

Decentralized Identifiers (DIDs)

v1.0



Core Data Model and Syntaxes

W3C Working Draft 09 December 2019

This version:

<https://www.w3.org/TR/2019/WD-did-core-20191209/>

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/2019/WD-did-core-20191127/>

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](#))

Participate:

[GitHub w3c/did-core](#)

[File a bug](#)

[Commit history](#)

[Pull requests](#)

Copyright © 2019 W3C® ([MIT](#), [ERCIM](#), [Keio](#), [Beihang](#)). W3C [liability](#), [trademark](#) and [permissive document license](#) rules apply.

Abstract

Decentralized identifiers (DIDs) are a new type of identifier to provide verifiable, decentralized digital identity. These new identifiers are designed to enable the controller of a DID to prove control over it and to be implemented independently of any centralized registry, identity provider, or certificate authority. DIDs are URLs that relate a DID subject to a DID document allowing trustable interactions with that subject. DID documents are simple documents describing how to use that specific DID. 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 with the DID subject.

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 [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 [1 March 2019 W3C Process Document](#).

Table of Contents

- 1. Introduction**
 - 1.1 A Simple Example
 - 1.2 Design Goals
 - 1.3 Interoperability
- 2. Terminology**
- 3. Data Model**

- 3.1 Identifier
- 3.2 DID Document
- 3.3 Cryptographic Keys and Other Verification Methods
- 3.4 Services

4. Decentralized Identifiers (DIDs)

- 4.1 Generic DID Syntax
- 4.2 Method-Specific Syntax
- 4.3 Generic DID Parameter Names
- 4.4 Method-Specific DID Parameter Names
- 4.5 Path
- 4.6 Query
- 4.7 Fragment
- 4.8 Normalization
- 4.9 Persistence

5. DID Documents

- 5.1 Contexts
- 5.2 DID Subject
- 5.3 Public Keys
- 5.4 Authentication
- 5.5 Authorization and Delegation
- 5.6 Service Endpoints
- 5.7 Created
- 5.8 Updated
- 5.9 Proof
- 5.10 Extensibility

6. DID Document Syntax

- 6.1 JSON
- 6.2 JSON-LD

7. DID Methods

- 7.1 DID Method Schemes
- 7.2 DID Operations
 - 7.2.1 Create
 - 7.2.2 Read/Verify

- 7.2.3 Update
- 7.2.4 Deactivate
- 7.3 Unique DID Method Names

8. DID Resolvers

9. Security Considerations

- 9.1 Requirements of DID Method Specifications
- 9.2 Choosing DID Resolvers
- 9.3 Binding of Identity
 - 9.3.1 Proving Control of a DID and DID Document
 - 9.3.2 Proving Control of a Public Key
 - 9.3.3 Authentication and Verifiable Claims
- 9.4 Authentication Service Endpoints
- 9.5 Non-Repudiation
- 9.6 Notification of DID Document Changes
- 9.7 Key and Signature Expiration
- 9.8 Key Revocation and Recovery
- 9.9 The Role of Human-Friendly Identifiers
- 9.10 Immutability

10. Privacy Considerations

- 10.1 Requirements of DID Method Specifications
- 10.2 Keep Personally-Identifiable Information (PII) Private
- 10.3 DID Correlation Risks and Pseudonymous DIDs
- 10.4 DID Document Correlation Risks
- 10.5 Herd Privacy

11. Future Work

- 11.1 Upper Limits on DID Character Length
- 11.2 Equivalence
- 11.3 Timestamps
- 11.4 Time Locks and DID Document Recovery
- 11.5 Smart Signatures
- 11.6 Verifiable Claims
- 11.7 Alternate Serializations and Graph Models

- A. **Current Issues**
- B. **Registries**
- C. **Real World Example**
- D. **References**
 - D.1 Normative references
 - D.2 Informative references

1. Introduction §

This section is non-normative.

Conventional [identity management](#) systems are based on centralized authorities such as corporate [directory services](#), [certificate authorities](#), or [domain name registries](#). From the standpoint of cryptographic trust verification, each of these centralized authorities serves as its own [root of trust](#). To make identity management work across these systems requires implementing [federated identity management](#).

The emergence of [distributed ledger technology](#) (DLT) and [blockchain technology](#) provides the opportunity for fully [decentralized identity management](#). In a decentralized identity system, entities (that is, discrete identifiable units such as, but not limited to, people, organizations, and things) are free to use any shared root of trust. Globally distributed ledgers, decentralized P2P networks, or other systems with similar capabilities, provide the means for managing a root of trust without introducing a centralized authority or a single point of failure. In combination, [DLTs](#) and [decentralized identity management](#) systems enable any entity to create and manage their own identifiers on any number of distributed, independent roots of trust.

Entities are identified by [decentralized identifiers](#) ([DIDs](#)), and can authenticate using proofs (for example, digital signatures, privacy-preserving biometric protocols, and so on). [DIDs](#) point to [DID documents](#). A [DID document](#) contains a set of [service endpoints](#) for interacting with the entity that the [DID](#) identifies (that is, the [DID subject](#)). Following the guidelines of [Privacy by Design](#), any entity can have as many [DIDs](#) (and corresponding [DID documents](#) and [service endpoints](#)) as necessary to respect the entity's desired separation of identities, personas, and contexts (in the everyday sense of these words).

[DID methods](#) are the mechanism by which a [DID](#) and its associated [DID document](#) are created, read, updated, and deactivated on a specific [distributed ledger](#) or network. [DID methods](#) are defined using separate [DID method](#) specifications.

This design eliminates dependence on centralized registries for identifiers as well as centralized certificate authorities for key management, which is the standard in hierarchical [PKI \(public key infrastructure\)](#). In cases where the [DID registry](#) is a [distributed ledger](#), each entity can serve as its own root authority. This architecture is referred to as [DPKI \(decentralized PKI\)](#).

NOTE


[DID methods](#) can also be developed for identifiers registered in federated or centralized identity management systems. Indeed, all types of identifier systems can add support for [DIDs](#). This creates an interoperability bridge between the worlds of centralized, federated, and [decentralized identifiers](#).

The first purpose of this specification is to define the generic [DID scheme](#) and a generic set of operations on [DID documents](#) that can be implemented for any [DID registry](#). The second purpose of this specification is to define the conformance requirements for a [DID method](#) specification. The [DID method](#) specification is a separate document that defines a specific [DID scheme](#) and specific set of [DID document](#) operations for a specific [DID registry](#).

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 for resolving and deactivating [DIDs](#) on, and writing [DID documents](#) to, the appropriate [DID registry](#).

The hierarchical design of a generic [DID](#) specification with specific [DID method](#) specifications introduces some of the same concepts as the [URI](#) specification:

- [DIDs](#) from different [DID methods](#) might not be interoperable, just as [URIs](#) from different URI schemes might not be interoperable.
- Entities might need multiple [DIDs](#) to support different relationships because the other party might only support certain [DID methods](#), in the same way that some browsers might only support certain URI schemes.
- Entities might need multiple [DIDs](#) to support the different cryptographic schemes of different [DID methods](#) because not all parties will support the same cryptographic schemes, in the same way that not all browsers support the same URI schemes.
- Managing multiple [DIDs](#), and tracking which [DID](#) belongs to which relationship, under which cryptographic scheme, introduces similar logistical challenges as managing multiple web addresses and tracking which address belongs to which website, or tracking which email address belongs to which relationship. 

For a list of [DID methods](#) and their corresponding specifications, see the DID Method Registry [[DID-METHOD-REGISTRY](#)].

1.1 A Simple Example §

This section is non-normative.

A [DID](#) is a simple text string consisting of three parts, the:

- URL 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 entity in control of the [DID](#), as well as services that can be used to interact with the entity.

