Decentralized Identifiers (DIDs) v1.0

Core architecture, data model, and representations

W3C Working Draft 02 October 2020

This version:

https://www.w3.org/TR/2020/WD-did-core-20201002/

Latest published version:

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

Latest editor's draft:

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

Previous version:

https://www.w3.org/TR/2020/WD-did-core-20201001/

Editors:

Drummond Reed (Evernym)

Manu Sporny (Digital Bazaar)

Markus Sabadello (Danube Tech)

Authors:

Drummond Reed (Evernym)

Manu Sporny (Digital Bazaar)

Dave Longley (Digital Bazaar)

Christopher Allen (Blockchain Commons)

Ryan Grant

Markus Sabadello (Danube Tech)

Jonathan Holt, DO, MS (ConsenSys Health)

Participate:

GitHub w3c/did-core

File a bug

Commit history

Pull requests

Abstract

Decentralized identifiers (DIDs) are a new type of identifier that enables verifiable, decentralized digital identity. A DID identifies any subject (e.g., a person, organization, thing, data model, abstract entity, etc.) that the controller of the DID decides that it identifies. 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 URLs 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 service endpoints, which provide a set of mechanisms enabling a DID controller to prove control of the DID. Service endpoints enable trusted interactions associated with the DID subject. A DID document might contain semantics about the subject that it identifies. A DID document might contain the DID subject itself (e.g. a data model).

This document specifies a common data model, a URL format, and a set of operations for DIDs, DID documents, and DID methods.

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/.

This specification is under active development and implementers are advised against implementing the specification unless they are directly involved with the W3C DID Working Group. There are use cases [DID-USE-CASES] in active development that establish requirements for this document.

At present, there exist 40 experimental implementations and a preliminary test suite that will eventually determine whether or not implementations are conformant. Readers are advised that Appendix § A. Current Issues contains a list of concerns and proposed changes that will most likely result in alterations to this specification.

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

Portions of the work on this specification have been funded by the United States Department of Homeland Security's Science and Technology Directorate under contracts HSHQDC-16-R00012-H-SB2016-1-002 and HSHQDC-17-C-00019. 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.

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.

This document was published by the Decentralized Identifier Working Group as a Working Draft. 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 (archives).

Publication as a Working Draft 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.

This document was produced by a group operating under the 1 August 2017 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). Resources on the Internet are identified by globally unique identifiers in the form of MAC addresses; 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 identify and when they can be revoked. They are useful only in certain contexts and recognized only by certain bodies (not of our choosing). They may disappear or cease to be valid with the failure of an organization. They may unnecessarily reveal personal information. And in many cases they can be fraudulently replicated and asserted by a malicious third-party ("identity theft").

The Decentralized Identifiers (DIDs) defined in this specification are a new type of globally unique identifier designed to enable individuals and organizations to generate our own identifiers using

systems we trust, and to prove control of those identifiers (authenticate) using cryptographic proofs (for example, digital signatures, privacy-preserving biometric protocols, and so on).

Because we control the generation and assertion of these identifiers, each of us can have as many DIDs as we need to respect our desired separation of identities, personas, and contexts (in the everyday sense of these words). We can scope the use of these identifiers to the most appropriate contexts. We can interact with other people, institutions or systems that require us to identify ourselves (or things we control) while maintaining control over how much personal or private data should be revealed, and without depending on a central authority to guarantee the continued existence of the identifier.

This specification does not require any particular technology or cryptography to underpin the generation, persistence, resolution or interpretation of DIDs. Rather, it defines: a) the generic syntax for all DIDs, and b) the generic requirements for performing the four basic CRUD operations (create, read, update, deactivate) on the metadata associated with a DID (called the DID document).

This enables implementers to design specific types of DIDs to work with the computing infrastructure they trust (e.g., blockchain, distributed ledger, decentralized file system, distributed database, peer-to-peer network). The specification for a specific type of DID is called a DID method. Implementers of applications or systems using DIDs can choose to support the DID methods most appropriate for their particular use cases.

This specification is for:

- Developers who want to enable users of their system to generate and assert their own identifiers (producers of DIDs);
- Developers who want to enable their systems to accept user-controlled identifiers (consumers of DIDs);
- Developers who wish to enable the use of DIDs with particular computing infrastructure (DID method developers).

> NOTE: Diversity of DID systems
>
> DID methods can also be developed for 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.

1.1 A Simple Example §

This section is non-normative.

A DID is a simple text string consisting of three parts, the:

- URI scheme identifier (`did`)
- Identifier for the DID method
- DID method-specific identifier.

> EXAMPLE 1: A simple example of a decentralized identifier (DID)
>
> ```
> did:example:123456789abcdefghi
> ```

The example DID above resolves to a DID document. A DID document contains information associated with the DID, such as ways to cryptographically authenticate the DID controller, as well as services that can be used to interact with the DID subject.

> EXAMPLE 2: Minimal self-managed DID document
>
> ```
> {
>   "@context": "https://www.w3.org/ns/did/v1",
>   "id": "did:example:123456789abcdefghi",
>   "authentication": [{
>     // used to authenticate as did:...fghi
>     "id": "did:example:123456789abcdefghi#keys-1",
>     "type": "Ed25519VerificationKey2018",
>     "controller": "did:example:123456789abcdefghi",
>     "publicKeyBase58": "H3C2AVvLMv6gmMNam3uVAjZpfkcJCwDwnZn6z3wXmqPV"
>   }],
>   "service": [{
>     // used to retrieve Verifiable Credentials associated with the DID
>     "id":"did:example:123456789abcdefghi#vcs",
>     "type": "VerifiableCredentialService",
>     "serviceEndpoint": "https://example.com/vc/"
>   }]
> }
> ```

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 drove the design goals for this specification. This section summarizes the primary design goals for this specification.

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, service endpoints, and other metadata.
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 provides a basic understanding of the major elements of DID architecture. Formal definitions of terms are provided in § 2. Terminology .

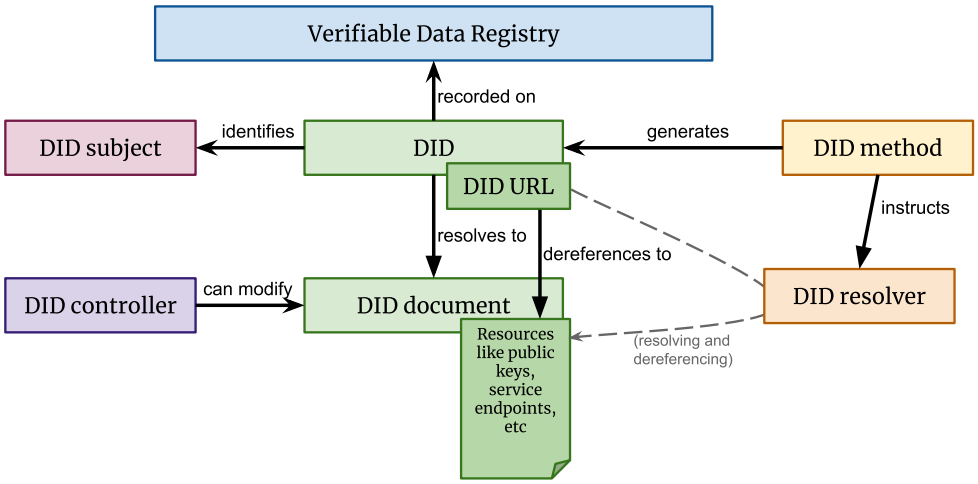

Figure 1 The basic components of DID architecture.

DIDs and DID URLs

A DID, or Decentralized Identifier, is a URI composed of three parts: the scheme "did:", a method identifier, and a unique, method-specific identifier generated 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 (path, query, fragment) in order to locate a particular resource—for example, a public key inside a DID document, or a resource available external to the DID document.

DID Subjects

The subject of a DID is, by definition, the entity identified by the DID. The DID subject may also be the DID controller. Anything can be the subject of a DID: person, group, organization, physical thing, logical thing, etc.

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 may also be asserted via other mechanisms. Note that a DID may have more than one controller, and the controller(s) may include the DID subject.

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.

DID documents

DID documents contain metadata associated with a DID. They typically express verification methods (such as public keys) and services relevant to interactions with the DID subject. A DID document is serialized according to a particular syntax (see § 6. Core Representations). The DID itself is the value of the `id` property. The generic properties supported in a DID document are specified in § 5. Core Properties. The properties present in a DID document may be updated according to the applicable operations outlined in § 7. Methods .

DID Methods

DID methods are the mechanism by which a particular type of DID and its associated DID document are created, resolved, updated, and deactivated using a particular verifiable data registry. DID methods are defined using separate DID method specifications (see § 7. Methods).

> NOTE
>
> 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: and https: schemes specified in [RFC7230]). It is also similar to the relationship between the IETF generic URN specification ([RFC8141]) and a specific URN namespace definition, (such as the UUID URN namespace defined in [RFC4122]). The difference is that a DID method specification, as well as defining a specific DID scheme, also specifies the methods creating, resolving, updating, and deactivating DIDs and DID documents using a specific type of verifiable data registry.

DID resolvers and DID resolution

A DID resolver is a software and/or hardware component that takes a DID (and associated input metadata) as input and produces a conforming DID document (and associated metadata) as output. This process is called DID resolution. The inputs and outputs of the DID resolution process are defined in § 8. Resolution . The specific steps for resolving a specific type of DID are defined by the relevant DID method specification. Additional considerations for implementing a DID resolver are discussed in [DID-RESOLUTION].

DID URL dereferencers and DID URL dereferencing

A DID URL dereferencer is a software and/or hardware component that takes a DID URL (and associated input metadata) as input and produces a resource (and associated metadata) as output. This process is called DID URL dereferencing. The inputs and outputs of the DID URL dereferencing process are defined in § 8.2 DID URL Dereferencing . Additional considerations for implementing a DID URL dereferencer are discussed in [DID-RESOLUTION].

1.4 Conformance §

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

The key words *MAY*, *MUST*, *MUST NOT*, *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, CBOR, 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, CBOR, or JSON-LD.

Interoperability of implementations for DIDs and DID documents will be tested by evaluating an implementation's ability to create and parse DIDs and DID documents that conform to the 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 must be interoperable for that method).

A ***conforming DID*** is any concrete expression of the rules specified in Section § 3. Identifier and *MUST* comply with relevant normative statements in that section.

A ***conforming DID document*** is any concrete expression of the data model described in this specification and *MUST* comply with the relevant normative statements in Sections § 4. Data Model and § 5. Core Properties. A serialization format for the conforming document *MUST* be deterministic, bi-directional, and lossless as described in Section § 6. Core Representations. The conforming DID document *MAY* be transmitted or stored in any such serialization format.

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

A ***conforming producer*** is any algorithm realized as software and/or hardware and conforms to this specification if it generates conforming DIDs or conforming DID Documents. A conforming producer *MUST NOT* produce non-conforming DIDs or DID documents.

A ***conforming consumer*** is any algorithm realized as software and/or hardware and conforms to this specification if it consumes conforming DIDs or conforming DID documents. A conforming consumer *MUST* produce errors when consuming non-conforming DIDs or DID documents.

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.

authenticate

Authentication is a process (typically some type of protocol) 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 private key associated with a public key published in a DID document.

blockchain

A specific type of distributed ledger technology (DLT) in which ledger entries are stored in blocks of transactions that are grouped together and hashed into a cryptographic chain. Because this type of DLT was introduced by Bitcoin, the term *blockchain* is sometimes used to refer specifically to the Bitcoin ledger.

binding

A concrete mechanism used by a caller to invoke a DID resolver or a DID URL dereferencer. This could be a local command line tool, a software library, or a network call such as an HTTPS request.

***decentralized identifier* (DID)**

A globally unique persistent identifier that does not require a centralized registration authority because it is generated and/or registered cryptographically. The generic format of a DID is defined in the DID Core specification. 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.

***decentralized public key infrastructure* (DPKI)**

Public key infrastructure that does not rely on traditional certificate authorities because it uses decentralized identifiers and DID documents) to discover and verify public key descriptions.

DID controller

An entity that has the capability to make changes to a DID document. A DID may 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 one DID controller may 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 may 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 public keys and pseudonymous biometrics, that the DID subject or a DID delegate can use to authenticate itself and prove its association with the DID. A DID document may also contain other attributes or claims describing the DID subject. A DID document may have one or more different representations as defined in § 6. Core 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 scheme must be implemented to work with a specific verifiable data registry. A DID method is defined by a DID method specification, which must specify the precise operations by which DIDs are created, resolved and deactivated and DID documents are written and updated. See § 7. 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 or a fragment hash sign (#) character (or the end of the DID URL). DID path syntax is identical to URI path syntax. See § 3.2.2 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 § 3.2.3 Query.

DID resolution

The function that takes as its input a DID and a set of input metadata and returns a DID document in a conforming representation plus additional metadata. This function relies on the "Read" operation of the applicable DID method. The inputs and outputs of this function are defined in § 8. 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 the section of the DID Core specification. Each DID method specification must define a specific DID scheme that works with that specific DID method. In a specific DID method scheme, the DID method name must follow the first colon and terminate with the second colon, e.g., `did:example:`

DID subject

The entity identified by a DID and described by a DID document. A DID has exactly one DID subject. 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, optional DID query (and its leading `?` character), and optional DID fragment (and its leading `#` character).

