqU3CHAdetouf
    }
  ]
}
```

EXAMPLE 31: DID Document with many different key types

```
{
  "@context": [
    "https://www.w3.org/ns/did/v1",
    "https://w3id.org/security/suites/jws-2020/v1"
  ],
  "verificationMethod": [
    {
      "id": "did:example:123#key-0",
      "type": "JsonWebKey2020",
      "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": "JsonWebKey2020",
      "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": "JsonWebKey2020",
      "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": "JsonWebKey2020",
      "controller": "did:example:123",
      "publicKeyJwk": {
        "kty": "EC", // external (property name)
        "crv": "secp256k1", // external (property name)
        "x": "U1V4TVZVMUpUa0ZVU1NBcU9CRm5IbmFaaEpGNkxkdWx", // externa
```

```
        "y": "i5a2NtJoUKXkLm6q8nOEu9WOkso1Ag6FTUT6k_LMnGk" // external
      }
    },
    {
      "id": "did:example:123#key-4",
      "type": "JsonWebKey2020",
      "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": "JsonWebKey2020",
      "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": "JsonWebKey2020",
      "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": "JsonWebKey2020",
      "controller": "did:example:123",
      "publicKeyJwk": {
        "kty": "RSA", // external (property name)
        "e": "AQAB", // external (property name)
        "n": "UkhWaGJGOUZRMTlFVWtKSElBdENGV2hlU1F2djFNRXh1NVJMQ01UNGpW
      }
    }
  ]
}
```

EXAMPLE 32: DID Document with different verification method types

```
{
  "@context": [
    "https://www.w3.org/ns/did/v1",
    "https://w3id.org/security/suites/ed25519-2018/v1",
    "https://w3id.org/security/suites/x25519-2019/v1",
    "https://w3id.org/security/suites/secp256k1-2019/v1",
    "https://w3id.org/security/suites/jws-2020/v1"
  ],
  "verificationMethod": [
    {
      "id": "did:example:123#key-0",
      "type": "Ed25519VerificationKey2018",
      "controller": "did:example:123",
      "publicKeyBase58": "3M5RCDjPTWPkKSN3sxUmmMqHbmRPegYP1tjcKyrDbt9J
    },
    {
      "id": "did:example:123#key-1",
      "type": "X25519KeyAgreementKey2019",
      "controller": "did:example:123",
      "publicKeyBase58": "FbQWLPRhTH95MCkQUeFYdiSoQt8zMwetqfWoxqPgaq7x
    },
    {
      "id": "did:example:123#key-2",
      "type": "EcdsaSecp256k1VerificationKey2019",
      "controller": "did:example:123",
      "publicKeyBase58": "ns2aFDq25fEV1NUd3wZ65sgj5QjFW8JCAHdUJfLwfodt
    },
    {
      "id": "did:example:123#key-3",
      "type": "JsonWebKey2020",
      "controller": "did:example:123",
      "publicKeyJwk": {
        "kty": "EC", // external (property name)
        "crv": "P-256", // external (property name)
        "x": "Er6KSSnAjI70ObRWhlaMgqyIOQYrDJTE94ej5hybQ2M", // externa
        "y": "pPVzCOTJwgikPjuUE6UebfZySqEJ0ZtsWFpj7YSPGEk" // external
      }
    }
  ]
}
```

A.2 Proving §

This section is non-normative.

> NOTE
>
> These examples are for information purposes only. See W3C Verifiable Credentials Data Model for additional examples.

EXAMPLE 33: Verifiable Credential linked to a verification method of type Ed25519VerificationKey2020

```
{  // external (all terms in this example)
  "@context": [
    "https://www.w3.org/2018/credentials/v1",
    "https://w3id.org/citizenship/v1"
  ],
  "type": [
    "VerifiableCredential",
    "PermanentResidentCard"
  ],
  "credentialSubject": {
    "id": "did:example:123",
    "type": [
      "PermanentResident",
      "Person"
    ],
    "givenName": "JOHN",
    "familyName": "SMITH",
    "gender": "Male",
    "image": "...kJggg==",
    "residentSince": "2015-01-01",
    "lprCategory": "C09",
    "lprNumber": "000-000-204",
    "commuterClassification": "C1",
    "birthCountry": "Bahamas",
    "birthDate": "1958-08-17"
  },
  "issuer": "did:example:456",
  "issuanceDate": "2020-04-22T10:37:22Z",
  "identifier": "83627465",
  "name": "Permanent Resident Card",
  "description": "Government of Example Permanent Resident Card.",
  "proof": {
    "type": "Ed25519Signature2018",
    "created": "2020-04-22T10:37:22Z",
    "proofPurpose": "assertionMethod",
    "verificationMethod": "did:example:456#key-1",
    "jws": "eyJjcml0IjpbImI2NCJdLCJiNjQiOmZhbHNlLCJhbGciOiJFZERTQSJ9..