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": "RsaVerificationKey2018",
    "controller": "did:example:123456789abcdefghi",
    "publicKeyPem": "-----BEGIN PUBLIC KEY...END PUBLIC KEY-----\r\n"
  }],
  "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	

Goal	Description
	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 relying parties to depend on DID documents for their required level of assurance.
Proof-based	Enable DID subjects 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 Interoperability §

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 [DID methods](#) will be determined by evaluating each [DID method](#) specification to determine, at a minimum, that the:

- [DID method](#) name is unique and not used by an existing, incompatible [DID method](#).
- Required operations are supported.
- Operations requiring descriptions are described.
- Specification is specific, detailed, and complete enough for independent implementation.
- Specification contains sections describing security and privacy considerations.

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.

2. Terminology §

This section is non-normative.

This document attempts to communicate the concepts outlined in the [decentralized identifier](#) space by using specialized terms to discuss specific concepts. This terminology is included below and linked to throughout the document to aid the reader:

blockchain

A specific type of [distributed ledger technology](#) (DLT) that stores ledger entries in blocks of transactions, which 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.

decentralized identifier (DID)

A globally unique identifier that does not require a centralized registration authority because it is registered with [distributed ledger technology](#) (DLT) or other form of decentralized network. The generic format of a DID is defined in this specification. A specific [DID scheme](#) is defined in a [DID method](#) specification.

decentralized identity management

[Identity management](#) based on [decentralized identifiers](#). Decentralized identity management extends the identifier creation authority beyond the traditional roots of trust required by [X.500 directory services](#), the [Domain Name System](#), and most national identification systems.

decentralized public key infrastructure (DPKI)

Public key infrastructure based on [decentralized identifiers](#) and identity records (for example, [DID documents](#)) containing verifiable [public key descriptions](#).

DID controller

The entity, or a group of entities, in control of a [DID](#) or [DID document](#). Note that the [DID controller](#) might include the [DID subject](#).

DID document

A set of data describing the [DID subject](#), including mechanisms, such as public keys and pseudonymous biometrics, that the [DID subject](#) can use to authenticate itself and prove their association with the [DID](#). A DID document might also contain other [attributes](#) or [claims](#) describing the subject. These documents are graph-based data structures that are typically expressed using [JSON-LD](#), but can be expressed using other compatible graph-based data formats.

DID fragment

The portion of a [DID URL](#) that follows the first hash sign character (<#>). DID fragment syntax is identical to the URI fragment syntax.

DID method

A definition of how a specific [DID scheme](#) can be implemented on a specific [distributed ledger](#) or network, including the precise methods by which [DIDs](#) are resolved and deactivated and [DID documents](#) are written and updated.

DID path

The portion of a [DID URL](#) that begins with and includes the first forward slash character ([/](#)). DID path syntax is identical to the URI path syntax.

DID query

The portion of a [DID URL](#) that follows the first question mark character ([?](#)). DID query syntax is identical to the URI query syntax.

DID registry

A role a system performs to mediate the creation, verification, updating, and deactivation of [decentralized identifiers](#). A DID registry is a type of verifiable data registry. For more information, see [\[VC-DATA-MODEL\]](#).

DID resolver

A system that is capable of retrieving a [DID document](#) for a given [DID](#). These systems are specified in the DID Resolution specification [\[DID-RESOLUTION\]](#).

DID scheme

The formal syntax of a [decentralized identifier](#). The generic DID scheme is defined in this specification. Separate [DID method](#) specifications define a specific DID scheme that works with that specific [DID method](#).

DID subject

The entity the [DID document](#) is about. That is, the entity identified by the [DID](#) and described by the [DID document](#).

DID URL

A [DID](#) plus an optional [DID path](#), optional [?](#) character followed by a [DID query](#), and optional <#> character followed by a [DID fragment](#).

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.

extensible data interchange (XDI)

A semantic graph format and semantic data interchange protocol defined by the [OASIS XDI Technical Committee](#).

JSON Pointer

Defines a string syntax for identifying a specific value within a JavaScript Object Notation (JSON) document, as defined in [\[RFC6901\]](#).

public key description

A JSON object contained inside a [DID document](#) that contains all the metadata necessary to use a public key or verification key.

service endpoint

A network address at which a service operates on behalf of a [DID subject](#). Examples of specific services include discovery services, social networks, file storage services, and verifiable claim repository services. Service endpoints might also be provided by a generalized data interchange protocol, such as [extensible data interchange](#).

Uniform Resource Identifier (URI)

An identifier, as defined by [\[RFC3986\]](#).

Universally Unique Identifier (UUID)

An identifier, as defined by [\[RFC4122\]](#).

3. Data Model §

This section is non-normative.

This section outlines the [decentralized identifier](#) data model concepts, specifically, how keys, services, and the [DID subject](#) are related to the [DID document](#).

For more information about how the data model can be extended, see Section [§ 5.10 Extensibility](#).

3.1 Identifier §

Identifiers are used in the data model to identify specific instances of people, organizations, devices, keys, services, and things in general. Identifiers are typically URLs, or more generally, [URIs](#). Non-[URI](#)-based identifiers are not advised because, while the data model supports them, they are not easy to use with other Internet-based identifiers.

3.2 DID Document §

A [DID document](#) is the resource that is associated with a [decentralized identifier](#) (DID). [DID documents](#) typically express verification methods (such as public keys) and services that can be used to interact with a [DID controller](#).

A [DID document](#) is serialized according to a particular syntax, as outlined in Section [§ 6. DID Document Syntax](#)). The [DID](#) itself is contained in the `id` property.

The properties that can be present in a [DID document](#) are outlined in Section [§ 5. DID Documents](#) .

The properties present in a [DID document](#) can be updated according to the applicable operations outlined in Section [§ 7. DID Methods](#) .

3.3 Cryptographic Keys and Other Verification Methods §

A [DID document](#) can express cryptographic keys and other verification methods, 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. Other information can be expressed, such as status information for the key (for example, whether it is suspended or revoked), or other attributes that enable one to determine whether it is a hardware-backed cryptographic key.

Regarding cryptographic key material, public keys can be included in a [DID document](#) using, for example, the `publicKey` or `authentication` properties, depending on what they are to be used for. Each public key has an identifier (`id`) of its own, a `type`, and a `controller`, as well as other properties that depend on the type of key it is. For more information, see Section [§ 5.3 Public Keys](#) .

3.4 Services §

[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](#). 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](#), as well as a [serviceEndpoint](#) with a [URI](#) or other properties describing the service.

For more information, see Section [§ 5.6 Service Endpoints](#).

4. Decentralized Identifiers (DIDs) §

The concept of a globally unique [decentralized identifier](#) is not new. [Universally Unique Identifiers](#) (UUIDs) were first developed in the 1980s and later became a standard feature of the Open Software Foundation's [Distributed Computing Environment](#). [UUIDs](#) achieve global uniqueness without a centralized registry service by using an algorithm that generates 128-bit values with sufficient entropy that the chance of collision are infinitesimally small. [UUIDs](#) are formally specified in [\[RFC4122\]](#) as a specific type of Unified Resource Name (URN).

A [DID](#) is similar to a [UUID](#) except that:

- Like a URL, it can be resolved or dereferenced to a standard resource describing the subject. That is, a [DID document](#). For more information, see Section [§ 5. DID Documents](#).
- Unlike most resources returned when dereferencing URLs, a [DID document](#) usually contains cryptographic material enabling authentication of the [DID subject](#).

4.1 Generic DID Syntax §

The generic DID scheme is a URI scheme conformant with [RFC3986]. The DID scheme specializes only the scheme and authority components of a DID URL. The path-abempty, query, and fragment components are identical to the ABNF rules defined in [RFC3986].