DID URL dereferencing

The function that takes as its input a DID URL, a DID document, plus a set of dereferencing options, and returns a resource. This resource may be a DID document plus additional metadata, or it may be a secondary resource contained within the DID document, or it may be a resource entirely external to the DID document. If the function begins with a DID URL, it use the DID resolution function to fetch a DID document indicated by the DID contained within the DID URL. The dereferencing function then can perform additional processing on the DID document to return the dereferenced resource indicated by the DID URL. The inputs and outputs of this function are defined in § 8.2 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 distributed database in which the various nodes use a consensus protocol to maintain a shared ledger in which each transaction is cryptographically signed and chained to the previous transaction.

proof purpose

A property of a DID document that communicates the purpose for which the DID controller included a specific type of proof. It acts as a safeguard to prevent the proof from being misused for a purpose other than the one it was intended for.

public key description

A data object contained inside a DID document that contains all the metadata necessary to use a public key or 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 may 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. The § 6. Core Representations section of the DID Core specification defines several representation formats for a DID document.

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 [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 may also be used for other cryptographically-verifiable data structures such as verifiable credentials. For more information, see [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 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 or protocol to independently verify a proof. For example, a 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 private key.

"Verification" and "proof" in this definition are intended to apply broadly. For example, a 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 § 5.4.1 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 abstract data model. When [INFRA] terminology is used, such as string, ordered 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.

3.1 DID Syntax §

The generic DID scheme is a URI scheme conformant with [RFC3986].

The DID scheme name *MUST* be an ASCII lowercase string.

The DID method name *MUST* be an ASCII lowercase string.

The following is the ABNF definition using the syntax in [RFC5234], which defines `ALPHA` and `DIGIT`. All other rule names not defined in this ABNF are defined in [RFC3986].

```
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 / "." / "-" / "_"
```

A DID method specification *MUST* further restrict the generic DID syntax by defining its own `method-name` and its own `method-specific-id` syntax. Case sensitivity and normalization of the value of the `method-specific-id` rule *MUST* be defined by the governing DID method specification. For more information, see Section § 7. Methods .

> NOTE: Persistence
>
> A DID is expected to be persistent and immutable. That is, a DID is bound exclusively and permanently to its one and only subject. Even after a DID is deactivated, it is intended that it never be repurposed.
>
> Ideally, a DID would be a completely abstract decentralized identifier (like a UUID) that could be bound to multiple underlying verifiable data registries over time, thus maintaining its persistence independent of any particular system. However, registering the same identifier on multiple verifiable data registries makes it extremely difficult to identify the authoritative version of a DID document if the contents diverge between the different verifiable data registries. It also greatly increases implementation complexity for developers.
>
> To avoid these issues, developers should refer to the Decentralized Characteristics Rubric [DID-RUBRIC] to decide which DID method best addresses the needs of the use case.

3.2 DID URL Syntax §

A DID URL always identifies a resource to be located. It can be used, for example, to identify a specific part of a DID document.

This 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 identical to the ABNF rules defined in [RFC3986].

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

> NOTE
>
> This specification reserves the semicolon (`;`) character for possible future use as a sub-delimiter for parameters as described in [MATRIX-URIS].

3.2.1 DID Parameters §

The DID URL syntax supports a simple format for parameters based on the `query` component (See § 3.2.3 Query).

Some DID parameter names (for example, for service selection) are completely independent of any specific DID method and *MUST* always function the same way for all DIDs. Other DID parameter names (for example, for versioning) *MAY* be supported by certain DID methods, but *MUST* operate uniformly across those DID methods that do support them.

The following table defines a set of DID parameter names.

Parameter Name	Description
`hl`	A resource hash of the DID document to add integrity protection, as specified in [HASHLINK]. The associated value *MUST* be an ASCII string. *This parameter is non-normative*.
`service`	Identifies a service from the DID document by service ID. The associated value *MUST* be an ASCII string.
`version-id`	Identifies a specific version of a DID document to be resolved (the version ID could be sequential, or a UUID, or method-specific). Note that this parameter might not be supported by all DID methods. The associated value *MUST* be an ASCII string.
`version-time`	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. Note that this parameter might not be supported by all DID methods. 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, as indicated by the trailing "Z".
`relative-ref`	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. The associated value *MUST* be an ASCII string and *MUST* use percent-encoding for certain characters as specified in RFC3986 Section 2.1.

Implementers as well as DID method specification authors *MAY* use additional DID parameters that are not listed here. For maximum interoperability, it is *RECOMMENDED* that DID parameters use the official W3C DID Specification Registries mechanism [DID-SPEC-REGISTRIES], to avoid collision with other uses of the same DID parameter with different semantics.

Additional considerations for processing these parameters are discussed in [DID-RESOLUTION].

Two example DID URLs using the `service` and `version-time` DID parameters are shown below.

> EXAMPLE 3: A DID URL with a 'service' DID parameter
>
> ```
> did:foo:21tDAKCERh95uGgKbJNHYp?service=agent
> ```

> EXAMPLE 4: A DID URL with a 'version-time' DID parameter
>
> ```
> did:foo:21tDAKCERh95uGgKbJNHYp?version-time=2002-10-10T17:00:00Z
> ```

Adding a DID parameter to a DID URL means that the parameter becomes part of an identifier for a resource (the DID document or other). Alternatively, the DID resolution and the DID URL dereferencing functions can also be influenced by passing input metadata to a DID resolver that are not part of the DID URL. (See § 8.1.1 DID Resolution Input Metadata Properties). Such input metadata could for example control caching or the desired encoding of a resolution result. 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*.

DID parameters *MAY* be used if there is a clear use case where the parameter needs to be part of a URI that can be used as a link, or as a resource in RDF / JSON-LD documents.

DID parameters *SHOULD NOT* be used if the same functionality can be expressed by passing input metadata to a DID resolver, and if there is no need to construct a URI for use as a link, or as a resource in RDF / JSON-LD documents.

3.2.2 Path §

A DID path is identical to a generic URI path and *MUST* conform to the `path-abempty` ABNF rule in [RFC3986].

A DID method specification *MAY* specify ABNF rules for DID paths that are more restrictive than the generic rules in this section.

EXAMPLE 5

```
did:example:123456/path
```

3.2.3 Query §

A DID query is derived from a generic URI query and *MUST* conform to the `did-query` ABNF rule in Section § 3.2 DID URL Syntax. If a DID query is present, it *MUST* be used as described in Section § 3.2.1 DID Parameters.

A DID method specification *MAY* specify ABNF rules for DID queries that are more restrictive than the generic rules in this section.

EXAMPLE 6

```
did:example:123456?query=true
```

3.2.4 Fragment §

A DID fragment is used as method-independent reference into the DID document to identify a component of the document (for example, a unique public key description or service endpoint). DID fragment syntax and semantics are identical to a generic URI fragment and *MUST* conform to RFC 3986, section 3.5. To dereference a DID fragment, the complete DID URL including the DID fragment *MUST* be used as input to the DID URL dereferencing function for the target component in the DID document object. For more information, see § 8.2 DID URL Dereferencing .

A DID method specification *MAY* specify ABNF rules for DID fragments that are more restrictive than the generic rules in this section.

EXAMPLE 7

```
did:example:123456#public-key-1
```

In order to maximize interoperability, implementers are urged to ensure that DID fragments are interpreted in the same way across representations (as described in § 6. Core Representations). For example, while JSON Pointer [RFC6901] can be used in a DID fragment, it will not be interpreted in the same way across 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 Section § B.2 application/did+ld+json.

3.2.5 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 Section § 3.1 DID Syntax. The contents of the URL typically refers to 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 Section § 5.1 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 Section § 3.2.2 Path, Section § 3.2.3 Query, and Section § 3.2.4 Fragment, respectively.

Relative DID URLs are often used to identify verification methods and services in a DID Document without having to use absolute URLs, which tend to be more verbose than necessary.

EXAMPLE 8: An example of a relative DID URL

```
{
  "@context": "https://www.w3.org/ns/did/v1",
  "id": "did:example:123456789abcdefghi",
  "verificationMethod": [{
    "id": "did:example:123456789abcdefghi#key-1",
    "type": "Ed25519VerificationKey2018",
    "controller": "did:example:123456789abcdefghi",
    "publicKeyBase58": "H3C2AVvLMv6gmMNam3uVAjZpfkcJCwDwnZn6z3wXmqPV"
  }, ...],
  "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 an abstract data model for DID documents, independent of any specific representation. This section provides a high-level description of the data model, a set of requirements for representations, and a set of requirements for extensibility.

4.1 Definition §

A DID document consists of a map of properties, which are name-value pairs (ie. a property name, and a property value). The definitions of each of these properties are specified in section § 5. Core Properties. Specific representations are defined in section § 6. Core Representations.

4.2 Representations §

Following are the requirements for representations.

1. A representation *MUST* define an unambiguous encoding and decoding of all property names and their associated values as defined in this specification. This means anything you can represent in the DID document data model can be represented in any compliant representation.
2. The representation *MUST* be associated with an IANA-registered MIME type.
3. The representation *MUST* define fragment processing rules for its MIME type that are conformant with the fragment processing rules defined in section § 3.2.4 Fragment of this specification.

The core representations are specified in section § 6. Core Representations.

4.3 Extensibility §

The data model supports two types of extensibility.

1. For maximum interoperability, it is *RECOMMENDED* that extensions use the official W3C DID Specification Registries mechanism [DID-SPEC-REGISTRIES]. The use of this mechanism for new properties or other extensions is the only specified method that ensures that two different representations will be able to work together.
2. Representations *MAY* define other extensibility mechanisms including methods for decentralized extensions. Such extension mechanisms *MUST* support lossless conversion into any other conformant representation.

> NOTE
>
> 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 Core Registries [DID-SPEC-REGISTRIES]; interoperability between such implementations and the larger ecosystem will be less reliable.

5. Core Properties §

A DID points to a DID document. DID documents are the serialization of the data model outlined in Section § 4. Data Model. 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.

For reference, the core properties found at the top level of a DID document are as follows. Properties belonging to other objects referenced in the DID document are also listed, with their respective top-level property.

- `id`: defined in § 5.1 DID Subject
- `alsoKnownAs`: defined in § 5.1.1 alsoKnownAs
- `controller`: defined in § 5.2 Control
- `verificationMethod`: defined in § 5.3 Verification Methods. Sub-properties include `id`, `type`, `controller`.
- `authentication`: defined in § 5.4.1 authentication.
- `assertionMethod`: defined in § 5.4.2 assertionMethod.
- `keyAgreement`: defined in § 5.4.3 keyAgreement.
- `capabilityDelegation`: defined in § 5.4.5 capabilityDelegation.
- `capabilityInvocation`: defined in § 5.4.4 capabilityInvocation.
- `service`: defined in § 5.5 Service Endpoints. Sub-properties include `id`, `type` and `serviceEndpoint`.

5.1 DID Subject §

The DID subject is denoted with the `id` property. The DID subject is the entity that the DID document is about. That is, it is the entity identified by the DID and described by the DID document.

DID documents *MUST* include the `id` property.

id

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

EXAMPLE 9

```
{
  "id": "did:example:21tDAKCERh95uGgKbJNHYp"
}
```

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. The value of `id` in the resolved DID document *MUST* match the DID that was resolved, or be populated with the equivalent canonical DID specified by the DID method, which *SHOULD* be used by the resolving party going forward.

5.1.1 alsoKnownAs §

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) identify the same DID subject can be made using the `alsoKnownAs` property.

DID documents *MAY* include the `alsoKnownAs` property.

alsoKnownAs

The value of `alsoKnownAs` *MUST* be a list where each item in the list 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 graphs of the two corresponding DID documents, is not necessarily appropriate, *even with* a reciprocal relationship.

5.2 Control §

Authorization is the mechanism used to state how operations are performed on behalf of the DID subject. A DID controller is authorized to make changes to the respective DID document.

A DID document *MAY* include a `controller` property to indicate the DID controller(s). If so:

controller

The value of the `controller` property *MUST* be a string or an ordered set of strings that conform to the rules in Section § 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.

> NOTE: Authorization vs authentication
>
> Note that Authorization is separate from § 5.4.1 authentication. This is particularly important for key recovery in the case of key loss, when the 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 Section § 9. Security Considerations .

EXAMPLE 10: DID document with a controller property

```
{
  "@context": "https://www.w3.org/ns/did/v1",
  "id": "did:example:123456789abcdefghi",
  "controller": "did:example:bcehfew7h32f32h7af3",
  "service": [{
    // used to retrieve Verifiable Credentials
    associated with the DID
    "type": "VerifiableCredentialService",
    "serviceEndpoint": "https://example.com/vc/"
  }]
}
```

5.3 Verification Methods §

A DID document can express verification methods, such as cryptographic keys, which can be used to authenticate or authorize interactions with the DID subject or associated parties. The information expressed often includes globally unambiguous identifiers and public key material, which can be used to verify digital signatures. For example, a 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 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 threshold signature. Methods need not be cryptographic. An example of this might be the contact information for a biometric service provider that compares a purported DID controller against a candidate biometric vector.

In order to maximize interoperability, support for public keys as verification methods is restricted: see § 5.3.1 Key types and formats. For other types of verification method, the verification method *SHOULD* be registered in the [DID-SPEC-REGISTRIES].

A DID document *MAY* include a `verificationMethod` property.

verificationMethod

If a DID document includes a `verificationMethod` property, the value of the property *MUST* be an ordered set of verification methods, where each verification method is described by a map containing properties. The properties *MUST* include the `id`, `type`, `controller`, and specific verification method properties, and *MAY* include additional properties.

The value of the `id` property for a verification method *MUST* be a URI. When more than one verification method is present, the value of `verificationMethod` *MUST NOT* contain multiple entries with the same `id`. If the value of `verificationMethod` contains multiple entries with the same `id`, a DID document processor *MUST* produce an error.

In the case where a verification method is a public key, the value of the `id` property *MAY* be structured as a compound key. This is especially useful for integrating with existing key management systems and key formats such as JWK [RFC7517]. It is *RECOMMENDED* that JWK `kid` values are set to the public key fingerprint [RFC7638]. It is *RECOMMENDED* that verification methods that use JWKs to represent their public keys utilize the value of `kid` as their fragment identifier. See the first key in Example 13 for an example of a public key with a compound key identifier.

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

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

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 public key. Since a key (for example) 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 controller(s) are expressed using the `controller` property on the *top level* of the DID document; see Section § 5.2 Control .

EXAMPLE 11: Example verification methods

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

As well as the `verificationMethod` property, verification methods can be embedded in or referenced from properties associated with various verification relationships (see § 5.4 Verification Relationships. Referencing verification methods allows them to be used with more than one verification relationship.

The steps to use when processing a verification method in a DID document are:

1. Let *value* be the data associated with the `verificationMethod` property or property for a particular verification relationship and initialize *result* to `null`.
2. If *value* is an object, the verification method material is embedded. Set *result* to *value*.
3. If *value* is a string, the verification method is included by reference. Assume *value* is a URL.
    1. Dereference the URL and retrieve the `verificationMethod` properties associated with the URL. For example, process the `verificationMethod` property at the top-level of the dereferenced document.
    2. Iterating through each object, if the `id` property of the object matches *value*, set *result* to the object.
4. If *result* does not contain at least the `id`, `type`, and `controller` properties, as well as any mandatory public cryptographic material, as determined by the `type` property of *result*, throw an error.

EXAMPLE 12: Embedding and referencing verification methods

```
{
...