  }
}
```

EXAMPLE 34: Verifiable Credential linked to a verification method of type JsonWebKey2020

```
{  // external (all terms in this example)
  "@context": [
    "https://www.w3.org/2018/credentials/v1",
    "https://www.w3.org/2018/credentials/examples/v1"
  ],
  "id": "http://example.gov/credentials/3732",
  "type": ["VerifiableCredential", "UniversityDegreeCredential"],
  "issuer": { "id": "did:example:123" },
  "issuanceDate": "2020-03-10T04:24:12.164Z",
  "credentialSubject": {
    "id": "did:example:456",
    "degree": {
      "type": "BachelorDegree",
      "name": "Bachelor of Science and Arts"
    }
  },
  "proof": {
    "type": "JsonWebSignature2020",
    "created": "2020-02-15T17:13:18Z",
    "verificationMethod": "did:example:123#_Qq0UL2Fq651Q0Fjd6TvnYE-faH
    "proofPurpose": "assertionMethod",
    "jws": "eyJiNjQiOmZhbHNlLCJjcml0IjpbImI2NCJdLCJhbGciOiJFZERTQSJ9..
  }
}
```

EXAMPLE 35: Verifiable Credential linked to a bls12381 verification method


```
{  // external (all terms in this example)
  "@context": [
    "https://www.w3.org/2018/credentials/v1",
    "https://w3id.org/security/bbs/v1",
    {
      "name": "https://schema.org/name",
      "birthDate": "https://schema.org/birthDate"
    }
  ],
  "id": "urn:uuid:c499e122-3ba9-4e95-8d4d-c0ebfcf8c51a",
  "type": ["VerifiableCredential"],
  "issuanceDate": "2021-02-07T16:02:08.571Z",
  "issuer": {
    "id": "did:example:123"
  },
  "credentialSubject": {
    "id": "did:example:456",
    "name": "John Smith",
    "birthDate": "2021-02-07"
  },
  "proof": {
    "type": "BbsBlsSignature2020",
    "created": "2021-02-07T16:02:10Z",
    "proofPurpose": "assertionMethod",
    "proofValue": "o7zD2eNTp657YzkJLub+IO4Zqy/R3Lv/AWmtSA/kUlEAOa73BNy
    "verificationMethod": "did:example:123#bls12381-g2-key"
  }
}
```

EXAMPLE 36: Verifiable Credential selective disclosure zero knowledge proof linked to a bls12381 verification method


```
{  // external (all terms in this example)
  "@context": [
    "https://www.w3.org/2018/credentials/v1",
    "https://w3id.org/security/bbs/v1",
    {
      "name": "https://schema.org/name",