NOTE

The term DID refers only to the URI conforming to the did rule in the ABNF below. A DID always identifies the DID subject. The term DID URL, defined by the did-url rule, refers to a URL that begins with a DID followed by one or more additional components. A DID URL always identifies the resource to be located.

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 *( ":" *idchar )
idchar             = ALPHA / DIGIT / "." / "-" / "_"
did-url            = did *( ";" param ) path-abempty [ "?" query ]
                  [ "#" fragment ]
param              = param-name [ "=" param-value ]
param-name         = 1*param-char
param-value        = *param-char
param-char         = ALPHA / DIGIT / "." / "-" / "_" / ":" /
                  pct-encoded
```

ISSUE 34: Should DID syntax allow an empty "method-specific-id"? discuss

The grammar currently allows an empty method-specific-id, e.g., did:example: would be a valid DID that could identify the DID method itself.

4.2 Method-Specific Syntax §

A [DID method](#) specification *MUST* further restrict the generic [DID](#) syntax by defining its own **method-name** and its own **method-specific-id** syntax. For more information, see Section § 7. [DID Methods](#).

4.3 Generic DID Parameter Names §

The [DID URL](#) syntax supports a simple, generalized format for parameters based on the matrix parameter syntax ([\[MATRIX-URIS\]](#)). The ABNF above does not specify any parameter names (the **param-name** rule).

Some generic 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.

Parameter names that are completely method-specific are described in Section § 4.4 [Method-Specific DID Parameter Names](#).

The following table defines a set of generic DID parameter names.

Generic DID	
Parameter Name	Description
hl	A resource hash of the DID document to add integrity protection, as specified in [HASHLINK] .
service	Identifies a service from the DID document by service ID.
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 .
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 exact processing rules for these parameters are specified in [\[DID-RESOLUTION\]](#).

NOTE

There might be additional parameters or options that are not part of the DID URL, but instead passed to a [DID resolver](#) *out-of-band*, that is, using a resolution protocol or some other mechanism. Such options could for example, control caching or the desired format of a resolution result. This is similar to HTTP, where caching or result format are expressed as HTTP headers instead of as part of an HTTP URL. The important distinction is that DID parameters that are part of the [DID URL](#) specify what resource is being identified, whereas [DID resolver](#) options that are not part of the [DID URL](#) control how that resource is dereferenced.

4.4 Method-Specific DID Parameter Names §

A [DID method](#) specification *MAY* specify additional method-specific parameter names. A method-specific parameter name *MUST* be prefixed by the method name, as defined by the [method-name](#) rule.

For example, if the method `did:foo:` defines the parameter `bar`, the parameter name must be `foo:bar`. An example [DID URL](#) using this method and this method-specific parameter would be as shown below.

EXAMPLE 3

```
did:foo:21tDAKCERh95uGgKbJNHYP;foo:bar=high
```

ISSUE 35: Use colon separator or kebab-case for method-specific DID parameter names? discuss

Consider using kebab-case style instead of colon separator, e.g., `foo-bar` instead of `foo:bar`.

A method-specific parameter name defined by one [DID method](#) *MAY* be used by other [DID methods](#).

EXAMPLE 4

```
did:example:21tDAKCERh95uGgKbJNHYP;foo:bar=low
```

Method-specific parameter names *MAY* be combined with generic parameter names in any order.

EXAMPLE 5

```
did:example:21tDAKCERh95uGgKbJNHYP;service=agent;foo:bar=high
```

Both [DID method](#) namespaces and method-specific parameter namespaces *MAY* include colons, so they might be partitioned hierarchically, as defined by a [DID method](#) specification. The following example [DID URL](#) illustrates both.

EXAMPLE 6

```
did:foo:baz:21tDAKCERh95uGgKbJNHYP;foo:baz:hex=b612
```

ISSUE 36: Details on the use of method-specific DID parameters [discuss](#)

Review what exactly we want to say about method-specific parameters defined by one method but used in a [DID URL](#) with a different method. Also discuss hierarchical method namespaces in DID parameter names.

4.5 Path §

A generic [DID path](#) is identical to a URI path and *MUST* conform to the [path-abempty](#) ABNF rule in [RFC3986]. A [DID path](#) *SHOULD* be used to address resources available through a [service endpoint](#). For more information, see Section [§ 5.6 Service Endpoints](#).

A specific [DID scheme](#) *MAY* specify ABNF rules for [DID paths](#) that are more restrictive than the generic rules in this section.

EXAMPLE 7

did:example:123456/path

4.6 Query §

A generic [DID query](#) is identical to a URI query and *MUST* conform to the [query](#) ABNF rule in [RFC3986]. A [DID query](#) *SHOULD* be used to address resources available through a [service endpoint](#). For more information, see Section § 5.6 [Service Endpoints](#).

A specific [DID scheme](#) *MAY* specify ABNF rules for [DID queries](#) that are more restrictive than the generic rules in this section.

EXAMPLE 8

did:example:123456?query=true

4.7 Fragment §

A generic [DID fragment](#) is identical to a URI fragment and *MUST* conform to the [fragment](#) ABNF rule in [RFC3986]. Implementers are strongly discouraged from using a [DID fragment](#) for anything other than a method-independent reference into the [DID document](#) to identify a component of a [DID document](#) (for example, a unique [public key description](#) or [service endpoint](#)). To resolve this reference, the complete [DID URL](#) including the [DID fragment](#) *MUST* be used as input to the [DID URL](#) dereferencing algorithm for the target component in the [DID document](#) object. For more information, see [DID-RESOLUTION].

A specific [DID scheme](#) *MAY* specify ABNF rules for [DID fragments](#) that are more restrictive than the generic rules in this section.

Implementations need not rely on graph-based processing of [DID documents](#) to locate metadata contained in the [DID document](#) when the [DID](#) includes a [DID fragment](#). Tree-based processing can be used instead.

Implementations *SHOULD NOT* prevent the use of [JSON Pointer](#) ([RFC6901]).

EXAMPLE 9

did:example:123456#oidc

4.8 Normalization §

For the broadest interoperability, make DID normalization as simple and universal as possible:

- The DID scheme name *MUST* be lowercase.
- The DID method name *MUST* be lowercase.
- Case sensitivity and normalization of the value of the **method-specific-id** rule in Section § 4.1 Generic DID Syntax *MUST* be defined by the governing DID method specification.

4.9 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 DID registries over time, thus maintaining its persistence independent of any particular system. However, registering the same identifier on multiple DID registries introduces extremely hard entityship and start-of-authority (SOA) problems. It also greatly increases implementation complexity for developers.

To avoid these issues, it is *RECOMMENDED* that DID method specifications only produce DIDs and DID methods bound to strong, stable DID registries capable of making the highest level of commitment to persistence of the DID and DID method over time.

NOTE

Although not included in this version, future versions of this specification might support a [DID document](#) `equivID` property to establish verifiable equivalence relations between [DIDs](#) representing the same subject on multiple [DID registries](#). Such equivalence relations can produce the practical equivalent of a single persistent abstract [DID](#). For more information, see Section [§ 11. Future Work](#).

5. DID Documents §

A [DID](#) points to a [DID document](#). [DID documents](#) are the serialization of the [§ 3. Data Model](#). The following sections define the properties in a [DID document](#), including whether these properties are required or optional.

5.1 Contexts §

When two software systems need to exchange data, they need to use terminology and a protocol that both systems understand. The `@context` property ensures that two systems operating on the same DID document are using mutually agreed terminology.

[DID documents](#) *MUST* include the `@context` property.

NOTE: The JSON-LD Context

More information about the [JSON-LD Context](#) in general can be found in the [[JSON-LD](#)] specification.

`@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 the [URIs](#) results in a document containing machine-readable information about the context.

Example:

EXAMPLE 10

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

DID method specifications *MAY* define their own JSON-LD contexts. However it is *NOT RECOMMENDED* to define a new context unless necessary to properly implement the method. Method-specific contexts *MUST NOT* override the terms defined in the generic DID context.

5.2 DID Subject §

The DID subject is denoted with the `id` property. This is the entity that the DID document is about, i.e., 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 single valid DID.

Example:

EXAMPLE 11