  "authentication": [
    // this key is referenced, it may be used with more than one verificat
    "did:example:123456789abcdefghi#keys-1",
    // this key is embedded and may *only* be used for authentication
    {
      "id": "did:example:123456789abcdefghi#keys-2",
      "type": "Ed25519VerificationKey2018",
      "controller": "did:example:123456789abcdefghi",
      "publicKeyBase58": "H3C2AVvLMv6gmMNam3uVAjZpfkcJCwDwnZn6z3wXmqPV"
    }
  ],

...
}
```

5.3.1 Key types and formats §

A public key can be used as a verification method.

A verification method *MUST NOT* contain multiple verification material properties. For example, expressing key material in a verification method using both `publicKeyJwk` and `publicKeyBase58` at the same time is prohibited.

This specification strives to limit the number of formats for expressing public key material in a DID document to the fewest possible, to increase the likelihood of interoperability. 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. The specific types of key formats that are supported in this specification are listed here.

When using any of the public key types described here, public key expression *MUST NOT* use any other key format than those listed in the Public Key Support table. For public key types that are not listed here, the type value and corresponding format property *SHOULD* be registered in [DID-SPEC-REGISTRIES], as with any other verification method.

> ISSUE
>
> The Working Group is still debating whether the base encoding format used will be Base58 (Bitcoin) [BASE58], base64url [RFC7515], or base16 (hex) [RFC4648]. The entries in the table below currently assume PEM and Base58 (Bitcoin), but might change to base64url and/or base16 (hex) after the group achieves consensus on this particular issue.

> ISSUE
>
> The Working Group is still debating whether secp256k1 Schnorr public key values will be elaborated upon in this specification and if so, how they will be expressed and encoded.

This table defines the support for public key expression in a DID document. For each public key type, a maximum of two encoding formats are supported.

Key Type (`type` value)	Support
RSA (`RsaVerificationKey2018`)	RSA public key values *MUST* either be encoded as a JWK [RFC7517] or be encoded in Privacy Enhanced Mail (PEM) format using the `publicKeyPem` property.
ed25519 (`Ed25519VerificationKey2018`)	Ed25519 public key values *MUST* either be encoded as a JWK [RFC7517] or be encoded as the raw 32-byte public key value in Base58 Bitcoin format [BASE58] using the `publicKeyBase58` property.
secp256k1-koblitz (pending)	Secp256k1 Koblitz public key values *MUST* either be encoded as a JWK [RFC7517] or be encoded as the raw 33-byte public key value in Base58 Bitcoin format [BASE58] using the `publicKeyBase58` property.
secp256r1 (`SchnorrSecp256k1VerificationKey2019`)	Secp256r1 public key values *MUST* either be encoded as a JWK [RFC7517] or be encoded as the raw 32-byte public key value encoded in Base58 Bitcoin format [BASE58] using the `publicKeyBase58` property.

Key Type (`type` value)	Support
Curve25519 (`X25519KeyAgreementKey2019`)	Curve25519 (also known as X25519) public key values *MUST* either be encoded as a JWK [RFC7517] or be encoded as the raw 32-byte public key value in Base58 Bitcoin format [BASE58] using the `publicKeyBase58` property.
JWK (`JsonWebKey2020`)	Key types listed in JOSE, represented using [RFC7517] using the `publicKeyJwk` property.

Example:

EXAMPLE 13: Various public keys

```
{
  "@context": ["https://www.w3.org/ns/did/v1", "https://w3id.org/security/
  "id": "did:example:123456789abcdefghi",
  ...
  "verificationMethod": [{
    "id": "did:example:123#_Qq0UL2Fq651Q0Fjd6TvnYE-faHiOpRlPVQcY_-tA4A",
    "type": "JsonWebKey2020",
    "controller": "did:example:123",
    "publicKeyJwk": {
      "crv": "Ed25519",
      "x": "VCpo2LMLhn6iWku8MKvSLg2ZAoC-nlOyPVQaO3FxVeQ",
      "kty": "OKP",
      "kid": "_Qq0UL2Fq651Q0Fjd6TvnYE-faHiOpRlPVQcY_-tA4A"
    }
  }, {
    "id": "did:example:123456789abcdefghi#keys-1",
    "type": "Ed25519VerificationKey2018",
    "controller": "did:example:pqrstuvwxyz0987654321",
    "publicKeyBase58": "H3C2AVvLMv6gmMNam3uVAjZpfkcJCwDwnZn6z3wXmqPV"
  }, {
    "id": "did:example:123456789abcdefghi#keys-2",
    "type": "Secp256k1VerificationKey2018",
    "controller": "did:example:123456789abcdefghi",
    "publicKeyHex": "02b97c30de767f084ce3080168ee293053ba33b235d7116a3263
  }],
  ...
}
```

If a public key does not exist in the DID document, it *MUST* be assumed the key was revoked or is invalid. The DID document *MUST NOT* express revoked keys using a verification relationship. Each DID method specification is expected to detail how revocation is performed and tracked.

NOTE

Caching and expiration of the keys in a DID document is entirely the responsibility of DID resolvers and requesting parties. For more information, see Section § 8. Resolution .

5.4 Verification Relationships §

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

A DID document *MAY* include a property expressing a specific verification relationship. In order to maximize global interoperability, the property *SHOULD* be registered in [DID-SPEC-REGISTRIES].

The information in a DID document *MUST* be explicit about the verification relationship between the DID subject and the verification method. Verification methods that are not associated with a particular verification relationship *MUST NOT* be used for that verification relationship. See the sections below for more detailed examples of a verification relationship.

5.4.1 authentication §

Authentication is a verification relationship which an entity can use to prove it is the DID subject or acting on behalf of the DID Subject as a DID Controller. The *verifier* of an authentication attempt can check if the authenticating party is presenting a valid proof of authentication, that is, that they are who they say they are. Note that a successful authentication on its own might or might not confer authority; that is up to the verifying application.

> NOTE: Uses of authentication
>
> 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 DID data model, but DID methods and applications are expected to define this themselves.

A DID document *MAY* include an `authentication` property. The `authentication` property is a relationship between the DID subject and a set of verification methods (such as, but not limited to, public keys). It means that the DID subject has authorized some set of verification methods (per the value of the `authentication` property) for the purpose of authentication.

authentication

The associated value *MUST* be an ordered set of one or more verification methods. Each verification method *MAY* be embedded or referenced.

This statement 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.

The verification method indicated by the `authentication` property of a DID document can only be used to authenticate the DID subject. To authenticate the DID controller (in cases where the DID controller is not *also* the DID subject) the entity associated with the value of `controller` (see Section § 5.2 Control) needs to authenticate itself with its *own* DID document and attached `authentication` verification relationship.

Example:

EXAMPLE 14: Authentication property containing three verification methods

```
{
  "@context": "https://www.w3.org/ns/did/v1",
  "id": "did:example:123456789abcdefghi",
  ...
  "authentication": [
    // this method can be used to authenticate as did:...fghi
    "did:example:123456789abcdefghi#keys-1",
    // this method can be used to authenticate as did:...fghi
    "did:example:123456789abcdefghi#biometric-1",
    // this method is *only* authorized 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": "Ed25519VerificationKey2018",
      "controller": "did:example:123456789abcdefghi",
      "publicKeyBase58": "H3C2AVvLMv6gmMNam3uVAjZpfkcJCwDwnZn6z3wXmqPV"
    }
  ],
  ...
}
```

5.4.2 assertionMethod §

The `assertionMethod` property is used to express a verification relationship which indicates that a verification method can be used to verify a proof that a statement was asserted on behalf of the DID

subject. A *verifier* of such a proof can ensure that a verification method used with the proof was authorized to be used with proofs for that purpose by checking that the verification method is contained in the `assertionMethod` of the DID Document.

> NOTE: Uses of assertionMethod
>
> If `assertionMethod` is established, it is up to the verifier to validate that the verification method used for providing proof of an assertion is valid and is associated with the `assertionMethod` verification relationship. An example of when this property is useful is during the processing of a verifiable credential by a verifier. During validation, a verifier checks to see if a verifiable credential has been signed 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.

A DID document *MAY* include an `assertionMethod` property.

assertionMethod

The associated value *MUST* be an ordered set of one or more verification methods. Each verification method *MAY* be embedded or referenced.

Example:

EXAMPLE 15: Assertion method property containing two verification methods

```
{
  "@context": "https://www.w3.org/ns/did/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* authorized for assertion of statements, it ma
    // be used for any other verification relationship, so its full descri
    // embedded here rather than using only a reference
    {
      "id": "did:example:123456789abcdefghi#keys-2",
      "type": "Ed25519VerificationKey2018",
      "controller": "did:example:123456789abcdefghi",
      "publicKeyBase58": "H3C2AVvLMv6gmMNam3uVAjZpfkcJCwDwnZn6z3wXmqPV"
    }
  ],
  ...
}
```

5.4.3 keyAgreement §

The `keyAgreement` property is used to express a verification relationship which an entity can use to engage in key agreement protocols on behalf of the DID subject. The *counterparties* in a key agreement protocol can use the `keyAgreement` verification relationship to check whether a party performing a key agreement protocol on behalf of the DID subject is authorized by checking if the verification method used during the key agreement protocol is associated with the `keyAgreement` property contained in the DID Document.

A DID document *MAY* include an `keyAgreement` property.

keyAgreement

The associated value *MUST* be an ordered set of one or more verification methods. Each verification method *MAY* be embedded or referenced.

NOTE: Uses of keyAgreement

It is up to a verifier to validate that the verification method used during a key agreement exchange is valid and is associated with the `keyAgreement` property. An example of when this property is useful is during the establishment of a TLS session on behalf of the DID Subject. In this case, the counterparty checks that the verification method used during the protocol handshake is associated with the `keyAgreement` property in the DID Document.

Example:

EXAMPLE 16: 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* authorized for key agreement usage, it may nc
    // be used for any other verification relationship, so its full descr
    // embedded here rather than using only a reference
    {
      "id": "did:example:123#zC9ByQ8aJs8vrNXyDhPHHNNMSHPcaSgNpjjsBYpMMjsTc
      "type": "X25519KeyAgreementKey2019",
      "controller": "did:example:123",
      "publicKeyBase58": "9hFgmPVfmBZwRvFEyniQDBkz9LmV7gDEqytWyGZLmDXE"
    }
  ],
  ...
}
```

5.4.4 capabilityInvocation §

The `capabilityInvocation` property is used to express a verification relationship which an entity can use to invoke capabilities as the DID subject or on behalf of the DID subject. A capability is an expression of an action that the DID subject is authorized to take. The *verifier* of a capability invocation attempt can check the validity of a capability by checking if the verification method used with the proof is contained in the `capabilityInvocation` property of the DID Document.

A DID document *MAY* include an `capabilityInvocation` property.

capabilityInvocation

The associated value *MUST* be an ordered set of one or more verification methods. Each verification method *MAY* be embedded or referenced.

> NOTE: Uses of capabilityInvocation
>
> It is the responsibility of a verifier to ensure that the verification method used when presenting a capability is invoked and is associated with the `capabilityInvocation` property. An example of when this property is useful is when a DID subject chooses to invoke their capability to start a vehicle through the combined usage of a verification method and the `StartCar` capability. In this example, the vehicle would be the *verifier* and would need to verify that the verification method exists in the `capabilityInvocation` property.

Example:

EXAMPLE 17: Capability invocation property containing two verification methods

```
{
  "@context": "https://www.w3.org/ns/did/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* authorized for capability invocation usage,
    // be used for any other verification relationship, so its full descr
    // embedded here rather than using only a reference
    {
    "id": "did:example:123456789abcdefghi#keys-2",
    "type": "Ed25519VerificationKey2018",
    "controller": "did:example:123456789abcdefghi",
    "publicKeyBase58": "H3C2AVvLMv6gmMNam3uVAjZpfkcJCwDwnZn6z3wXmqPV"
    }
  ],
  ...
}
```

5.4.5 capabilityDelegation §

The `capabilityDelegation` property is used to express a verification relationship which an entity can use to grant capabilities as the DID subject or on behalf of the DID subject to other *capability invokers*. The *verifier* of a `capabilityDelegation` attempt can check the validity of a capability to grant invocation of a capability by checking if the verification method used with the proof is contained in the `capabilityDelegation` section of the DID Document.