      "birthDate": "https://schema.org/birthDate"
    }
  ],
  "id": "urn:uuid:c499e122-3ba9-4e95-8d4d-c0ebfcf8c51a",
  "type": "VerifiableCredential",
  "issuanceDate": "2021-02-07T16:02:08.571Z",
  "issuer": {
    "id": "did:example:123"
  },
  "credentialSubject": {
    "id": "did:example:456",
    "birthDate": "2021-02-07"
  },
  "proof": {
    "type": "BbsBlsSignatureProof2020",
    "created": "2021-02-07T16:02:10Z",
    "nonce": "OqZHsV/aunS34BhLaSoxiHWK+SUaG4iozM3V+1jO06zRRNcDWID+I0uw
    "proofPurpose": "assertionMethod",
    "proofValue": "AAsH34lcKsqaqPaLQWcnLMe3mDM+K7fZM0t4Iesfj7BhD//HBtu
    "verificationMethod": "did:example:123#bls12381-g2-key"
  }
}
```

EXAMPLE 37: Verifiable Credential as Decoded JWT

```
{ // external (all terms in this example)
  "protected": {
    "kid": "did:example:123#_Qq0UL2Fq651Q0Fjd6TvnYE-faHiOpRlPVQcY_-tA4
    "alg": "EdDSA"
  },
  "payload": {
    "iss": "did:example:123",
    "sub": "did:example:456",
    "vc": {
      "@context": [
        "https://www.w3.org/2018/credentials/v1",
        "https://www.w3.org/2018/credentials/examples/v1"
      ],
      "id": "http://example.gov/credentials/3732",
      "type": [
        "VerifiableCredential",
        "UniversityDegreeCredential"
      ],
      "issuer": {
        "id": "did:example:123"
      },
      "issuanceDate": "2020-03-10T04:24:12.164Z",
      "credentialSubject": {
        "id": "did:example:456",
        "degree": {
          "type": "BachelorDegree",
          "name": "Bachelor of Science and Arts"
        }
      }
    },
    "jti": "http://example.gov/credentials/3732",
    "nbf": 1583814252
  },
  "signature": "qSv6dpZJGFybtcifLwGf4ujzlEu-fam_M7HPxinCbVhz9iIJCg70UM
}
```

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 38: 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 §

B.1 Detailed Architecture Diagram §

Following is a diagram showing the relationships among § 4. Data Model, § 5. Core Properties, and § 8. Methods, and § 7. Resolution.

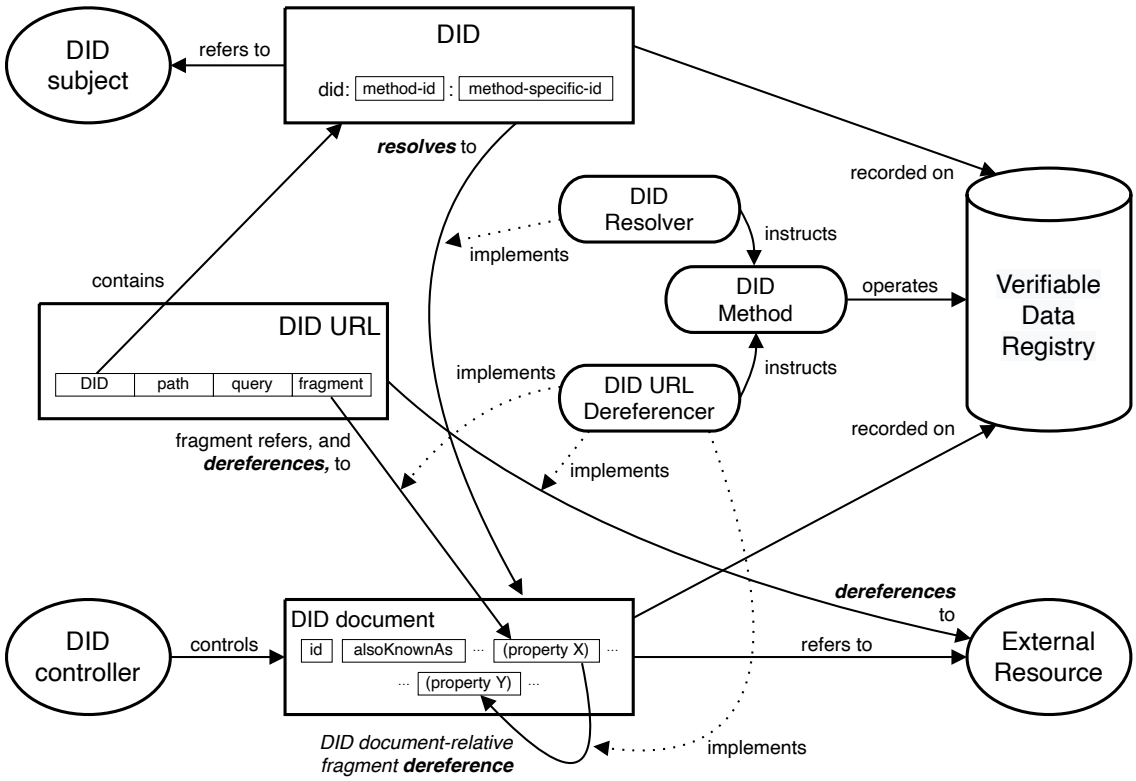

Figure 7 Detailed overview of DID architecture and the relationship of the basic components.