```
{
  "id": "did:example:21tDAKCERh95uGgKbJNHYP"
}
```

NOTE

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

5.3 Public Keys §

Public keys are used for digital signatures, encryption and other cryptographic operations, which in turn are the basis for purposes such as authentication (see Section § 5.4 [Authentication](#)) or establishing secure communication with [service endpoints](#) (see Section § 5.6 [Service Endpoints](#)). In addition, public keys may play a role in authorization mechanisms of [DID](#) CRUD operations (see Section § 7.2 [DID Operations](#)), which can be defined by [DID method](#) specifications.

A [DID document](#) *MAY* include a **publicKey** property. If so:

publicKey

The value of the **publicKey** property *MUST* be an array of public key objects. Each public key object *MUST* have the **type**, **controller**, and specific public key properties, and *SHOULD* have an **id** property. The object *MAY* include additional properties.

The value of the **id** property (if present) *MUST* be a URI. The array of public keys *MUST NOT* contain multiple entries with the same **id**. A [DID document](#) processor *MUST* produce an error in that case.

The value of the **type** property *MUST* be exactly one public key value (see available key types in Appendix § B. [Registries](#)).

The value of the **controller** property, which identifies the controller of the corresponding private key, *MUST* be a valid DID.

All public key properties *MUST* be from the Linked Data Cryptographic Suite Registry. A registry of key types and formats is available in Appendix § B. [Registries](#)

NOTE

The following is a non-exhaustive list of public key properties used by the community: **publicKeyPem**, **publicKeyJwk**, **publicKeyHex**, **publicKeyBase64**, **publicKeyBase58**, **publicKeyMultibase**, **ethereumAddress**.

If a public key does not exist in the [DID document](#), it *MUST* be assumed the key has been revoked or is invalid. The [DID document](#) *MAY* contain revoked keys. A [DID document](#) that contains a revoked key *MUST* also contain or refer to the revocation information for the key (e.g., a revocation list). Each [DID method](#) specification is expected to detail how revocation is performed and tracked.

Example:

EXAMPLE 12: Various public keys

```
{
  "@context": ["https://www.w3.org/ns/did/v1", "https://w3id.org/security/v1"],
  "id": "did:example:123456789abcdefghi",
  ...
  "publicKey": [{
    "id": "did:example:123456789abcdefghi#keys-1",
    "type": "RsaVerificationKey2018",
    "controller": "did:example:123456789abcdefghi",
    "publicKeyPem": "-----BEGIN PUBLIC KEY...END PUBLIC KEY-----\r\n"
  }, {
    "id": "did:example:123456789abcdefghi#keys-2",
    "type": "Ed25519VerificationKey2018",
    "controller": "did:example:pqrstuvwxyz0987654321",
    "publicKeyBase58": "H3C2AVvLMv6gmMNam3uVAjZpfkcJCwDwnZn6z3wXmqPV"
  }, {
    "id": "did:example:123456789abcdefghi#keys-3",
    "type": "Secp256k1VerificationKey2018",
    "controller": "did:example:123456789abcdefghi",
    "publicKeyHex": "02b97c30de767f084ce3080168ee293053ba33b235d7116a3263d29f1450936b71"
  }],
  ...
}
```

A key *MAY* be *embedded* or *referenced* in a [DID document](#). For example, the **authentication** property may refer to keys in both ways:

EXAMPLE 13: Various public keys

```
{
  ...

  "authentication": [
    // this key is referenced, it may be used for more than one proof purpose
    "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"
    }
  ],
  ...
}
```

The algorithm to use when processing a **publicKey** property in a [DID document](#) is:

1. Let *value* be the data associated with the **publicKey** property and initialize *result* to **null**.
2. If *value* is an object, the key material is embedded. Set *result* to *value*.
3. If *value* is a string, the key is included by reference. Assume *value* is a URL.
 1. Dereference the URL and retrieve the **publicKey** properties associated with the URL (e.g., process the **publicKey** property at the top-level of the dereferenced document).
 2. Iterate through each public key object.
 1. 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 *result*'s **type** property, throw an error.

NOTE

While the **controller** field may seem redundant in some of the examples above, keys may be expressed in a [DID document](#) where the controller is described in another [DID document](#). [Linked Data Proof libraries](#) typically expect the **controller** field to always exist and may throw an exception if it is missing. Furthermore, per the requirement that [DID documents](#) be interpretable as either a graph or a tree, a default **controller** field cannot be inferred by using a key's position in a tree.

NOTE

Caching and expiration of the keys in a [DID document](#) is entirely the responsibility of [DID resolvers](#) and other clients. See Section § 8. [DID Resolvers](#).

5.4 Authentication §

Authentication is the mechanism by which [DID controllers](#) can cryptographically prove that they are associated with a [DID](#). For more information, see Section § 9.3 [Binding of Identity](#). Note that Authentication is separate from Authorization because [DID controllers](#) might wish to enable others to update their [DID document](#) (for example, to assist with key recovery, as discussed in Section § 9.8 [Key Revocation and Recovery](#)) without enabling them to prove control (and thus be able to impersonate the [DID controller\(s\)](#)).

A [DID document](#) *MAY* include an **authentication** property. If so:

authentication

The value of the **authentication** property *SHOULD* be an array of verification methods. Each verification method *MAY* be embedded or referenced. An example of a verification method is a public key (see Section § 5.3 [Public Keys](#)).

Example:

EXAMPLE 14: Authentication field 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"
    }
  ],
  ...
}
```

5.5 Authorization and Delegation §

Authorization is the mechanism used to state how operations may be performed on behalf of the DID subject. Delegation is the mechanism that the subject uses to authorize others to act on their behalf. Note that Authorization is separate from Authentication as explained in Section § 5.4 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.

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

There are at least two suggested methods for implementing Authorization and Delegation, which may be layered:

1. A [DID registry](#) could implement a coarse grained **controller** pattern by enabling [DID documents](#) to express the [DID](#) of another [DID controller](#) that controls it, or additionally,
2. A [DID registry](#) could implement a Capabilities-based approach that enables further fine-grained control of authorization and delegation.

Example:

EXAMPLE 15: 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/"
  }]
}
```

5.6 Service Endpoints §

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](#) wishes to advertise, including [decentralized identity management](#) services for further discovery, authentication, authorization, or interaction.

A [DID document](#) *MAY* include a **service** property. If so:

service

The value of the **service** property *SHOULD* be an array of [service endpoint](#) objects. Each service endpoint *MUST* have **type** and **serviceEndpoint** properties, *SHOULD* have an **id** property, and *MAY* include additional properties.

The value of the **id** property (if present) *MUST* be a URI. The array of [service endpoints](#) *MUST NOT* contain multiple entries with the same **id**. A [DID document](#) processor *MUST* produce an error in that case.

The value of the **serviceEndpoint** property *MUST* be a JSON-LD object or 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.

It is expected that the [service endpoint](#) protocol is published in an open standard specification.

Example:

EXAMPLE 16: 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",
    "publicKey": "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:123456789abc  
defg"  
  }]  
}
```

See Sections § 7.1 [DID Method Schemes](#) and § 5.4 [Authentication](#) for further security considerations regarding authentication [service endpoints](#).