A DID document *MAY* include an `capabilityDelegation` property.

capabilityDelegation

The associated value *MUST* be an ordered set of one or more verification methods. Each verification method *MAY* be embedded or referenced.

> NOTE: Uses of capabilityDelegation
>
> It is up to a verifier to validate that the verification method used when presenting a capability is valid and is associated with the `capabilityDelegation` property. An example of when this property is useful is when a DID Subject chooses to grant their capability to start a vehicle through the combined usage of a verification method and the `StartCar` capability to a capability invoker.

Example:

EXAMPLE 18: Capability Delegation property containing two verification methods

```
{
  "@context": "https://www.w3.org/ns/did/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* authorized for granting capabilities it may n
    // be used for any other verification relationship, so its full descri
    // embedded here rather than using only a reference
    {
    "id": "did:example:123456789abcdefghi#keys-2",
    "type": "Ed25519VerificationKey2018",
    "controller": "did:example:123456789abcdefghi",
    "publicKeyBase58": "H3C2AVvLMv6gmMNam3uVAjZpfkcJCwDwnZn6z3wXmqPV"
    }
  ],
  ...
}
```

5.5 Service Endpoints §

Service endpoints are used in DID documents to express ways of communicating with the DID subject or associated entities. Services listed in the DID document can contain information about privacy preserving messaging services, or more public information, such as social media accounts, personal websites, and email addresses although this is discouraged. See § 10.1 Keep Personally-Identifiable Information (PII) Private for additional details. The metadata associated with services are often service-specific. For example, the metadata associated with an encrypted messaging service can express how to initiate the encrypted link before messaging begins.

Pointers to services are expressed using the `service` property. Each service has its own `id` and `type` properties, as well as a `serviceEndpoint` property with a URI or a set of other properties describing the service.

One of the primary purposes of a DID document is to enable discovery of service endpoints. A service endpoint can be any type of service the DID subject wants to advertise, including decentralized identity management services for further discovery, authentication, authorization, or interaction.

A DID document *MAY* include a `service` property.

service

If a DID document includes a `service` property, the value of the property *SHOULD* be an unordered set of service endpoints, where each service endpoint is described by a set of properties. Each service endpoint *MUST* have `id`, `type`, and `serviceEndpoint` properties, and *MAY* include additional properties.

The value of the `id` property *MUST* be a URI. The value of `service` *MUST NOT* contain multiple entries with the same `id`. In this case, a DID document processor *MUST* produce an error.

The value of the ***`serviceEndpoint`*** property *MUST* be a valid URI conforming to [RFC3986] and normalized according to the rules in section 6 of [RFC3986] and to any normalization rules in its applicable URI scheme specification, *OR* a set of properties which describe the service endpoint further.

It is expected that the service endpoint protocol is published in an open standard specification.

EXAMPLE 19: Various service endpoints

```
{
  "service": [{
    "id": "did:example:123456789abcdefghi#openid",
    "type": "OpenIdConnectVersion1.0Service",
    "serviceEndpoint": "https://openid.example.com/"
  }, {
    "id": "did:example:123456789abcdefghi#vcr",
    "type": "CredentialRepositoryService",
    "serviceEndpoint": "https://repository.example.com/service/8377464"
  }, {
    "id": "did:example:123456789abcdefghi#xdi",
    "type": "XdiService",
    "serviceEndpoint": "https://xdi.example.com/8377464"
  }, {
    "id": "did:example:123456789abcdefghi#agent",
    "type": "AgentService",
    "serviceEndpoint": "https://agent.example.com/8377464"
  }, {
    "id": "did:example:123456789abcdefghi#hub",
    "type": "IdentityHub",
    "verificationMethod": "did:example:123456789abcdefghi#key-1",
    "serviceEndpoint": {
      "@context": "https://schema.identity.foundation/hub",
      "type": "UserHubEndpoint",
      "instances": ["did:example:456", "did:example:789"]
    }
  }, {
    "id": "did:example:123456789abcdefghi#messages",
    "type": "MessagingService",
    "serviceEndpoint": "https://example.com/messages/8377464"
  }, {
    "id": "did:example:123456789abcdefghi#inbox",
    "type": "SocialWebInboxService",
    "serviceEndpoint": "https://social.example.com/83hfh37dj",
    "description": "My public social inbox",
    "spamCost": {
      "amount": "0.50",
      "currency": "USD"
    }
  }, {
    "id": "did:example:123456789abcdefghi#authpush",
    "type": "DidAuthPushModeVersion1",
```

```
    "serviceEndpoint": "http://auth.example.com/did:example:123456789abcd
  }]
}
```

For more information about security considerations regarding authentication service endpoints see Sections § 7.1 Method Schemes and § 5.4.1 authentication.

6. Core Representations §

All concrete representations of a DID document *MUST* be serialized using a deterministic mapping that is able to be unambiguously parsed into the data model defined in this specification. All serialization methods *MUST* define rules for the bidirectional translation of a DID document both into and out of the representation in question. As a consequence, translation between any two representations *MUST* be done by parsing the source format into a DID document model (described in Sections § 4. Data Model and § 5. Core Properties) and then serializing the DID document model into the target representation. An implementation *MUST NOT* convert between representations without first parsing to a DID document model.

Although syntactic mappings are provided for JSON, JSON-LD, and CBOR here, applications and services *MAY* use any other data representation syntax that is capable of expressing the data model, such as XML or YAML.

Producers *MUST* indicate which representation of a document has been used via a media type in the document's metadata. Consumers *MUST* determine which representation a document is in via the `content-type` DID resolver metadata field. (See § 8.1 DID Resolution). Consumers *MUST NOT* determine the representation of a document through its content alone.

> ISSUE 203: Define DID Document Metadata **metadata** **pending close**
>
> This requirement depends on the return of DID document metadata that still needs to be defined by this specification. Once defined, that should be linked from here.

The production and consumption rules in this section apply to all implementations seeking to be fully compatible with independent implementations of the specification. Deployments of this specification *MAY* use a custom agreed-upon representation, including localized rules for handling properties not listed in the registry. See section § 4.3 Extensibility for more information.

~~ISSUE 207~~: Add section on extensibility and conformance editorial **extensibility** **pending close**

A link to a section on extensibility and conformance as it applies to data representations should be added here once that section has been written.

6.1 JSON §

When producing and consuming DID documents that are in plain JSON (as indicated by a `content-type` of `application/did+json` in the resolver metadata), the following rules *MUST* be followed.

6.1.1 Production §

A DID document *MUST* be a single JSON object conforming to [RFC8259]. All top-level properties of the DID document *MUST* be represented by using the property name as the name of the member of the JSON object. The values of properties of the data model described in Section § 4. Data Model, including all extensions, *MUST* be encoded in JSON [RFC8259] by mapping property values to JSON types as follows:

- Numeric values representable as IEEE754 *MUST* be represented as a Number type.
- Boolean values *MUST* be represented as a Boolean literal.
- Sequence value *MUST* be represented as an Array type.
- Unordered sets of values *MUST* be represented as an Array type.
- Sets of properties *MUST* be represented as an Object type.
- Empty values *MUST* be represented as a null literal.
- Other values *MUST* be represented as a String type.

~~ISSUE 204~~: Define terminology for properties and values **PR exists** editorial

An "empty" value is not specified by this document. It seems to imply a null value, but this is unclear.

Implementers producing JSON are advised to ensure that their algorithms are aligned with the JSON serialization rules in the [INFRA] specification.

All properties of the DID document *MUST* be included in the root object. Properties *MAY* define additional data sub structures subject to the value representation rules in the list above.

The member name `@context` *MUST NOT* be used as this property is reserved for JSON-LD producers.

6.1.2 Consumption §

> ~~ISSUE 204~~: Define terminology for properties and values **PR exists** editorial
>
> In this section and we use the term "property name" to refer to the string that represents the property itself, but this specification still needs to define a concrete term for such aspects of a property of a DID document. We also need a concrete term for "the document itself" as opposed to "the collection or properties of the document".

The top-level element *MUST* be a JSON object. Any other data type at the top level is an error and *MUST* be rejected. The top-level JSON object represents the DID document, and all members of this object are properties of the DID document. The object member name is the property name, and the member value is interpreted as follows:

- Number types *MUST* interpreted as numeric values representable as IEEE754.
- Boolean literals *MUST* be interpreted as a Boolean value.
- An Array type *MUST* be interpreted as a Sequence or Unordered set, depending on the definition of the property for this value.
- An Object type *MUST* be interpreted as a sets of properties.
- A null literal *MUST* be interpreted as an Empty value.
- String types *MUST* be interpreted as Strings, which may be further parsed depending on the definition of the property for this value into more specific data types such as URIs, date stamps, or other values.

Implementers consuming JSON are advised to ensure that their algorithms are aligned with the JSON consumption rules in the [INFRA] specification.

> ~~ISSUE 204~~: Define terminology for properties and values **PR exists** editorial
>
> An "empty" value is not specified by this document. It seems to imply a null value, but this is unclear.

The value of the `@context` object member *MUST* be ignored as this is reserved for JSON-LD consumers.

Unknown object member names *MUST* be ignored as unknown properties.

> ISSUE 205: How to treat unknown properties discuss **extensibility**
>
> This specification needs to define clear and consistent rules for how to handle unknown data members on consumption, and this section needs to be updated with that decision.

6.2 JSON-LD §

[JSON-LD] is a JSON-based format used to serialize Linked Data.

When producing and consuming DID documents that are in JSON-LD (as indicated by a `content-type` of `application/did+ld+json` in the resolver metadata), the following rules *MUST* be followed.

- The `@id` and `@type` keywords are aliased to `id` and `type` respectively, enabling developers to use this specification as idiomatic JSON.
- Even though JSON-LD allows any IRI as node identifiers, DID documents are explicitly restricted to only describe DIDs. This means that the value of `id` that refers to the DID subject *MUST* be a valid DID and not any other kind of IRI.
- Data types, such as integers, dates, units of measure, and URLs, are automatically typed to provide stronger type guarantees for use cases that require them.

6.2.1 Production §

The DID document is serialized following the rules in the JSON processor, with one addition: DID documents *MUST* include the `@context` property.

@context
: The value of the `@context` property *MUST* be one or more URIs, where the value of the first URI is `https://www.w3.org/ns/did/v1`. All members of the `@context` property *SHOULD* exist in the DID specification registries in order to achieve interoperability across different representations. If a member does not exist in the DID specification registries, then the DID Document will not be interoperable across representations.

~~ISSUE 202~~: JSON-LD Contexts in Registry **PR exists** **extensibility**

This specification defines globally interoperable documents, and the requirement that the context value be in the verifiable data registry means that different JSON-LD processors can consume the document without having to dereference anything in the context field. However, a pair of producers and consumers can have local extension agreements. This needs to be clarified in a section on extensibility and linked here when available.

6.2.2 Consumption §

The top-level element *MUST* be a JSON object. Any other data type at the top level is an error and *MUST* be rejected. This top-level JSON object is interpreted using JSON-LD processing under the rules of the defined `@context` fields.

@context

The value of the `@context` property *MUST* be one or more URIs, where the value of the first URI is `https://www.w3.org/ns/did/v1`. If more than one URI is provided, the URIs *MUST* be interpreted as an ordered set. It is *RECOMMENDED* that dereferencing each URI results in a document containing machine-readable information about the context. To enable interoperability with other representations, URLs registered in the DID Specification Registries [DID-SPEC-REGISTRIES] referring to JSON-LD Contexts *SHOULD* be associated with a cryptographic hash of the content of the JSON-LD Context. This ensures that the interpretation of the information by JSON-LD consumers will be the same as interpretations over other representations by other consumers that rely on the DID Specification Registries [DID-SPEC-REGISTRIES].

Unknown object member names *MUST* be ignored as unknown properties.

ISSUE 205: How to treat unknown properties **discuss** **extensibility**

This specification needs to define clear and consistent rules for how to handle unknown data members on consumption, and this section needs to be updated with that decision.

6.3 CBOR §

Like Javascript Object Notation (JSON) [RFC8259], Concise Binary Object Representation (CBOR) [RFC7049] defines a set of formatting rules for the portable representation of structured data. CBOR is a more concise, machine-readable, language-independent data interchange format that is self-describing and has built-in semantics for interoperability. With specific constraints, CBOR can support

all JSON data types (including JSON-LD) for translation between the DID document model (described in Data Model and DID Documents) and other core representations.

Concise Data Definition Language (CDDL) [RFC8610] is a notation used to express Concise Binary Object Representation (CBOR), and by extension JSON Data Structures. The following notation expresses the DID Document model in CBOR representation with specific constraints for deterministic mappings between other core representations.

EXAMPLE 20: DID Document data model for CBOR expressed in CDDL notation

```
DID-document = {
  ? @context : uri
  id : did
  ? publicKey : [* publicKey ]
  ? authentication :  [ *did // *publicKey // *tstr ]
  ? service : [ + service  ]
  ? controller : did / [ *did ]
  ? created : time
  ? updated : time
  proof :  any
}

publicKey = {
  id : did
  type : text
  controller : uri
}

did = tstr .pcre "^did\\:(?<method-name>[a-z0-9]{2,})\\:(?<method-specifi

did-url =  tstr .pcre "^did\\:(?<method-name>[a-z0-9]{2,})\\:(?<method-sp

service = {
  id : did-url
  type : text
  serviceEndpoint : uri
  ? description : text
  * tstr => any
}
```

6.3.1 Production §

When producing DID Documents that are represented as CBOR, in addition to the suggestions in section 3.9 of the CBOR [RFC7049] specification for deterministic mappings, the following constraints of the DID Document model *MUST* be followed:

- Map keys *MUST* be strings.
- Integer encoding *MUST* be as short as possible.
- The expression of lengths in CBOR major types 2 through 5 *MUST* be as short as possible.
- All floating point values *MUST* be encoded as 64-bits, even for integral values.