B.2 Creation of a DID §

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 §

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 §

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 describing the DID subject.

This distinction is illustrated by the graph model shown below.

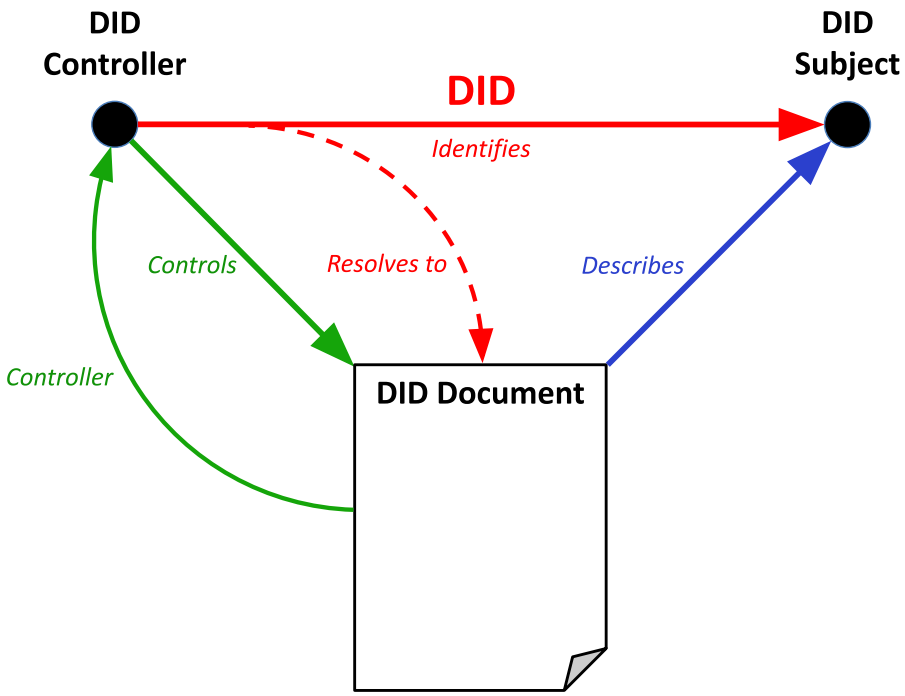

Figure 8 A DID is an identifier assigned by a DID controller to refer to a DID subject and resolve to a DID document that describes the DID subject. The DID document is an artifact of DID resolution and not a separate resource distinct from the DID subject. See also: narrative description.

B.5 Statements in the DID document §

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 8 with a direct link between the DID and the DID subject.

B.6 Discovering more information about the DID subject §

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.

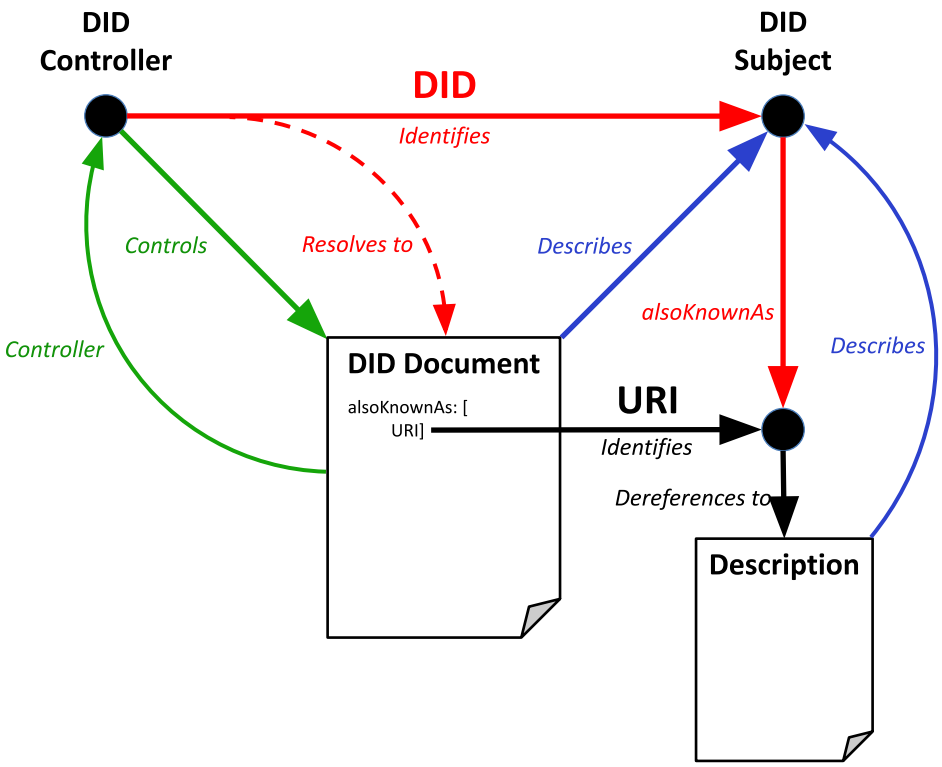

Figure 9 A DID document can use the alsoKnownAs property to assert that another URI (including, but not necessarily, another DID) refers to the same DID subject. See also: narrative description.