5.7 Created §

A [DID document](#) *SHOULD* include a **created** property. If so:

created

The value of **created** *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".

Example:

EXAMPLE 17

```
{  
  "created": "2002-10-10T17:00:00Z"  
}
```

5.8 Updated §

Standard metadata for identifier records includes a timestamp of the most recent change.

A [DID document](#) *SHOULD* include an **updated** property. If so:

updated

The **updated** value *MUST* follow the same formatting rules as the **created** property (see Section § 5.7 [Created](#)).

Example:

EXAMPLE 18

```
{
  "updated": "2016-10-17T02:41:00Z"
}
```

5.9 Proof §

A **proof** on a [DID document](#) is cryptographic proof of the integrity of the [DID document](#) according to either:

1. The [DID subject](#) as defined in Section § 5.2 [DID Subject](#), or;
2. The [DID controller](#) as defined in Section § 5.5 [Authorization and Delegation](#), if present.

This proof is NOT proof of the binding between a [DID](#) and a [DID document](#) (see Section § 9.3 [Binding of Identity](#)).

A [DID document](#) *MAY* include a **proof** property. If so:

proof

The value of **proof** *MUST* be a valid JSON-LD proof as defined by [Linked Data Proofs](#).

Example:

EXAMPLE 19: A signature-based proof

```
{
  "proof": {
    "type": "LinkedDataSignature2015",
    "created": "2016-02-08T16:02:20Z",
    "creator": "did:example:8uQhQMGzWxR8vw5P3UWH1ja#keys-1",
    "signatureValue": "QNB13Y7Q9...1tzjn4w=="
  }
}
```

5.10 Extensibility §

One of the goals of the [decentralized identifiers](#) data model is to enable permissionless innovation. This requires that the data model is extensible in a number of different ways:

- The requirement to model complex multi-entity relationships is provided through the use of a graph-based data model.
- The requirement to enable extending the machine-readable vocabularies used to describe information in the data model — without relying on a centralized system — is accomplished via the use of [\[LINKED-DATA\]](#).
- The requirement to support multiple types of cryptographic proof formats is accomplished via the use of Linked Data Proofs [\[LD-PROOFS\]](#), Linked Data Signatures [\[LD-SIGNATURES\]](#), and a variety of signature suites.
- The requirement to provide all of the extensibility mechanisms outlined above in a data format that is popular among software developers and web page authors is enabled via the use of [\[JSON-LD\]](#).

This approach to data modeling is often called an "open world assumption", meaning that anyone can say anything about any other thing. This approach often feels in conflict with building simple and predictable software systems. Balancing extensibility with program correctness is always more challenging with an open world assumption than it is with closed software systems.

The rest of this section describes how both extensibility and program correctness are achieved through a series of examples.

Let us assume that we start with the following [DID document](#):

[EXAMPLE 20](#): A simple DID document

```
{
  "@context": "https://example.org/example-method/v1",
  "id": "did:example:123456789abcdefghi",
  "publicKey": [{ ... }],
  "authentication": [{ ... }],
  "service": [{ ... }]
}
```

The contents of the `publicKey`, `authentication`, and `service` properties are not important for the purposes of this section. What is important is that the object above is a valid [DID document](#). Let's assume that a developer wanted to extend the [DID document](#) to express an additional piece of information: the subject's public photo stream.

The first thing that a developer would do is create a JSON-LD Context containing the new term:

EXAMPLE 21: An example JSON-LD Context

```
{
  "@context": {
    "PhotoStreamService": "https://example.com/vocab#PhotoStreamService"
  }
}
```

Now that the JSON-LD Context has been created, the developer *MUST* publish it somewhere that is accessible to any [DID document](#) processor. For this example, let us assume that the JSON-LD Context above is published at the following URL:

`did:example:contexts:987654321`. At this point, extending the first example in this section is a simple matter of including the context above and adding the new property to the [DID document](#).

EXAMPLE 22: A DID document with a custom extension

```
{
  "@context": "https://example.org/example-method/v1",
  "id": "did:example:123456789abcdefghi",
  "authentication": [ ... ],
  "service": [{
    "@context": "did:example:contexts:987654321",
    "id": "did:example:123456789abcdefghi#photos",
    "type": "PhotoStreamService",
    "serviceEndpoint": "https://example.org/photos/379283"
  }]
}
```

The examples so far have shown that it is easy to extend the [decentralized identifiers](#) data model in a permissionless and decentralized way. The mechanism also ensures that

[decentralized identifiers](#) created in this way prevent namespace conflicts and semantic ambiguity.

An extensibility model that is this dynamic does increase implementation burden. Software written for such a system will have to determine if accepting [DID documents](#) with extensions is acceptable based on the risk profile of the application. Some applications may choose to accept but ignore extensions, others may choose to only accept certain extensions, while highly secure environments may disallow extensions. These decisions are up to the application developers and are specifically not the domain of this specification.

Implementations *MUST* produce an error when an extension JSON-LD Context overrides the expanded URL for a term specified in this specification. To avoid the possibility of accidentally overriding terms, developers *SHOULD* scope their extensions. For example, the following extension scopes the new **PhotoStreamService** term so that it may only be used within the **service** property:

EXAMPLE 23: Scoping terms in a JSON-LD Context

```
{
  "@context": {
    "service": {
      "@id": "https://w3id.org/did#service",
      "@context": {
        "PhotoStreamService": "https://example.com/vocab#PhotoStreamService"
      }
    }
  }
}
```

Developers are urged to ensure that extension JSON-LD Contexts are highly available. Implementations that cannot fetch a context will produce an error. Strategies for ensuring that extension JSON-LD Contexts are always available include using content-addressed URLs for contexts, bundling context documents with implementations, or enabling aggressive caching of contexts.

6. DID Document Syntax §

A DID document *MUST* be a single JSON object conforming to [RFC8259]. Many of the concepts in this document were introduced by example using the JSON-LD syntax, a format for mapping JSON data into the RDF semantic graph model, as defined by [JSON-LD]. This section formalizes how the data model (described in Sections § 3. Data Model and § 5. DID Documents) is realized in JSON-LD.

Although syntactic mappings are provided for JSON and JSON-LD only, applications and services can use any other data representation syntax, such as JXD (JSON XDI Data, a serialization format for the XDI graph model), XML, YAML, or CBOR, that is capable of expressing the data model.

6.1 JSON §

The data model, as described in Section § 3. Data Model, can be encoded in Javascript Object Notation (JSON) [RFC8259] by mapping property values to JSON types as follows:

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

6.2 JSON-LD §

[JSON-LD] is a JSON-based format used to serialize Linked Data. The syntax is designed to easily integrate into deployed systems already using JSON, and provides a smooth upgrade path from JSON to JSON-LD. It is primarily intended to be a way to use Linked Data in Web-based programming environments, to build interoperable Web services, and to store Linked Data in JSON-based storage engines.

JSON-LD is useful when extending the data model described in this specification. Instances of the data model are encoded in JSON-LD in the same way they are encoded in JSON (see Section § 6.1 [JSON](#)), with the addition of the `@context` property. The [JSON-LD Context](#) is described in detail in the [\[JSON-LD\]](#) specification and its use is elaborated on in Section § 5.10 [Extensibility](#).

In general, the data model and syntaxes described in this document are designed such that developers can copy and paste examples into their software systems. The design goal of this approach is to provide a low barrier to entry while still ensuring global interoperability between a heterogeneous set of software systems. This section describes some of these approaches, which will likely go unnoticed by most developers, but whose details will be of interest to implementers. Noteworthy features provided by JSON-LD are:

- The `@id` and `@type` keywords are aliased to `id` and `type` respectively, enabling developers to use this specification as idiomatic JSON.
- 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.
- The `@protected` properties feature of JSON-LD 1.1 is used to ensure that terms defined by this specification cannot be overridden. This means that as long as the same `@context` declaration is made at the top of a [DID document](#), interoperability is guaranteed between implementations that use a JSON-LD processor and implementations that do not.