EXAMPLE 21: An example DID Document represented as contrained CBOR and exported in diagnostic annotation mode for easy readibility

```
a7                                                  # map(7)
62                                                  #   text(2)
   6964                                             #     "id"
78 40                                               #   text(64)
   6469643a6578616d706c653a31324433                 #     "did:example:12D3'
   4b6f6f574d4864727a6377706a626472                 #     "KooWMHdrzcwpjbdr'
   5a7335474771455241766367715833 62                #     "Zs5GGqERAvcgqX3b'
   356470755074506139 6f743639796577                #     "5dpuPtPa9ot69yew'
65                                                  #   text(5)
   70726f6f66                                       #     "proof"
a4                                                  #   map(4)
   64                                               #     text(4)
      74797065                                      #       "type"
   74                                               #     text(20)
      656432353531395369676e617475726532303138      #       "ed25519Signatu
   67                                               #     text(7)
      63726561746564                                #       "created"
   74                                               #     text(20)
      323032302d30352d30315430333a30303a30325a      #       "2020-05-01T03:0
   67                                               #     text(7)
      63726561746f72                                #       "creator"
   78 8c                                            #     text(140)
      6469643a6578616d706c653a31324433              #       "did:example:12D
      4b6f6f574d4864727a6377706a626472              #       "KooWMHdrzcwpjbd
      5a7335474771455241766367715833 62             #       "Zs5GGqERAvcgqX3
      356470755074506139 6f743639796577             #       "5dpuPtPa9ot69ye
      3b206578616d706c653a6b65793d6964              #       "; example:key=i
      3d626166797265696375627478357771              #       "=bafyreicubtx5v
      6f336e6f73633463617a726b63746668              #       "o3nosc4cazrkct
      776436726577657a6770776f65347377              #       "wd6rewezgpwoe4s
      69726c733465626468733269                       #       "irls4ebdhs2i"
   6e                                               #     text(14)
      7369676e617475726556616c7565                  #       "signatureValue'
   78 58                                            #     text(88)
      6f3972364c78676f474e38466f616565              #       "o9r6LxgoGN8Foae
      554136456444637631324776447a4645              #       "UA6EdDcv12GvDzE
      6d43676a577a76707572325953517941              #       "mCgjWzvpur2YSQy
      385732723053535755 4b2b6e4835744d             #       "8W2r0SSWUK+nH5t
      717a61464c756e3677775a31456f7433              #       "qzaFLun6wwZ1Eot
      37616d4744673d3d                              #       "7amGDg=="
67                                                  #   text(7)
```

```
   63726561746564                                  #     "created"
74                                                 #   text(20)
   323031382d31322d30315430333a30303a30305a        #     "2018-12-01T03:00:
67                                                 #   text(7)
   75706461746564                                  #     "updated"
74                                                 #   text(20)
   323032302d30352d30315430333a30303a30305a        #     "2020-05-01T03:00:
68                                                 #   text(8)
   40636f6e74657874                                #     "@context"
78 1c                                              #   text(28)
   68747470733a2f2f7777772e77332e6f                #     "https://www.w3.o'
   72672f6e732f6469642f7631                        #     "rg/ns/did/v1"
69                                                 #   text(9)
   7075626c69634b6579                              #     "verificationMetho
81                                                 #   array(1)
   a5                                              #     map(5)
      62                                           #       text(2)
         6964                                      #         "id"
      78 85                                        #       text(133)
         626166797265696375627478357716f           #         "bafyreicubtx5
         336e6f73633463617a726b6374666877          #         "3nosc4cazrkct
         6436726577657a6770776f6534737769          #         "d6rewezgpwoe4
         726c733465626468733269b6578616d           #         "rls4ebdhs2i;e
         706c653a6b65793d6964626166797265          #         "ple:key=idbaf
         696375627478357716f336e6f736334           #         "icubtx5wqo3nc
         63617a726b6374666877643672657765          #         "cazrkctfhwd6r
         7a6770776f6534737769726c73346562          #         "zgpwoe4swirls
         6468733269                                #         "dhs2i"
      64                                           #       text(4)
         74797065                                  #         "type"
      6e                                           #       text(14)
         45644473615075626c69634b6579              #         "EdDsaPublicKe
      65                                           #       text(5)
         6375727665                                #         "curve"
      67                                           #       text(7)
         65643235353139                            #         "ed25519"
      67                                           #       text(7)
         65787069726573                            #         "expires"
      74                                           #       text(20)
         323031392d31322d30315430333a30303a30305a  #         "2019-12-01T03
      6f                                           #       text(15)
         7075626c69634b6579426173653634            #         "publicKeyBase
      78 2c                                        #       text(44)
         716d7a3774704c4e4b4b4b646c376344          #         "qmz7tpLNKKKd
```

```
         375062656a4469425670374f4e706d5a          #         "7PbejDiBVp7ON
         62666d633763454b396d673d                  #         "bfmc7cEK9mg="
6e                                                #   text(14)
   61757468656e7469636174696f6e                   #     "authentication"
81                                                #   array(1)
   78 83                                          #     text(131)
      6469643a6578616d706c653a31324433            #       "did:example:12D
      4b6f6f574d4864727a6377706a626472            #       "KooWMHdrzcwpjbd
      5a7335474771455241766367715833 62            #       "Zs5GGqERAvcgqX3
      3564707550745061396f743639796577            #       "5dpuPtPa9ot69ye
      3b6b65792d69643d6261667972656963            #       ";key-id=bafyrei
      756274783577716f336e6f7363346361            #       "ubtx5wqo3nosc4c
      7a726b6374666877643672657765a67              #       "zrkctfhwd6rewez
      70776f6534737769726c733465626468            #       "pwoe4swirls4ebd
      733269                                      #       "s2i"
```

6.3.2 Consumption §

When consuming DID Documents that are represented as CBOR, in addition to the suggestions in section 3.9 of the CBOR [RFC7049] specification for deterministic mappings the following constraints of the DID Document model *MUST* be followed:

- The keys in every map must be sorted lowest value to highest. Sorting is performed on the bytes of the representation of the keys.
- Indefinite-length items must be made into definite-length items.

6.3.3 CBOR Extensibility §

In CBOR, one point of extensibility is with the use of CBOR tags. [RFC7049] defines a basic set of data types, as well as a tagging mechanism that enables extending the set of data types supported via the CBOR Tag Registry. This allows for tags to enhance the semantic description of the data that follows.

6.3.3.1 DagCBOR §

DagCBOR is a further restricted subset of CBOR for representing the DID Document as a Directed Acyclic Graph model using canonical CBOR encoding as noted above with additional constraints.

DagCBOR requires that there exist a single way of encoding any given object, and that encoded forms contain no superfluous data that may be ignored or lost in a round-trip decode/encode. When producing and consuming DID Documents representing in DagCBOR the following rules *MUST* be followed

- Use no CBOR tags other than the CID tag (42)

> EXAMPLE 22: DID Document as DagCBOR same as the previous example, but serialized to JSON for easy readability
>
> ```
> { "@context": "https://www.w3.org/ns/did/v1",
>   "authentication": [
>     "did:example:12D3KooWMHdrzcwpjbdrZs5GGqERAvcgqX3b5dpuPtPa9ot69yew;key-
>   ],
>   "created": "2018-12-01T03:00:00Z",
>   "id": "did:example:12D3KooWMHdrzcwpjbdrZs5GGqERAvcgqX3b5dpuPtPa9ot69yew'
>   "proof": {
>     "created": "2020-05-01T03:00:02Z",
>     "creator": "did:example:12D3KooWMHdrzcwpjbdrZs5GGqERAvcgqX3b5dpuPtPa9
>     "signatureValue": "o9r6LxgoGN8FoaeeUA6EdDcv12GvDzFEmCgjWzvpur2YSQyA8W
>     "type": "ed25519Signature2018"
>   },
>   "verificationMethod": [
>     {
>       "curve": "ed25519",
>       "expires": "2019-12-01T03:00:00Z",
>       "id": "bafyreicubtx5wqo3nosc4cazrkctfhwd6rewezgpwoe4swirls4ebdhs2i;
>       "publicKeyBase64": "qmz7tpLNKKKdl7cD7PbejDiBVp7ONpmZbfmc7cEK9mg=",
>       "type": "EdDsaPublicKey"
>     }
>   ],
>   "updated": "2020-05-01T03:00:00Z"
> }
> ```

6.3.3.2 COSE signatures §

A DID Document proof may be constructed using CBOR semantic tagging, such as tag 98 for CBOR Object Signing and Encryption (COSE) [RFC8152]

EXAMPLE 23: An example extensibility of COSE signature of CBOR using tag 98 and 42 expressed in diagnostic annotated form

```
D8 62                                           # tag(98)
67                                              #   text(7)
   7061796c6f6164                               #     "payload"
d8 2a                                           #   tag(42)
   58 25                                        #     bytes(37)
      00017112206c8fdc5c3d2302dda95034          #       "\x00\x01q\x12 l\x8f\
      f9de57a8591918ecb7d7789387c547f7          #       "\xf9\xdeW\xa8Y\x19\x
      a89d05e72f                                #       "\xa8\x9d\x05\xe7/"
69                                              #   text(9)
   70726f746563746564                           #     "protected"
a0                                              #   map(0)
6a                                              #   text(10)
   7369676e617475726573                         #     "signatures"
81                                              #   array(1)
   a3                                           #     map(3)
      69                                        #       text(9)
         70726f746563746564                     #         "protected"
      66                                        #       text(6)
         613130313236                           #         "a10126"
      69                                        #       text(9)
         7369676e6174757265                     #         "signature"
      78 80                                     #       text(128)
         6532616561666643430643963964313964    #         "e2aeafd40d69d19d"
         6665366535323037376335643766663 4     #         "fe6e52077c5d7ff4"
         6534303832383263626566623564303 6     #         "e408282cbefb5d06"
         6362663431346166326531396439383 2     #         "cbf414af2e19d982"
         6163343561633938623835343463393 0     #         "ac45ac98b8544c90"
         3862343530376465316539306237313 7     #         "8b4507de1e90b717"
         6333643334383136666539323661326 2     #         "c3d34816fe926a2b"
         3938663533616664326661306633306 1     #         "98f53afd2fa0f30a"
      6b                                        #       text(11)
         756e70726f746563746564                 #         "unprotected"
      a1                                        #       map(1)
         63                                     #         text(3)
            6b6964                              #           "kid"
         78 85                                  #         text(133)
            6469643a697069643a313244334b6f6f    #           "did:ipid:12D3Koo
            574d4864727a6377706a6264725a7335    #           "WMHdrzcwpjbdrZs5
            4747714552417663677158336235647 0   #           "GGqERAvcgqX3b5dp
            7550745061396f7436397965773b697 0   #           "uPtPa9ot69yew;ip
            69643a6b65792d69643d626166797265    #           "id:key-id=bafyre
```

```
            696375627478357716f336e6f736334 #           "icubtx5wqo3nosc4
            63617a726b6374666877643672657765 #          "cazrkctfhwd6rewe
            7a6770776f6534737769726c73346562 #          "zgpwoe4swirls4eb
            6468733269                      #           "dhs2i"
6b                                          #   text(11)
   756e70726f746563746564                   #     "unprotected"
a0                                          #   tag(0)
```

7. Methods §

DID methods provide the means to implement this specification on different verifiable data registries. New DID methods are defined in their own specifications, so that interoperability between different implementations of the same DID method is ensured. This section specifies the requirements on any DID method, which are met by the DID method's associated specification.

For adding properties to a DID document which are specific to a particular DID method, see § 4.3 Extensibility.

7.1 Method Schemes §

A DID method specification *MUST* define exactly one method-specific DID scheme, identified by exactly one method name (see the `method-name` rule in Section § 3.1 DID Syntax).

The authors of a new DID method specification *SHOULD* use a method name that is unique among all DID method names known to them at the time of publication.

The method name *SHOULD* be five characters or less.

> NOTE: Unique DID method names
>
> Because there is no central authority for allocating or approving DID method names, there is no way to know for certain if a specific DID method name is unique. To help with this challenge, a non-authoritative list of known DID method names and their associated specifications is maintained in the DID Methods Registry, which is part of [DID-SPEC-REGISTRIES].
>
> Authors of new DID method specifications are encouraged to add their method names to the DID Method Registry so that other implementors and members of the community have a place to see an overview of existing DID methods.

The DID method specification *MUST* specify how to generate the `method-specific-id` component of a DID.

The `method-specific-id` value *MUST* be able to be generated without the use of a centralized registry service.

The `method-specific-id` value *SHOULD* be globally unique by itself. Any DID generated by the method *MUST* be globally unique.

If needed, a method-specific DID scheme *MAY* define multiple `method-specific-id` formats. It is *RECOMMENDED* that a method-specific DID scheme define as few `method-specific-id` formats as possible.

The `method-specific-id` format *MAY* include colons. The use of colons *MUST* comply syntactically with the `method-specific-id` ABNF rule.

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

7.2 Method Operations §

This section sets out the requirements for DID method specifications with regards to operations that can be performed on a DID document.