B.7 Serving a representation of the DID subject §

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 DID parameter (see § 3.2.1 DID Parameters) 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 §

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 §

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 §

The first case, shown in Figure 10, 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.

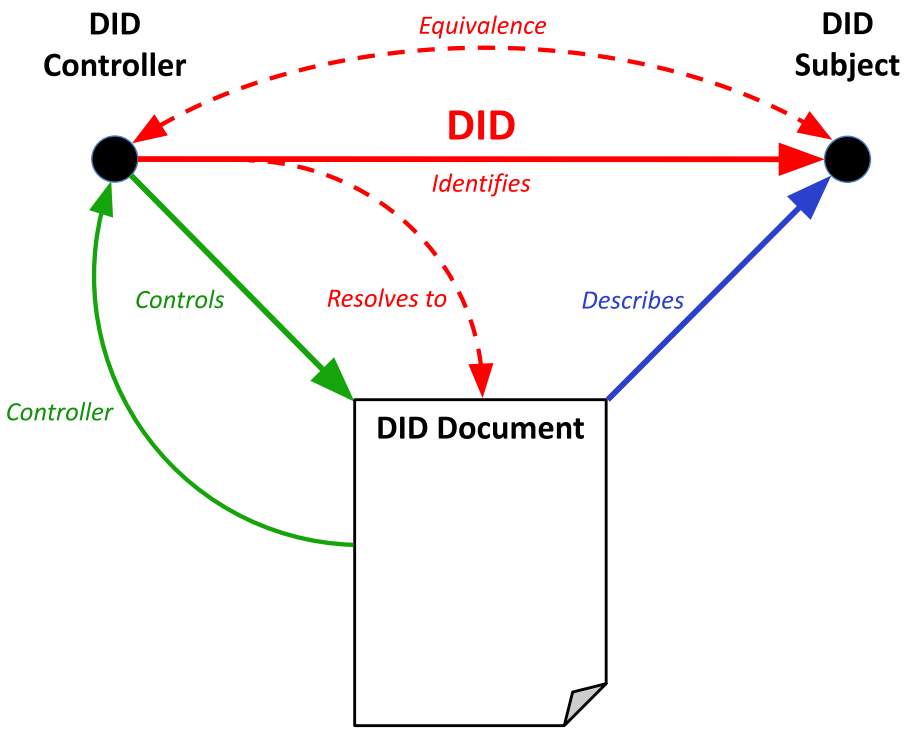

Figure 10 The DID subject is the same entity as the DID controller. See also: narrative description.

From a graph model perspective, even though the nodes identified as the DID controller and DID subject in Figure 10 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 §

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 §

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

B.10.1 Independent Control §

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.