7. DID Methods §

7.1 DID Method Schemes §

A [DID method](#) specification *MUST* define exactly one method-specific [DID scheme](#), identified by exactly one method name. For more information, see the `method-name` rule in Section § 4.1 [Generic DID Syntax](#).

Because the method name is part of the [DID](#), short method names are preferred; the method name *SHOULD* be five characters or less. The method name might reflect the name of the [DID registry](#) to which the [DID method](#) specification applies. For more information, see Section § 7.3 [Unique DID Method Names](#).

The [DID method](#) specification for the method-specific [DID scheme](#) *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. The [DID](#), as defined by the `did` rule in Section § 4.1 [Generic DID Syntax](#), *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.

7.2 DID Operations §

To enable the full functionality of [DIDs](#) and [DID documents](#) on a particular [DID registry](#), a [DID method](#) specification *MUST* specify how each of the following CRUD operations is performed by a client. Specify each operation to the level of detail necessary to build and test interoperable client implementations with the [DID registry](#). These operations can effectively be used to perform all the operations required of a CKMS (cryptographic key management system). For example, key registration, key replacement, key rotation, key recovery, and key expiration.

Any [DID method](#) specification that does not support a specific operation, such as update or deactivate, *MUST* clearly state these limitations.

7.2.1 Create §

The [DID method](#) specification *MUST* specify how a client creates a [DID](#) and its associated [DID document](#) on the [DID registry](#), including all cryptographic operations necessary to establish proof of control.

7.2.2 Read/Verify §

The [DID method](#) specification *MUST* specify how a client uses a [DID](#) to request a [DID document](#) from the [DID registry](#), including how the client can verify the authenticity of the response.

7.2.3 Update §

The [DID method](#) specification *MUST* specify how a client can update a [DID document](#) on the [DID registry](#), including all cryptographic operations necessary to establish proof of control, *or* state that updates are not possible.

7.2.4 Deactivate §

The [DID method](#) specification *MUST* specify how a client can deactivate a [DID](#) on the [DID registry](#), including all cryptographic operations necessary to establish proof of deactivation, *or* state that deactivation is not possible.

7.3 Unique DID Method Names §

This section is non-normative.

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.

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, the [W3C Credentials Community Group](#) maintains a non-authoritative list of known [DID method](#) names and their associated specifications. For more information, see Appendix [§ B. Registries](#).

The [\[DID-METHOD-REGISTRY\]](#) is a tool for implementors to use when coming to consensus on a new method name, as well as an informative reference for software developers implementing [DID resolvers](#) for different [DID methods](#). For more information about [DID resolvers](#) see Section [§ 8. DID Resolvers](#). The [\[DID-METHOD-REGISTRY\]](#) is not a definitive or official list of [DID methods](#). Nonetheless, adding [DID method](#) names to the [\[DID-METHOD-REGISTRY\]](#) is encouraged so that other implementors and members of the community have a place to see an overview of existing [DID methods](#). The lightweight criteria for inclusion are documented in the [\[DID-METHOD-REGISTRY\]](#).

8. DID Resolvers §

A DID resolver is a software or hardware component with an API for resolving DIDs of at least one DID method. It executes the read operation for the DID method corresponding to the DID being resolved to obtain the authoritative DID document. For more information, see Section § 7.2.2 Read/Verify.

The interfaces and algorithms for resolving DIDs and dereferencing DID URLs are specified in [DID-RESOLUTION].

9. Security Considerations §

This section is non-normative.

NOTE: Note to implementers

During the Implementer's Draft stage, this section focuses on security topics that should be important in early implementations. The editors are also seeking feedback on threats and threat mitigations that should be reflected in this section or elsewhere in the spec. As the root identifier records for decentralized identifiers (DIDs) and DID documents are a vital component of decentralized identity management. They are also the foundational building blocks of decentralized public key infrastructure (DPKI) as an augmentation to conventional X.509 certificates. As such, DIDs are designed to operate under the general Internet threat model used by many IETF standards. We assume uncompromised endpoints, but allow messages to be read or corrupted on the network. Protecting against an attack when a system is compromised requires external key-signing hardware. See also Section § 9.8 Key Revocation and Recovery regarding key revocation and recovery. For their part, the DLTs hosting DIDs and DID documents have special security properties for preventing active attacks. Their design uses public/private key cryptography to allow operation on passively monitored networks without risking compromise of private keys. This is what makes DID architecture and decentralized identity possible.

9.1 Requirements of DID Method Specifications §

1. DID method specifications *MUST* include their own Security Considerations sections.

2. This section *MUST* consider all the requirements mentioned in section 5 of [\[RFC3552\]](#) (page 27) for the [DID](#) operations defined in the specification. In particular:

ISSUE

Discussions at Rebooting the Web of Trust 5 resulted in consensus to move Authorization to [DID method](#) specifications. It is currently expected that there will be an attempt to create a generalized authorization mechanism that is build on object capabilities.

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. If the protocol incorporates cryptographic protection mechanisms, it should be clearly indicated which portions of the data are protected and what the protections are (i.e., integrity only, confidentiality, and/or endpoint authentication, etc.). Some indication should also be given to what sorts of attacks the cryptographic protection is susceptible. Data which should be held secret (keying material, random seeds, etc.) should be clearly labeled. If the technology involves authentication, particularly user-host authentication, the security of the authentication method *MUST* be clearly specified.

3. This section *MUST* also 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 has been deployed.
4. This section *MUST* provide integrity protection and update authentication for all operations required by Section [§ 7.2 DID Operations](#).
5. 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.
6. 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.
7. [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 [\[RFC8141\]](#) — a persistent identifier that is assigned once to a resource and never

reassigned. In a security context this is particularly important since a [DID](#) may be used to identify a specific party subject to a specific set of authorization rights.

8. [DID methods](#) that introduce new authentication [service endpoint](#) types (Section [§ 5.6 Service Endpoints](#)) *SHOULD* consider the security requirements of the supported authentication protocol.

9.2 Choosing DID Resolvers §

The [\[DID-METHOD-REGISTRY\]](#) is an informative list of [DID method](#) names and their corresponding [DID method](#) specifications. Implementors should bear in mind that there is no central authority to mandate which [DID method](#) specification must be used with any particular [DID method](#) name, but can use the [\[DID-METHOD-REGISTRY\]](#) to make an informed decision when choosing which [DID resolver](#) implementations to use.

9.3 Binding of Identity §

9.3.1 Proving Control of a DID and DID Document §

Signatures are one method to 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 the following:

1. The [DID document](#) has not been tampered with since it was registered.
2. The [DID controller\(s\)](#) controlled the private key used for the signature at the time the signature was generated.

Proving control of a [DID](#), i.e., 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.

It is good practice to combine timestamps with signatures.

9.3.2 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](#) may 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](#).


9.3.3 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 company, for example with credentials with the same subject as that [DID](#), is out of scope for this specification. See the [\[VC-DATA-MODEL\]](#) instead.

9.4 Authentication Service Endpoints §


If a [DID document](#) publishes a [service endpoint](#) intended for authentication or authorization of the [DID subject](#) (section § 5.6 [Service Endpoints](#)), it is the responsibility of the [service endpoint](#) provider, subject, and/or relying party to comply with the requirements of the authentication protocol(s) supported at that [service endpoint](#). 