Determining the authority of a party to carry out the operations is method-specific. For example, a DID method *MAY*:

- make use of the `controller` property.
- use the verification methods listed under `authentication` to decide whether an update/deactivate operation is allowed.
- use other constructs in the DID Document to decide this, for example, a verification method specified under `capabilityInvocation` could be used to verify the invocation of a capability to update the DID Document.
- not use the DID Document at all to decide this, but have rules that are "built into" the method.

Each DID method *MUST* define how authorization is implemented, including any necessary cryptographic operations.

7.2.1 Create §

The DID method specification *MUST* specify how a DID controller creates a DID and its associated DID document on the verifiable data registry, including all cryptographic operations necessary to establish proof of control.

7.2.2 Read/Verify §

The DID method specification *MUST* specify how a DID resolver uses a DID to request a DID document from the verifiable data registry, including how the DID resolver can verify the authenticity of the response.

7.2.3 Update §

The DID method specification *MUST* specify how a DID controller can update a DID document on the verifiable data registry, including all cryptographic operations necessary to establish proof of control, *or* state that updates are not possible.

An update to a DID is any change, after creation, in the data used to produce a DID document. DID Method implementers are responsible for defining what constitutes an update, and what properties of the DID document are supported by a given DID method. For example, an update operation which replaces key material without changing it could be a valid update that does not result in changes to the DID document.

7.2.4 Deactivate §

The DID method specification *MUST* specify how a DID controller can deactivate a DID on the verifiable data registry, including all cryptographic operations necessary to establish proof of deactivation, *or* state that deactivation is not possible.

7.3 Security Requirements §

DID method specifications *MUST* include their own Security Considerations sections. This section *MUST* consider all the requirements mentioned in section 5 of [RFC3552] (page 27) for the DID operations defined in the specification.

At least the following forms of attack *MUST* be considered: eavesdropping, replay, message insertion, deletion, modification, and man-in-the-middle. Potential denial of service attacks *MUST* be identified as well.

This section *MUST* discuss, per Section 5 of [RFC3552], residual risks (such as the risks from compromise in a related protocol, incorrect implementation, or cipher) after threat mitigation was deployed.

This section *MUST* provide integrity protection and update authentication for all operations required by Section § 7.2 Method Operations.

If the technology involves authentication, particularly user-host authentication, the security of the authentication method *MUST* be clearly specified.

DID methods *MUST* discuss the policy mechanism by which DIDs are proven to be uniquely assigned. A DID fits the functional definition of a URN, as defined in [RFC8141]. That is, a DID is a persistent identifier that is assigned once to a resource and never reassigned to a different resource. This is particularly important in a security context because a DID might be used to identify a specific party subject to a specific set of authorization rights.

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.

If the protocol incorporates cryptographic protection mechanisms, the DID method specification *MUST* clearly indicate which portions of the data are protected and what the protections are, and *SHOULD* give an indication to what sorts of attacks the cryptographic protection is susceptible. For example, integrity only, confidentiality, endpoint authentication, and so on.

Data which is to be held secret (keying material, random seeds, and so on) *SHOULD* be clearly labeled.

Where DID methods make use of 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.

DID methods that introduce new authentication service endpoint types (see Section § 5.5 Service Endpoints) *SHOULD* consider the security requirements of the supported authentication protocol.

7.4 Privacy Requirements §

DID method specifications *MUST* include their own Privacy Considerations sections, if only to point to § 10. Privacy Considerations .

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, exclusion.

8. Resolution §

This section defines the inputs and outputs of DID resolution and DID URL dereferencing. These functions are defined in an abstract way. Their exact implementation is out of scope for this specification, but some considerations for implementors 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.

8.1 DID Resolution §

The DID resolution functions resolve a DID into a DID document by using the "Read" operation of the applicable DID method. (See § 7.2.2 Read/Verify .) The details of how this process is accomplished are outside the scope of this specification, but all conformant implementations *MUST* implement two functions which have the following abstract forms:

```
resolve ( did, did-resolution-input-metadata )
     -> ( did-resolution-metadata, did-document, did-document-metadata )

resolveStream ( did, did-resolution-input-metadata )
     -> ( did-resolution-metadata, did-document-stream, did-document-
metadata )
```

The input variables of these functions *MUST* be as follows:

did
: A conformant DID as a single string. This is the DID to resolve. This input is *REQUIRED*.

did-resolution-input-metadata
: A metadata structure consisting of input options to the `resolve` and `resolveStream` functions in addition to the `did` itself. Properties defined by this specification are in § 8.1.1 DID Resolution

Input Metadata Properties . This input is *REQUIRED*, but the structure *MAY* be empty.

The output variables of these functions *MUST* be as follows:

did-resolution-metadata
: A metadata structure consisting of values relating to the results of the DID resolution process. This structure is *REQUIRED* and *MUST NOT* be empty. This metadata typically changes between invocations of the `resolve` and `resolveStream` functions as it represents data about the resolution process itself. Properties defined by this specification are in § 8.1.2 DID Resolution Metadata Properties . If the resolution is successful, and if the `resolveStream` function was called, this structure *MUST* contain a `content-type` property containing the mime-type of the `did-document-stream` in this result. If the resolution is not successful, this structure *MUST* contain an `error` property describing the error.

did-document
: If the resolution is successful, and if the `resolve` function was called, this *MUST* be a DID document conforming to the abstract data model. If the resolution is unsuccessful, this value *MUST* be empty.

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

did-document-metadata
: If the resolution is successful, this *MUST* be a metadata structure. This structure contains metadata about the DID document contained in the `did-document` or `did-document-stream`. This metadata typically does not change between invocations of the `resolve` function unless the DID document changes, as it represents data about the DID document. If the resolution is unsuccessful, this output *MUST* be an empty metadata structure. Properties defined by this specification are in § 8.1.3 DID Document Metadata Properties .

DID resolver implementations *MUST NOT* alter the signature of these functions in any way. DID resolver implementations *MAY* map the `resolve` and `resolveStream` functions to a method-specific internal function to perform the actual DID resolution process. DID resolver implementations *MAY* implement and expose additional functions with different signatures in addition to the `resolve` function specified here.

8.1.1 DID Resolution Input Metadata Properties §

The possible properties within this structure and their possible values are defined by [DID-SPEC-REGISTRIES]. This specification defines the following common properties.

accept
: The MIME type of the caller's preferred representation of the DID document. The DID resolver implementation *SHOULD* use this value to determine the representation contained in the returned `did-document-stream` if such a representation is supported and available. This property is *OPTIONAL*. It is only used if the `resolveStream` function is called and *MUST* be ignored if the `resolve` function is called.

8.1.2 DID Resolution Metadata Properties §

The possible properties within this structure and their possible values are defined by [DID-SPEC-REGISTRIES]. This specification defines the following common properties.

content-type
: The MIME type of the returned `did-document-stream`. This property is *REQUIRED* if resolution is successful and if the `resolveStream` function was called. It *MUST NOT* be present if the `resolve` function was called. The value of this property *MUST* be the MIME type of one of the conformant representations. The caller of the `resolveStream` function *MUST* use this value when determining how to parse and process the `did-document-stream` returned by this function into a DID document abstract 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 is a single keyword string. The possible property values of this field are defined by [DID-SPEC-REGISTRIES]. This specification defines the following error values:

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

  unauthorized
  : The caller is not authorized to resolve this DID with this DID resolver.

  not-found
  : The DID resolver was unable to return the DID document resulting from this resolution request.

8.1.3 DID Document Metadata Properties §

The possible properties within this structure and their possible values are defined by [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. This property *MAY* not be supported by a given DID method. The value of the property *MUST* be 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, as indicated by the trailing "Z".

updated
: DID document metadata *SHOULD* include an `updated` property to indicate the timestamp of the last Update operation. This property *MAY* not be supported by a given DID method. The value of the property *MUST* follow the same formatting rules as the `created` property.

8.2 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. The details of how this process is accomplished are outside the scope of this specification, but all conformant implementations *MUST* implement a function which has the following abstract form:

```
dereference ( did-url, did-url-dereferencing-input-metadata )
         -> ( did-url-dereferencing-metadata, content-stream, content-
metadata )
```

The input variables of this function *MUST* be as follows:

did-url
: A conformant DID URL as a single string. This is the DID URL to dereference. This input is *REQUIRED*.

did-url-dereferencing-input-metadata
: A metadata structure consisting of input options to the `dereference` function in addition to the `did-url` itself. Properties defined by this specification are in § 8.2.1 DID URL Dereferencing Input Metadata Properties . This input is *REQUIRED*, but the structure *MAY* be empty.

The output variables of this function *MUST* be as follows:

did-url-dereferencing-metadata

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

content-stream

If the `dereferencing` function was called and successful, this *MUST* contain a resource corresponding to the DID URL. The `content-stream` *MAY* be a DID document in one of the conformant representations obtained through the resolution process. If the dereferencing is unsuccessful, this value *MUST* be empty.

content-metadata

If the dereferencing is successful, this *MUST* be a metadata structure, but the structure *MAY* be empty. This structure contains metadata about the `content-stream`. If the `content-stream` is a DID document, this *MUST* be a `did-document-metadata` structure as described in DID Resolution. If the dereferencing is unsuccessful, this output *MUST* be an empty metadata structure.

DID URL Dereferencing implementations *MUST NOT* alter the signature of these functions in any way. DID URL Dereferencing implementations *MAY* map the `dereference` function to a method-specific internal function to perform the actual DID URL Dereferencing process. DID URL Dereferencing implementations *MAY* implement and expose additional functions with different signatures in addition to the `dereference` function specified here.

8.2.1 DID URL Dereferencing Input Metadata Properties §

The possible properties within this structure and their possible values are defined by [DID-SPEC-REGISTRIES]. This specification defines the following common properties.

accept

The MIME type the caller prefers for `content-stream`. The DID URL Dereferencing implementation *SHOULD* use this value to determine the representation contained in the returned value if such a representation is supported and available. This property is *OPTIONAL*.

8.2.2 DID URL Dereferencing Metadata Properties §

The possible properties within this structure and their possible values are defined by [DID-SPEC-REGISTRIES]. This specification defines the following common properties.

content-type

The MIME type of the returned `content-stream`. This property is *OPTIONAL* if dereferencing is successful.

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 is a single keyword string. The possible property values of this field are defined by [DID-SPEC-REGISTRIES]. This specification defines the following error values:

invalid-did-url

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

unauthorized

The caller is not authorized to dereference the given DID URL with the given DID URL dereferencer.

not-found

The DID URL dereferencer was unable to return the `content-stream` resulting from this dereferencing request.

8.2.3 DID URL Dereferencing Metadata Properties §

The possible properties within this structure and their possible values are defined by [DID-SPEC-REGISTRIES]. This specification defines the following common properties.

8.3 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, 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 *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 where possible.

All implementations of functions that use metadata structures as either input or output *MUST* be 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": "not-found"
}
```

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

EXAMPLE 27: DID resolution metadata example

```
«[
  "error" → "not-found"
]»
```

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"
]»
```

9. Security Considerations §

This section is non-normative.

NOTE: Note to implementers

During the Working Draft stage, this section focuses on security topics that should be important in early implementations. The editors are seeking feedback on threats and threat mitigations that should be reflected in this section or elsewhere in the spec. DIDs are designed to operate under the general Internet threat model used by many IETF standards. We assume uncompromised endpoints, but anticipate that messages could be read or corrupted on the network.

9.1 Choosing DID Resolvers §

The DID Method Registry (see [DID-SPEC-REGISTRIES] is an informative list of DID method names and their corresponding DID method specifications. Implementors 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, but can use the DID Method Registry to make an informed decision when choosing which DID resolver implementations to use.

9.2 Binding of Identity §

The following sections describe binding identities to DIDs and DID documents.

9.2.1 Proving Control of a DID and DID Document §

Signatures and verifiable timestamps allow DID documents to be cryptographically verifiable.

By itself, a verified signature on a self-signed DID document does not prove control of a DID. It only proves that the:

- DID document was not tampered with since it was timestamped.
- DID controller(s) controlled the private key used for the signature at the time the timestamp was created.

Proving control of a DID, that is, the binding between the DID and the DID document that describes it, requires a two step process:

1. Resolving the DID to a DID document according to its DID method specification.
2. Verifying that the `id` property of the resulting DID document matches the DID that was resolved.

It should be noted that this process proves control of a DID and DID document regardless of whether the DID document is signed.

Signatures on DID documents are optional. DID method specifications *SHOULD* explain and specify their implementation if applicable.

9.2.2 Proving Control of a Public Key §

There are two methods for proving control of the private key corresponding to a public key description in the DID document: static and dynamic.

The static method is to sign the DID document with the private key. This proves control of the private key at a time no later than the DID document was registered. If the DID document is not signed, control of a public key described in the DID document can still be proven dynamically as follows:

1. Send a challenge message containing a public key description from the DID document and a nonce to an appropriate service endpoint described in the DID document.
2. Verify the signature of the response message against the public key description.

9.2.3 Authentication and Verifiable Claims §

A DID and DID document do not inherently carry any PII (personally-identifiable information). The process of binding a DID to something in the real world, such as a person or a company, for example with credentials with the same subject as that DID, is out of scope for this specification. For more information, see the [VC-DATA-MODEL] instead.

9.3 Authentication Service Endpoints §

If a DID document publishes a service endpoint intended for authentication or authorization of the DID subject (see Section § 5.5 Service Endpoints), 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 under the assumption that the subject:

- Is monitoring for unauthorized updates (see Section § 9.5 Notification of DID Document Changes).
- Has had adequate opportunity to revert malicious updates according to the access control mechanism for the DID method (see Section § 5.4.1 authentication).

Non-repudiation is further supported if timestamps are included (see Section § 8.1.3 DID Document Metadata Properties) and the target DLT system supports timestamps.