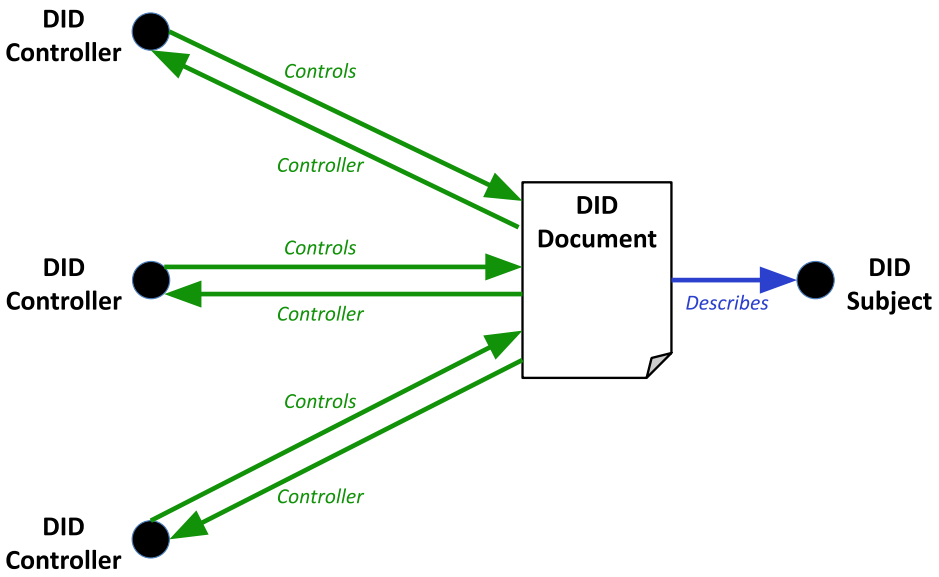

Figure 11 Multiple independent DID controllers that can each act independently. See also: Text Description

B.10.2 Group Control §

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"). 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 12.

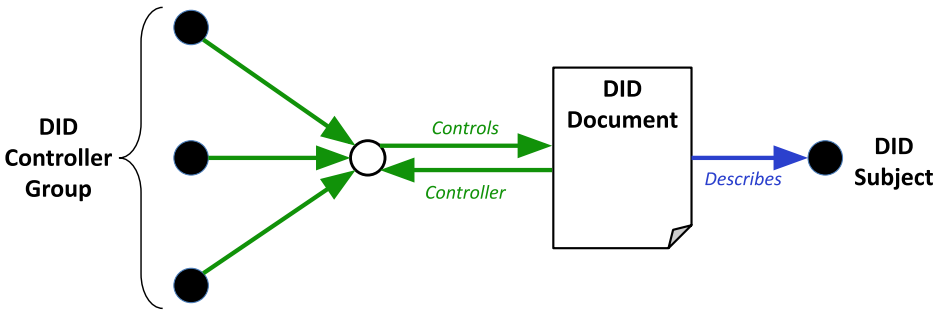

Figure 12 Multiple DID controllers who are expected to act together as a DID controller group. See also: narrative description.

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 §

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 § 9.16 Persistence.

B.12 Changing the DID controller §

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

Changes since the 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`.
- Remove at risk issue markers for features that gained enough implementation experience.

- Finalize the Editors, Authors, and Acknowledgements information.

Changes since the 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+ld+json 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 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 DID Specifications Registry 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 §

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.

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 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+json §

Type name:
application

Subtype name:
did+json

Required parameters:
None

Optional parameters:
None

Encoding considerations:
See RFC 8259, section 11.

Security considerations:
See RFC 8259, section 12 [RFC8259].

Interoperability considerations:
Not Applicable

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

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 W3C Verifiable Credentials.

Additional information:

Magic number(s):
Not Applicable

File extension(s):
.didjson

Macintosh file type code(s):
TEXT

Person & email address to contact for further information:
Ivan Herman <ivan@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 with application/did+json are treated according to the rules defined in § Fragment.

E.2 application/did+ld+json §

> NOTE: IETF Structured Media Types
>
> The Candidate Recommendation phase for this specification received a significant number of implementations for the `application/did+ld+json` media type. Registration of the media type `application/did+ld+json` at IANA is pending resolution of the Media Types with Multiple Suffixes issue. Work is expected to continue in the IETF Media Type Maintenance Working Group with a registration of the `application/did+ld+json` media type by W3C following shortly after the publication of the Media Types with Multiple Suffixes RFC.