9.5 Non-Repudiation §

Non-repudiation of [DIDs](#) and [DID document](#) updates is supported under the assumption that:
(1) the subject is monitoring for unauthorized updates  See Section § 9.6 [Notification of DID](#)


[Document Changes](#)) and (2) the [subject](#) has had adequate opportunity to revert malicious updates according to the [DID method's access control mechanism](#) (section § 5.4 [Authentication](#)). This capability is further supported if timestamps are included (sections § 5.7 [Created](#) and § 5.8 [Updated](#)) and the target [DLT](#) system supports timestamps.

9.6 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](#), where there is no intermediary registrar or account provider to generate the notification, the following approaches are suggested:

1. [Subscriptions](#).  In the [DID registry](#) on which the [DID](#) is registered directly supports change notifications, this service can be offered to [DID controllers](#). Notifications could be sent directly to the relevant [service endpoints](#) listed in an existing [DID](#).
2. [Self-monitoring](#). A [DID subject](#) can employ their own local or online agent to periodically monitor for changes to a [DID document](#).
3. [Third-party monitoring](#). A [DID subject](#) could rely on a third party monitoring service, however this introduces another vector of attack.

9.7 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 other client applications need to validate that keys were not expired at the time they were used. Since some use cases may have legitimate reasons why already-expired keys can be extended, a key expiration shouldn't prevent any further use of the key, and implementations of a resolver ought to be compatible with such extension behavior.


9.8 Key Revocation and Recovery §


Section § 7.2 [DID Operations](#) specifies the [DID](#) operations that must be supported by a [DID method](#) specification, including deactivation of a [DID document](#) by replacing it with an

updated [DID document](#). In general, checking for key revocation on [DLT](#)-based methods is expected to be handled in a manner similar to checking the balance of a cryptocurrency account on a [distributed ledger](#): if the balance is empty, the entire [DID](#) is deactivated. [DID method](#) specifications are 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 [§ 5.5 Authorization and Delegation](#). 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 (see Section [§ 11.4 Time Locks and DID Document Recovery](#)).

9.9 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 known as [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 — 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 out-of-scope for 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: (a) the numerous security attacks based on [deceiving users](#) about the true human-friendly identifier for a target entity, and (b) the [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 e-mail addresses using DNS lookups is available at [\[DNS-DID\]](#).

9.10 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 spec is focused on a data model rather than a protocol, it offers no opinion about many aspects of how that model is put to use. However, individual [DID methods](#) may wish to consider constraints that would eliminate behaviors or semantics they don't 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](#) once it's been 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 spec to be immutable (e.g., 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 may provide 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 client 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, may 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. 

10. Privacy Considerations §

This section is non-normative.

It is critically important to apply the principles of Privacy by Design to all aspects of 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, 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 bear in mind.

10.1 Requirements of DID Method Specifications §

1. DID method specifications *MUST* include their own Privacy Considerations sections, if only to point to the general privacy considerations in this section.
2. 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.

10.2 Keep Personally-Identifiable Information (PII) Private §

If a DID method specification is written for a public DID registry where all DIDs and DID documents will be publicly available, it is *critical* that DID documents contain no PII. All PII should be kept behind service endpoints under the control of the DID subject. With this privacy architecture, PII may 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 relying parties to implement the GDPR right to be forgotten, as no PII will be written to an immutable distributed ledger.

10.3 DID Correlation Risks and Pseudonymous DIDs §

Like any type of globally unique identifier, DIDs may be used for correlation. DID controllers can mitigate this privacy risk by using pairwise unique DIDs, i.e., by 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.

10.4 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 to DIDs to be isolated perfectly into unique buckets, where timing correlation and similar analysis is easy. Therefore, a better strategy for endpoint privacy may be to share an endpoint among thousands or millions of DIDs controlled by many different subjects.

10.5 Herd Privacy §

When a DID subject is indistinguishable from others in the herd, privacy is available. When the act of engaging privately with another party is by itself a recognizable flag, privacy is greatly diminished. DIDs and DID methods need to work to improve herd privacy, particularly for those who legitimately need it most. Choose technologies and human interfaces that default to preserving anonymity and pseudonymity in order to reduce digital fingerprints, share common settings across client implementations, keep negotiated options to a minimum on wire protocols, use encrypted transport layers, and pad messages to standard lengths.

11. Future Work §

This section is non-normative.

11.1 Upper Limits on DID Character Length §

The current specification does not take a position on maximum length of a DID. The maximum interoperable URL length is currently about 2K characters. QR codes can handle about 4K characters. Clients using DIDs will be responsible for storing many DIDs, and some methods would be able to externalize some of their costs onto clients by relying on more complicated signature schemes or by adding state into DIDs intended for temporary use. A future version of this specification should set reasonable limits on DID character length to minimize externalities.

11.2 Equivalence §

Including an equivalence property, such as equivID, in DID documents whose value is an array of DIDs would allow subjects to assert two or more DIDs that represent the same subject. This capability has numerous uses, including supporting migration between DID registries and providing forward compatibility of existing DIDs to future DID registries. In theory, equivalent DIDs should have the same identifier rights, allowing verifiable claims made against one DID to apply to equivalent DIDs. Equivalence was not included in the current specification due to the complexity of verifying equivalence across different DLTs and different DID methods, and also of aggregating properties of equivalent DID documents. However equivalence should be supported in a future version of this specification.

11.3 Timestamps §

Verifiable timestamps have significant utility for identifier records. This is a good fit for DLTs, since most offer some type of timestamp mechanism. Despite some transactional cost, they are the most censorship-resistant transaction ordering systems in the world, so they are nearly ideal for DID document timestamping. In some cases a DLT's immediate timing is approximate, however their sense of "median time past" (see Bitcoin BIP 113) can be precisely defined. A generic DID document timestamping mechanism could work across all DLTs and might operate via a mechanism including either individual transactions

or transaction batches. The generic mechanism was deemed out of scope for this version, although it may be included in a future version of this specification.

11.4 Time Locks and DID Document Recovery §

Section § 9.8 [Key Revocation and Recovery](#) mentions one possible clever use of time locks to recover control of a [DID](#) after a key compromise. The technique relies on an ability to override the most recent update to a [DID document](#) with Authorization applied by an earlier version of the [DID document](#) in order to defeat the attacker. This protection depends on adding a [time lock \(see Bitcoin BIP 65\)](#) to protect part of the transaction chain, enabling a Authorization block to be used to recover control. We plan to add support for time locks in a future version of this specification.

11.5 Smart Signatures §

Not all [DLTs](#) can support the Authorization logic in section § 5.5 [Authorization and Delegation](#). Therefore, in this version of the specification, all Authorization logic is delegated to [DID method](#) specifications. A potential future solution is a [Smart Signature](#) specification that specifies the code any conformant [DLT](#) may implement to process signature control logic.

11.6 Verifiable Claims §



Although [DIDs](#) and [DID documents](#) form a foundation for decentralized identity, they are only the first step in describing their subjects. The rest of the descriptive power comes through collecting and selectively using [verifiable claims](#). Future versions of the specification will describe in more detail how [DIDs](#) and [DID document](#) can be integrated with — and help enable — the verifiable claims ecosystem.

11.7 Alternate Serializations and Graph Models §

This version of the specification relies on JSON-LD and the RDF graph model for expressing a [DID document](#). Future versions of this specification might specify other semantic graph formats for a [DID document](#).