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 are no centralized authorities to enforce key or signature expiration policies. Therefore DID resolvers and requesting parties need to validate that keys were not expired at the time they were used. Because some use cases might have legitimate reasons why already-expired keys can be extended, make sure a key expiration does not prevent any further use of the key, and implementations of a resolver ought to be compatible with such extension behavior.

9.7 Key Revocation and Recovery §

Section § 7.2 Method Operations specifies the DID operations to be supported by a DID method specification, including deactivation of a DID document by replacing it with an updated DID document. It is also up to the DID method to define how revocation of cryptographic keys might occur. Additionally, DID method specifications are also expected to enable support for a quorum of trusted parties to enable key recovery. Some of the facilities to do so are suggested in Section § 5.2 Control . 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 key 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. Further specification of this type of control is a matter for future work.

9.8 The Role of Human-Friendly Identifiers §

DIDs achieve global uniqueness without the need for a central registration authority. This comes, however, at the cost of human memorability. The algorithms capable of generating globally unique identifiers automatically produce random strings of characters that have no human meaning. This demonstrates the axiom about identifiers described in Zooko's Triangle: "human-meaningful, decentralized, secure — pick any two".

There are of course many 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, Twitter handle, 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 (and there are many) should be defined in separate specifications 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.

> NOTE
>
> A draft specification for discovering a DID from domain names and email addresses using DNS lookups is available at [DNS-DID].

9.9 Immutability §

Many cybersecurity abuses hinge on exploiting gaps between reality and the assumptions of rational, good-faith actors. Like any ecosystem, the DID ecosystem has some potential for this to occur. Because this specification is focused on a data model instead of a protocol, it offers no opinion about many aspects of how that model is put to use. However, individual DID methods might want 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 the flexibility that the data model offers with respect to updating. A single edit to a DID document can change anything and everything except the root `id` property of the document. And any individual JSON object in the data model can change all of its properties except its `id`. But is it actually desirable for a service endpoint 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 web site often aim for an outcome where the site keeps its identifier (the host name), but gets subtle, dangerous changes underneath. If certain properties of the site were required by the specification to be immutable (for example, the ASN associated with its IP address), such attacks might be much harder and more expensive to carry out, and anomaly detection would be easier.

The notion that immutability provides some cybersecurity benefits is particularly relevant because of caching. 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.

9.10 Encrypted Data in DID Documents §

DID documents are typically publicly available. 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.

Encrypting all or parts of DID documents is *not* an appropriate means to protect data in the long term. Similarly, placing encrypted data in DID documents is not an appropriate means to include personally identifiable information.

Given the caveats above, if encrypted data is included in a DID document, implementers are advised to not encrypt with the public keys of entities that do not wish to be correlated with the DID.

10. Privacy Considerations §

This section is non-normative.

It is critically important to apply the principles of Privacy by Design to all aspects of the decentralized identifier architecture, because DIDs and DID documents are, by design, administered directly by the DID controller(s). There is no registrar, hosting company, or other intermediate service provider to recommend or apply additional privacy safeguards. The authors of this specification have applied all seven Privacy by Design principles throughout its development. For example, privacy in this specification is preventative not remedial, and privacy is an embedded default. Furthermore, the decentralized identifier architecture by itself embodies principle #7, "Respect for user privacy — keep it user-centric."

This section lists additional privacy considerations that implementers, delegates, and DID subjects should keep in mind.

10.1 Keep Personally-Identifiable Information (PII) Private §

If a DID method specification is written for a public verifiable data registry where all DIDs and DID documents are publicly available, it is *critical* that DID documents contain no personal data. All personal data should be kept behind service endpoints under the control of the DID subject. Additional

due diligence should be taken around the use of URLs in service endpoints as well to prevent leakage of unintentional personal data or correlation within a URL of a service endpoint. For example, a URL that contains a username is likely dangerous to include in a DID Document because the username is likely to be human-meaningful in a way that can unintentionally reveal information that the DID subject did not consent to sharing. With this privacy architecture, personal data can be exchanged on a private, peer-to-peer basis using communications channels identified and secured by public key descriptions 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 and Pseudonymous DIDs §

Like any type of globally unique identifier, DIDs might be used for correlation. DID controllers can mitigate this privacy risk by using pairwise unique DIDs, that is, sharing a different private DID for every relationship. In effect, each DID acts as a pseudonym. A pseudonymous DID need only be shared with more than one party when the DID subject explicitly authorizes correlation between those parties. If pseudonymous DIDs are the default, then the only need for a public DID (a DID published openly or shared with a large number of parties) is when the DID subject explicitly desires public identification.

10.3 DID Document Correlation Risks §

The anti-correlation protections of pseudonymous DIDs are easily defeated if the data in the corresponding DID documents can be correlated. For example, using same public key descriptions 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 pseudonymous DID also needs to use pairwise unique public keys. It might seem natural to also use pairwise unique service endpoints in the DID document for a pseudonymous 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 thousands or millions of DIDs controlled by many different subjects.

10.4 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.

11. Examples §

This section is non-normative.

11.1 DID Documents §

This section is non-normative.

See did-spec-registries for optional extensions and other verifcation 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",
  "id": "did:example:123",
  "authentication": [
    {
      "id": "did:example:123#z6MkecaLyHuYWkayBDLw5ihndj3T1m6zKTGqau3A51
      "type": "Ed25519VerificationKey2018",
      "controller": "did:example:123",
      "publicKeyBase58": "AKJP3f7BD6W4iWEQ9jwndVTCBq8ua2Utt8EEjJ6Vxsf"
    }
  ],
  "capabilityInvocation": [
    {
      "id": "did:example:123#z6MkhdmzFu659ZJ4XKj31vtEDmjvsi5yDZG5L7Caz6
      "type": "Ed25519VerificationKey2018",
      "controller": "did:example:123",
      "publicKeyBase58": "4BWwfeqdp1obQptLLMvPNgBw48p7og1ie6Hf9p5nTpNN"
    }
  ],
  "capabilityDelegation": [
    {
      "id": "did:example:123#z6Mkw94ByR26zMSkNdCUi6FNRsWnc2DFEeDXyBGJ5K
      "type": "Ed25519VerificationKey2018",
      "controller": "did:example:123",
      "publicKeyBase58": "Hgo9PAmfeoxHG8Mn2XHXamxnnSwPpkyBHAMNF3VyXJCL"
    }
  ],
  "assertionMethod": [
    {
      "id": "did:example:123#z6MkiukuAuQAE8ozxvmahnQGzApvtW7KT5XXKfojjv
      "type": "Ed25519VerificationKey2018",
      "controller": "did:example:123",
      "publicKeyBase58": "5TVraf9itbKXrRvt2DSS95Gw4vqU3CHAdetoufdcKazA"
    }
  ]
}
```

EXAMPLE 31: DID Document with many different verification methods


```
{
  "@context": "https://www.w3.org/ns/did/v1",
  "id": "did:example:123",
  "verificationMethod": [
    {
      "id": "did:example:123#ZC2jXTO6t4R501bfCXv3RxarZyUbdP2w_psLwMuY6ec'
      "type": "Ed25519VerificationKey2018",
      "controller": "did:example:123",
      "publicKeyBase58": "H3C2AVvLMv6gmMNam3uVAjZpfkcJCwDwnZn6z3wXmqPV"
    },
    {
      "id": "did:example:123#zQ3shP2mWsZYWgvgM11nenXRTx9L1yiJKmkf9dfX7NaM
      "type": "EcdsaSecp256k1VerificationKey2019",
      "controller": "did:example:123",
      "publicKeyBase58": "d5cW2R53NHTTkv7EQSYR8YxaKx7MVCcchjmK5EgCNXxo",
    },
    {
      "id": "did:example:123#_Qq0UL2Fq651Q0Fjd6TvnYE-faHiOpRlPVQcY_-tA4A'
      "type": "JsonWebKey2020",
      "controller": "did:example:123",
      "publicKeyJwk": {
        "kty": "OKP",
        "crv": "Ed25519",
        "x": "VCpo2LMLhn6iWku8MKvSLg2ZAoC-nlOyPVQaO3FxVeQ"
      }
    },
    {
      "id": "did:example:123#z6LSnjagzhe8Df6gZmroW3wjDd7XQLwAuYfwa4ZeTBC(
      "type": "JsonWebKey2020",
      "controller": "did:example:123",
      "publicKeyJwk": {
        "kty": "OKP",
        "crv": "X25519",
        "x": "pE_mG098rdQjY3MKK2D5SUQ6ZOEW3a6Z6T7Z4SgnzCE"
      },
    }
    {
      "id": "did:example:123#4SZ-StXrp5Yd4_4rxHVTCYTHyt4zyPfN1fIuYsm6k3A'
      "type": "JsonWebKey2020",
      "controller": "did:example:123",
      "publicKeyJwk": {
        "kty": "EC",
```

```
      "crv": "secp256k1",
      "x": "Z4Y3NNOxv0J6tCgqOBFnHnaZhJF6LdulT7z8A-2D5_8",
      "y": "i5a2NtJoUKXkLm6q8nOEu9WOkso1Ag6FTUT6k_LMnGk"
    }
  },
  {
    "id": "did:example:123#n4cQ-I_WkHMcwXBJa7IHkYu8CMfdNcZKnKsOrnHLpFs'
    "type": "JsonWebKey2020",
    "controller": "did:example:123",
    "publicKeyJwk": {
      "kty": "RSA",
      "e": "AQAB",
      "n": "omwsC1AqEk6whvxyOltCFWheSQvv1MExu5RLCMT4jVk9khJKv8JeMXWe3bV
    }
  },
  {
    "id": "did:example:123#_TKzHv2jFIyvdTGF1Dsgwngfdg3SH6TpDv0Ta1aOEkw'
    "type": "JsonWebKey2020",
    "controller": "did:example:123",
    "publicKeyJwk": {
      "kty": "EC"
      "crv": "P-256",
      "x": "38M1FDts7Oea7urmseiugGW7tWc3mLpJh6rKe7xINZ8",
      "y": "nDQW6XZ7b_u2Sy9slofYLlG03sOEoug3I0aAPQ0exs4"
    }
  },
  {
    "id": "did:example:123#8wgRfY3sWmzoeAL-78-oALNvNj67ZlQxd1ss_NX1hZY'
    "type": "JsonWebKey2020",
    "controller": "did:example:123",
    "publicKeyJwk": {
      "kty": "EC",
      "crv": "P-384",
      "x": "GnLl6mDti7a2VUIZP5w6pcRX8q5nvEIgB3Q_5RI2p9F_QVsaAlDN7IG68Jr
      "y": "jq4QoAHKiIzezDp88s_cxSPXtuXYFliuCGndgU4Qp8l91xzD1spCmFIzQgV
    }
  },
  {
    "id": "did:example:123#NjQ6Y_ZMj6IUK_XkgCDwtKHlNTUTVjEYOWZtxhp1n-E'
    "type": "JsonWebKey2020",
    "controller": "did:example:123",
    "publicKeyJwk": {
      "kty": "EC",
      "crv": "P-521",
```

```
        "x": "AVlZG23LyXYwlbjbGPMxZbHmJpDSu-IvpuKigEN2pzgWtSo--Rwd-n78nrV
        "y": "ANIbFeRdPHf1WYMCUjcPz-ZhecZFybOqLIJjVOlLETH7uPlyG0gEoMWnIZ>
      }
    }
  ]
}
```

11.2 Proving §

This section is non-normative.

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

EXAMPLE 32: Verifiable Credential linked to a verification method of type Ed25519VerificationKey2018


```
{
  "@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..BhV
  }
}
```

EXAMPLE 33: Verifiable Credential linked to a verification method of type JsonWebKey2020


```
{
  "@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-faHiOp
    "proofPurpose": "assertionMethod",
    "jws": "eyJiNjQiOmZhbHNlLCJjcml0IjpbImI2NCJdLCJhbGciOiJFZERTQSJ9..Y0K
  }
}
```

EXAMPLE 34: Verifiable Credential as Decoded JWT


```
{
  "protected": {
    "kid": "did:example:123#_Qq0UL2Fq651Q0Fjd6TvnYE-faHiOpRlPVQcY_-tA4A",
    "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_M7HPxinCbVhz9iIJCg70UMeQ
}
```


11.3 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 35: JWE linked to a verification method via kid