Type name:
application

Subtype name:

did+ld+json

Required parameters:

None

Optional parameters:

None

Encoding considerations:

See RFC 8259, section 11.

Security considerations:

See JSON-LD 1.1, Security Considerations [JSON-LD11].

Interoperability considerations:

Not Applicable

Published specification:

https://www.w3.org/TR/did-core/

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 W3C Verifiable Credentials.

Additional information:

Magic number(s):

Not Applicable

File extension(s):

.didjsonld

Macintosh file type code(s):

TEXT

Person & email address to contact for further information:

Ivan Herman <ivan@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 with application/did+ld+json are treated according to the rules associated with the JSON-LD 1.1: application/ld+json media type [JSON-LD11].

F. References §

F.1 Normative references §

[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. 12 December 2019. W3C Candidate 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://tools.ietf.org/html/rfc2119

[RFC3552]

Guidelines for Writing RFC Text on Security Considerations. E. Rescorla; B. Korver. IETF. July 2003. Best Current Practice. URL: https://datatracker.ietf.org/doc/html/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://tools.ietf.org/html/rfc5234

[RFC7517]

JSON Web Key (JWK). M. Jones. IETF. May 2015. Proposed Standard. URL: https://tools.ietf.org/html/rfc7517

[RFC7638]

JSON Web Key (JWK) Thumbprint. M. Jones; N. Sakimura. IETF. September 2015. Proposed Standard. URL: https://tools.ietf.org/html/rfc7638

[RFC8174]

Ambiguity of Uppercase vs Lowercase in RFC 2119 Key Words. B. Leiba. IETF. May 2017. Best Current Practice. URL: https://tools.ietf.org/html/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-RESOLUTION]

Decentralized Identifier Resolution. Markus Sabadello; Dmitri Zagidulin. Credentials Community Group. Draft Community Group Report. URL: https://w3c-ccg.github.io/did-resolution/

[DID-RUBRIC]

Decentralized Characteristics Rubric v1.0. Joe Andrieu. Credentials Community Group. Draft Community Group Report. URL: https://w3c.github.io/did-rubric/

[DID-SPEC-REGISTRIES]

DID Specification Registries. Orie Steele; Manu Sporny. W3C. 24 June 2021. W3C Note. URL: https://www.w3.org/TR/did-spec-registries/

[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 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/

[HASHLINK]

Cryptographic Hyperlinks. Manu Sporny. IETF. December 2018. Internet-Draft. URL: https://tools.ietf.org/html/draft-sporny-hashlink-05

[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

[MULTIBASE]

The Multibase Encoding Scheme. Juan Benet; Manu Sporny. IETF. February 2021. Internet-Draft. URL: https://datatracker.ietf.org/doc/html/draft-multiformats-multibase-03

[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

[RFC4122]

A Universally Unique IDentifier (UUID) URN Namespace. P. Leach; M. Mealling; R. Salz. IETF. July 2005. Proposed Standard. URL: https://tools.ietf.org/html/rfc4122

[RFC6901]

JavaScript Object Notation (JSON) Pointer. P. Bryan, Ed.; K. Zyp; M. Nottingham, Ed.. IETF. April 2013. Proposed Standard. URL: https://tools.ietf.org/html/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://tools.ietf.org/html/rfc6973

[RFC7230]

Hypertext Transfer Protocol (HTTP/1.1): Message Syntax and Routing. R. Fielding, Ed.; J. Reschke, Ed.. IETF. June 2014. Proposed Standard. URL: https://httpwg.org/specs/rfc7230.html

[RFC7231]

Hypertext Transfer Protocol (HTTP/1.1): Semantics and Content. R. Fielding, Ed.; J. Reschke, Ed.. IETF. June 2014. Proposed Standard. URL: https://httpwg.org/specs/rfc7231.html

[RFC8141]

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

[VC-DATA-MODEL]

Verifiable Credentials Data Model 1.0. Manu Sporny; Grant Noble; Dave Longley; Daniel Burnett; Brent Zundel. W3C. 19 November 2019. W3C Recommendation. URL: https://www.w3.org/TR/vc-data-model/

↥