A. Current Issues §

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

ISSUE 85: Syntactically differentiate data about the DID versus application data

discuss high priority

Syntactically differentiate data about the DID versus application data

ISSUE 84

Add **initial-values** matrix parameter to Generic DID Parameters

ISSUE 80: Define conformance classes such as "DID document processor" editorial

Define conformance classes such as "DID document processor"

ISSUE 76: Bikeshed the DID specification short name discuss pending close

Bikeshed the DID specification short name

ISSUE 75: tracking revocation of public keys discuss

tracking revocation of public keys

ISSUE 72: Privacy Considerations - Specifically call out GDPR discuss

Privacy Considerations - Specifically call out GDPR

ISSUE 70: Enable instant DID use/resolution for DID Methods that include a propagation delay discuss

Enable instant DID use/resolution for DID Methods that include a propagation delay

ISSUE 69: How to integrate certificates with DIDs? discuss

How to integrate certificates with DIDs?

ISSUE 68: When do we publish a FPWD?

When do we publish a FPWD?

[ISSUE 67: Supported public key formats?](#) discuss pr pending

Supported public key formats?

[ISSUE 65: Does DID Document metadata belong in the Document?](#) discuss

Does DID Document metadata belong in the Document?

[ISSUE 64: Encrypted serviceEndpoint values?](#) discuss

Encrypted serviceEndpoint values?

[ISSUE 63](#)

Add publicKeyHex as a valid publicKey format

[ISSUE 62](#)

Add "service-type" DID URL matrix parameter.

[ISSUE 61](#)

Add "content-type" and "content-id" DID URL matrix parameters.

[ISSUE 60](#)

Add "key-type" DID URL matrix parameter.

[ISSUE 59](#)

Add "key" DID URL matrix parameter.

[ISSUE 58: Registry handling](#) discuss

Registry handling

[ISSUE 57: Clarification of other verification methods in authentication section](#) missing discuss

Clarification of other verification methods in authentication section missing

[ISSUE 56](#)

Added support for ethereumAddress in context - fixed #55

[ISSUE 55](#): Add support for ethereumAddress public key type in @context discuss **high priority**

Add support for ethereumAddress public key type in @context

[ISSUE 53](#): Normative vs. non-normative references editorial

Normative vs. non-normative references

[ISSUE 52](#): (Minor note) update the IANA file editorial

(Minor note) update the IANA file

[ISSUE 51](#): [Convention] Method `0` (zero) become a well-known method for retrieving properties/metadata from/about a particular DID Server discuss

[Convention] Method **0** (zero) become a well-known method for retrieving properties/metadata from/about a particular DID Server

[ISSUE 49](#): If an existing DID Document has a Service Endpoint fragment, what are the primary keys to be used if that Service Endpoint needs to be replaced, updated, or deleted? discuss

If an existing DID Document has a Service Endpoint fragment, what are the primary keys to be used if that Service Endpoint needs to be replaced, updated, or deleted?

[ISSUE 48](#): Some comments by Steven Rowat editorial

Some comments by Steven Rowat

[ISSUE 46](#): It would be useful to have `services` as a mapping instead of an `array` discuss



It would be useful to have **services** as a mapping instead of an **array**

[ISSUE 45](#): Is method-specific-id supposed to be equivalent to param-char? editorial

Is method-specific-id supposed to be equivalent to param-char?

B. Registries §

There are multiple registries that define DID methods and extensions to this specification. These registries are:

Registry	Purpose
<u>DID Method Registry</u>	Lists all known <u>DID methods</u> and contains links to their specifications. 
<u>Linked Data Cryptography Suite Registry</u>	Defines all known Linked Data Cryptography Suites and Key Formats. 

C. Real World Example §

A future-facing real-world context is provided below:

EXAMPLE 24: Advanced DID document example

```
{
  "@context": "https://w3id.org/future-method/v1",
  "id": "did:example:123456789abcdefghi",

  "publicKey": [{
    "id": "did:example:123456789abcdefghi#keys-1",
    "type": "RsaVerificationKey2018",
    "controller": "did:example:123456789abcdefghi",
    "publicKeyPem": "-----BEGIN PUBLIC KEY...END PUBLIC KEY-----\r\n"
  }, {
    "id": "did:example:123456789abcdefghi#keys-3",
    "type": "Ieee2410VerificationKey2018",
    "controller": "did:example:123456789abcdefghi",
    "publicKeyPem": "-----BEGIN PUBLIC KEY...END PUBLIC KEY-----\r\n"
  }],

  "authentication": [
    // this mechanism can be used to authenticate as did:...fghi
    "did:example:123456789abcdefghi#keys-1",
    // this mechanism can be used to biometrically authenticate as did:..fghi
    "did:example:123456789abcdefghi#keys-3",
    // this mechanism 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"
    }
  ],

  "service": [{
    "id": "did:example:123456789abcdefghi#oidc",
    "type": "OpenIdConnectVersion1.0Service",
    "serviceEndpoint": "https://openid.example.com/"
  }, {
    "id": "did:example:123456789abcdefghi#vcStore",
```



```
    "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#hub",
    "type": "HubService",
    "serviceEndpoint": "https://hub.example.com/.identity/did:example:01
23456789abcdef/"
  }, {
    "id": "did:example:123456789abcdefghi#messaging",
    "type": "MessagingService",
    "serviceEndpoint": "https://example.com/messages/8377464"
  }, {
    "type": "SocialWebInboxService",
    "id": "did:example:123456789abcdefghi#inbox",
    "serviceEndpoint": "https://social.example.com/83hfh37dj",
    "description": "My public social inbox",
    "spamCost": {
      "amount": "0.50",
      "currency": "USD"
    }
  }, {
    "type": "DidAuthPushModeVersion1",
    "id": "did:example:123456789abcdefghi#push",
    "serviceEndpoint": "http://auth.example.com/did:example:123456789abc
defghi"
  }, {
    "id": "did:example:123456789abcdefghi#bops",
    "type": "BopsService",
    "serviceEndpoint": "https://bops.example.com/enterprise/"
  }]
}
```

D. References §

D.1 Normative 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/>

[HASHLINK]

Cryptographic Hyperlinks. Manu Sporny. IETF. December 2018. Internet-Draft. URL: <https://tools.ietf.org/html/draft-sporny-hashlink-02>

[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/>

[LD-PROOFS]

Linked Data Proofs. Manu Sporny; Dave Longley. Digital Verification Community Group. CG-DRAFT. URL: <https://w3c-dvcg.github.io/ld-proofs/>

[LD-SIGNATURES]

Linked Data Signatures. Manu Sporny; Dave Longley. Digital Verification Community Group. CG-DRAFT. URL: <https://w3c-dvcg.github.io/ld-signatures/>

[LINKED-DATA]

Linked Data Design Issues. Tim Berners-Lee. W3C. 27 July 2006. W3C-Internal Document. URL: <https://www.w3.org/DesignIssues/LinkedData.html>

[MATRIX-URIS]

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

[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>

[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>

[RFC5234]

Augmented BNF for Syntax Specifications: ABNF. D. Crocker, Ed.; P. Overell. IETF. January 2008. Internet Standard. URL: <https://tools.ietf.org/html/rfc5234>

[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>

[RFC8259]

The JavaScript Object Notation (JSON) Data Interchange Format. T. Bray, Ed.. IETF. December 2017. Internet Standard. URL: <https://tools.ietf.org/html/rfc8259>

[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/>

D.2 Informative references §

[DID-METHOD-REGISTRY]

The Decentralized Identifier Method Registry. Manu Sporny; Drummond Reed. Credentials Community Group. CG-DRAFT. URL: <https://w3c-ccg.github.io/did-method-registry/>

[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/>

[IANA-URI-SCHEMES]

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

[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>

[RFC6973]

Privacy Considerations for Internet Protocols. A. Cooper; H. Tschofenig; B. Aboba; J. Peterson; J. Morris; M. Hansen; R. Smith. IETF. July 2013. Informational. URL: <https://tools.ietf.org/html/rfc6973>

[RFC7230]

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

[RFC8141]

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

[VC-DATA-MODEL]

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