```
{
  "ciphertext": "3SHQQJajNH6q0fyAHmw...",
  "iv": "QldSPLVnFf2-VXcNLza6mbylYwphW57Q",
  "protected": "eyJlbmMiOiJYQzIwUCJ9",
  "recipients": [
    {
      "encrypted_key": "BMJ19zK12YHftJ4sr6Pz1rX1HtYni_L9DZvO1cEZfRWDN2vXe
      "header": {
        "alg": "ECDH-ES+A256KW",
        "apu": "Tx9qG69ZfodhRos-8qfhTPc6ZFnNUcgNDVdHqX1UR3s",
        "apv": "ZGlkOmVsZW06cm9wc3RlbjpFa...",
        "epk": {
          "crv": "X25519",
          "kty": "OKP",
          "x": "Tx9qG69ZfodhRos-8qfhTPc6ZFnNUcgNDVdHqX1UR3s"
        },
        "kid": "did:example:123#zC1Rnuvw9rVa6E5TKF4uQVRuQuaCpVgB81Um2u17F
      }
    }
  ],
  "tag": "xbfwwDkzOAJfSVem0jr1bA"
}
```


A. Current Issues §

The list of issues below are under active discussion and are likely to result in changes to this specification.

~~ISSUE 5~~: Where will the DID contexts(s) live? **extensibility** **pending close**

Where will the DID contexts(s) live?

~~ISSUE 8~~: Leverage RFC7518 to specify cryptographic algorithms **jose** **keys** **needs-special-call** **pending close**

Leverage RFC7518 to specify cryptographic algorithms

~~ISSUE 9~~: Replace RsaSignature2017 by a standard JWA signature **pending close**

Replace RsaSignature2017 by a standard JWA signature

~~ISSUE 10~~: Explain RsaSignature2018 **pending close**

Explain RsaSignature2018

~~ISSUE 14~~: Standardize the key revocation list **pending close**

Standardize the key revocation list

~~ISSUE 23~~: publicKeyJwk, publicKeyHex, publicKeyBase64, publicKeyBase58 missing from context. discuss **extensibility** **high priority**

publicKeyJwk, publicKeyHex, publicKeyBase64, publicKeyBase58 missing from context.

ISSUE 33: Cheap DIDs and the option to migrate DIDs between ledgers using standard DID Deprecation Registries discuss

Cheap DIDs and the option to migrate DIDs between ledgers using standard DID Deprecation Registries

~~ISSUE 36~~: Details on the use of method-specific DID parameters **pending close**

Details on the use of method-specific DID parameters

~~ISSUE 55~~: Add support for ethereumAddress public key type in @context discuss **high priority**

Add support for ethereumAddress public key type in @context

~~ISSUE 57~~: Clarification of other verification methods in authentication section missing discuss **pending close**

Clarification of other verification methods in authentication section missing

ISSUE 58: Registry handling discuss **extensibility**

Registry handling

~~ISSUE 65~~: Does DID Document metadata belong in the Document? **metadata** **pending close**

Does DID Document metadata belong in the Document?

ISSUE 72: Privacy Considerations - Specifically call out GDPR discuss editorial **metadata**

Privacy Considerations - Specifically call out GDPR

~~ISSUE 75~~: tracking revocation of public keys **pending close**

tracking revocation of public keys

~~ISSUE 85~~: Syntactially differentiate data about the DID versus application data discuss **high priority** **metadata** **pending close**

Syntactially differentiate data about the DID versus application data

~~ISSUE 92~~: Add CBOR a valid type of DID document syntax similar to JSON and on par with JSON-LD **PR exists** discuss

Add CBOR a valid type of DID document syntax similar to JSON and on par with JSON-LD

ISSUE 94: Create DID explainer discuss **horizontal review**

Create DID explainer

~~ISSUE 95~~: Document Structure discuss

Document Structure

ISSUE 104: Horizontal Review: Internationalization self test **horizontal review** **i18n-tracker**

Horizontal Review: Internationalization self test

ISSUE 105: Horizontal Review: Accessibility self test **a11y-tracker** **horizontal review**

Horizontal Review: Accessibility self test

ISSUE 118: Specification needs to be compliant with WCAG 2.0 editorial **just before CR**

Specification needs to be compliant with WCAG 2.0

ISSUE 119: Horizontal Review: offer review opportunity to TAG **horizontal review**

Horizontal Review: offer review opportunity to TAG

ISSUE 122: When is a DID subject not a DID controller (if ever)? **pending close**

When is a DID subject not a DID controller (if ever)?

~~ISSUE 137~~: Should the DID parameters be normative in the spec? **pending close**

Should the DID parameters be normative in the spec?

ISSUE 151: Include discussion of eIDAS levels-of-assurance editorial

Include discussion of eIDAS levels-of-assurance

~~ISSUE 154~~: Decoupling DID Core spec from LD-Proof / LDS specs **extensibility** **pending close**

Decoupling DID Core spec from LD-Proof / LDS specs

ISSUE 163: Uses of terms defined in the specification should be links to their definitions editorial **just before CR**

Uses of terms defined in the specification should be links to their definitions

~~ISSUE 165~~: What are entityship and start-of-authority (SOA) problems? editorial **pending close**

What are entityship and start-of-authority (SOA) problems?

~~ISSUE 169~~: Replace registries administered community groups with registries established by this specification **extensibility**

Replace registries administered community groups with registries established by this specification

ISSUE 170: Public key "id" and "type" members duplicate JWK "kid" and "kty" members editorial **jose**

Public key "id" and "type" members duplicate JWK "kid" and "kty" members

~~ISSUE 171~~: Add public key examples using JWKs editorial **jose** **pending close**

Add public key examples using JWKs

ISSUE 174: Underspecified semantics of "updated" property editorial **metadata**

Underspecified semantics of "updated" property

~~ISSUE 176~~: Unsubstantiated statement about protecting against attacks when compromised editorial

Unsubstantiated statement about protecting against attacks when compromised

~~ISSUE 178~~: Underspecified statement on combining timestamps with signatures editorial **pending close**

Underspecified statement on combining timestamps with signatures

~~ISSUE 185~~: Supported ciphers in a DID document **jose**

Supported ciphers in a DID document

~~ISSUE 190~~: What is being discussed in issue 4 (clarification of TERMX via use-cases, spec pointers, and PR) discuss **pending close**

What is being discussed in issue 4 (clarification of TERMX via use-cases, spec pointers, and PR)

~~ISSUE 195~~: Unclear which verification methods are authorized for did document operations discuss

Unclear which verification methods are authorized for did document operations

~~ISSUE 198~~: Add sections on DID Resolution **pending close**

Add sections on DID Resolution

ISSUE 199: Clarification on what DIDs might identify discuss

Clarification on what DIDs might identify

~~ISSUE 202~~: JSON-LD Contexts in Registry **PR exists** **extensibility**

JSON-LD Contexts in Registry

ISSUE 203: Define DID Document Metadata **metadata** **pending close**

Define DID Document Metadata

~~ISSUE 204~~: Define terminology for properties and values **PR exists** editorial

Define terminology for properties and values

ISSUE 205: How to treat unknown properties discuss extensibility

How to treat unknown properties

~~ISSUE 207~~: Add section on extensibility and conformance editorial extensibility pending close

Add section on extensibility and conformance

ISSUE 208: IETF did+ld+json media type registration extensibility

IETF did+ld+json media type registration

~~ISSUE 236~~: publicKeyHex format unused by spec currently pending close

publicKeyHex format unused by spec currently

ISSUE 240: Should did-core restrict the use of JWK? jose

Should did-core restrict the use of JWK?

~~ISSUE 248~~: Need term for relying party

Need term for relying party

ISSUE 249: How to mitigate the single source of failure wrt/ "Trust into the Universal Resolver"?

How to mitigate the single source of failure wrt/ "Trust into the Universal Resolver"?

ISSUE 253

Added DID resolution and dereferencing contracts.

~~ISSUE 258~~: List of early implementations conforming to spec? pending close

List of early implementations conforming to spec?

~~ISSUE 259~~: DIDs and JOSE: publicKey.id and publicKey.publicKeyJwk.kid discuss pending close

DIDs and JOSE: publicKey.id and publicKey.publicKeyJwk.kid

~~ISSUE 260~~: Clear explanation on how can A DID have more than one controller editorial pending close question

Clear explanation on how can A DID have more than one controller

ISSUE 261: Definition of the term "client" in regard to SSI principles editorial pending close

Definition of the term "client" in regard to SSI principles

~~ISSUE 266~~: Should DID support self-signed certificates? extensibility pending close question

Should DID support self-signed certificates?

ISSUE 267

Put key points up front

~~ISSUE 268~~: What degree should proof purposes be defined for specific application layer usages?

PR exists editorial

What degree should proof purposes be defined for specific application layer usages?

~~ISSUE 269~~: transfer of controllership and it's intersection with the subject of an identifier

PR exists editorial question

transfer of controllership and it's intersection with the subject of an identifier

~~ISSUE 270~~: did parameter equivilance question

did parameter equivilance

~~ISSUE 272~~: Remove all unspecified properties/functionality from the spec extensibility question

Remove all unspecified properties/functionality from the spec

~~ISSUE 273~~: invert mapping between proof purposes and verification methods? pending close

invert mapping between proof purposes and verification methods?

~~ISSUE 274~~: Ambiguity around necessity of populated top-level DID Document 'id' property

question

Ambiguity around necessity of populated top-level DID Document 'id' property

~~ISSUE 280~~: Remove uses of publicKeyHex editorial pending close

Remove uses of publicKeyHex

~~ISSUE 281~~: Specifications needed for supported key representations publicKeyJwk, publcKeyPem, and publicKeyBase58 editorial **keys** **pending close**

Specifications needed for supported key representations publicKeyJwk, publcKeyPem, and publicKeyBase58

ISSUE 282

Added CBOR section

~~ISSUE 283~~: Verification method block should be a first citizen like public keys **PR exists**

Verification method block should be a first citizen like public keys

~~ISSUE 289~~: Should DID Methods expose Proof Purposes for DID Operations? **question**

Should DID Methods expose Proof Purposes for DID Operations?

ISSUE 291: PING Horizontal Review **horizontal review**

PING Horizontal Review

ISSUE 292: Horizontal Review Tracking **horizontal review**

Horizontal Review Tracking

~~ISSUE 293~~: Remove `proof` **PR exists**

Remove `proof`

~~ISSUE 294~~: Create a seperate top level block for defining proof purposes **pending close**

Create a seperate top level block for defining proof purposes

ISSUE 295

Define simple type-less resolution function

ISSUE 296

Define resolution function with data types

ISSUE 297

Define resolution function with data types and property values

ISSUE 298

Define resolution function with data types, property values, and simple metadata structures

ISSUE 299

Define resolution function with data types, property values, and full metadata structures

ISSUE 300

Define resolution function with data types, property values, and full metadata structures, without transformation

B. 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.

B.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:
http://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):
.did

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 DID Core v1.0, Fragment [DID-CORE].

B.2 application/did+ld+json §

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:
http://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):
.did

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].

B.3 application/did+cbor §

Type name:
application

Subtype name:

did+cbor

Required parameters:

None

Optional parameters:

None

Encoding considerations:

See RFC 7049, section 4.2.

Security considerations:

See RFC 7049, section 10 [RFC7049].

Interoperability considerations:

Not Applicable

Published specification:

http://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):

.did

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, Jonathan Holt

Change controller:

W3C

Fragment identifiers used with application/did+cbor are treated according to the rules defined in DID Core v1.0, Fragment [DID-CORE].

B.4 application/did+dag+cbor §

Type name:
application

Subtype name:
did+dag+cbor

Required parameters:
None

Optional parameters:
None

Encoding considerations:
See RFC 7049, section 4.2.

Security considerations:
See RFC 7049, section 10 [RFC7049].

Interoperability considerations:
Not Applicable

Published specification:
http://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):
.did

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, Jonathan Holt

Change controller:

W3C

Fragment identifiers used with application/did+cbor are treated according to the rules defined in DID Core v1.0, Fragment [DID-CORE].

C. References §

C.1 Normative references §

[BASE58]

The Base58 Encoding Scheme. Manu Sporny. IETF. December 2019. Internet-Draft. URL: https://tools.ietf.org/html/draft-msporny-base58

[DID-CORE]

Decentralized Identifiers (DIDs) v1.0. Drummond Reed; Manu Sporny; Markus Sabadello; Dave Longley; Christopher Allen; Jonathan Holt. W3C. 1 October 2020. W3C Working Draft. URL: https://www.w3.org/TR/did-core/

[DID-SPEC-REGISTRIES]

DID Specification Registries. Orie Steele; Manu Sporny. Decentralized Identifier Working Group. W3C Editor's Draft. URL: https://w3c.github.io/did-spec-registries/

[INFRA]

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

[JSON-LD]

JSON-LD 1.0. Manu Sporny; Gregg Kellogg; Markus Lanthaler. W3C. 16 January 2014. W3C Recommendation. URL: https://www.w3.org/TR/json-ld/

[JSON-LD11]

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

[RFC2119]

Key words for use in RFCs to Indicate Requirement Levels. S. Bradner. IETF. March 1997. Best Current Practice. URL: https://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://tools.ietf.org/html/rfc3552

[RFC3986]

Uniform Resource Identifier (URI): Generic Syntax. T. Berners-Lee; R. Fielding; L. Masinter. IETF. January 2005. Internet Standard. URL: https://tools.ietf.org/html/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

[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

[RFC7049]

Concise Binary Object Representation (CBOR). C. Bormann; P. Hoffman. IETF. October 2013. Proposed Standard. URL: https://tools.ietf.org/html/rfc7049

[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

[RFC8141]

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

[RFC8152]

CBOR Object Signing and Encryption (COSE). J. Schaad. IETF. July 2017. Proposed Standard. URL: https://tools.ietf.org/html/rfc8152

[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://tools.ietf.org/html/rfc8259

[RFC8610]

Concise Data Definition Language (CDDL): A Notational Convention to Express Concise Binary Object Representation (CBOR) and JSON Data Structures. H. Birkholz; C. Vigano; C. Bormann. IETF. June 2019. Proposed Standard. URL: https://tools.ietf.org/html/rfc8610

[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/

C.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-USE-CASES]

Decentralized Identifier Use Cases. Joe Andrieu; Kim Hamilton Duffy; Ryan Grant; Adrian Gropper. Decentralized Identifier Working Group. W3C Editor's Draft. URL: https://w3c.github.io/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-02

[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

[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

[RFC4648]

The Base16, Base32, and Base64 Data Encodings. S. Josefsson. IETF. October 2006. Proposed Standard. URL: https://tools.ietf.org/html/rfc4648

[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

[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

[RFC7515]

JSON Web Signature (JWS). M. Jones; J. Bradley; N. Sakimura. IETF. May 2015. Proposed Standard. URL: https://tools.ietf.org/html/rfc7515

[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/

↑