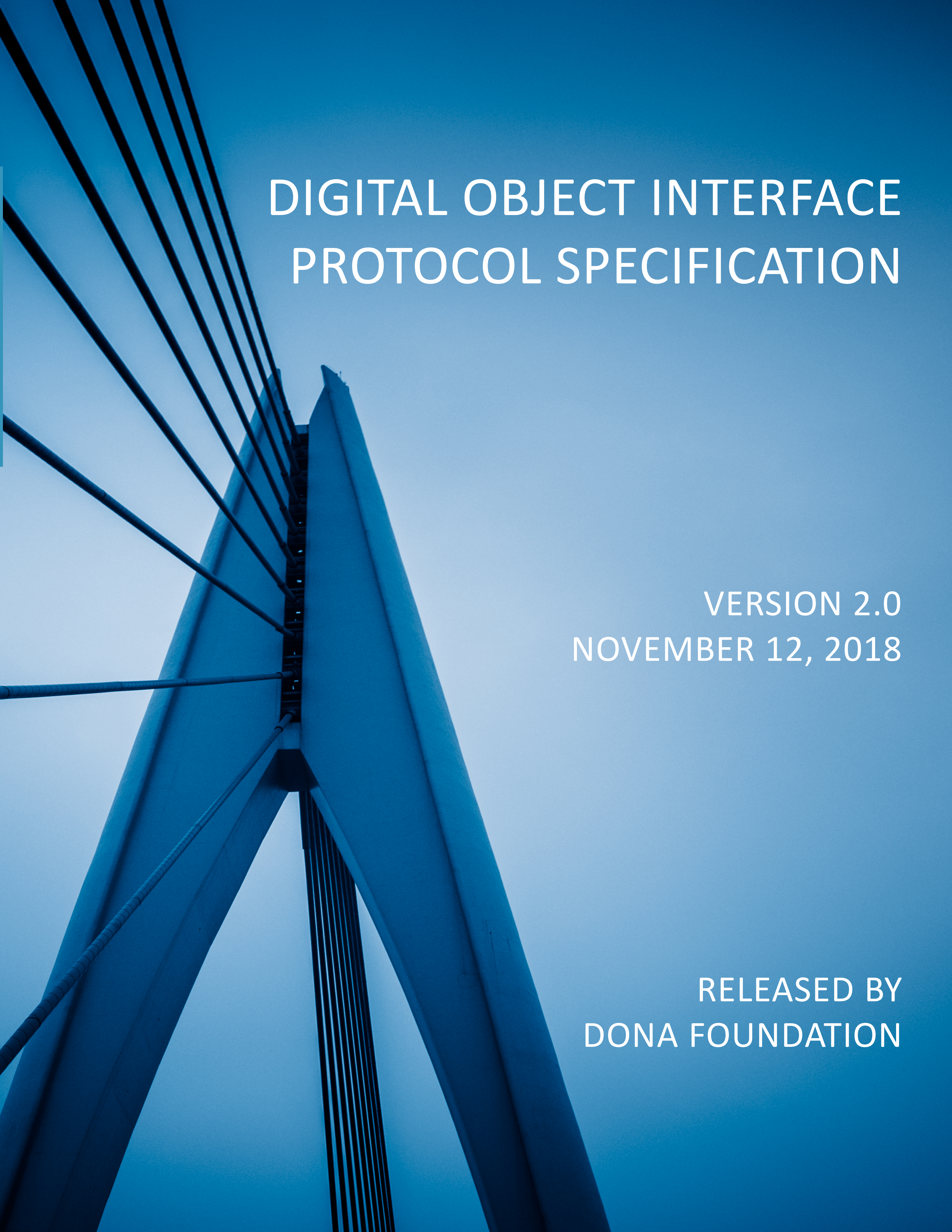

DIGITAL OBJECT INTERFACE PROTOCOL SPECIFICATION

VERSION 2.0
NOVEMBER 12, 2018

RELEASED BY
DONA FOUNDATION

ACKNOWLEDGEMENTS

The DONA Foundation would like to thank the following individuals for their many contributions and insights in the preparation of this DOIP Specification:

Robert E. Kahn, Christophe Blanchi, Laurence Lannom, Patrice A. Lyons, Giridhar Manepalli, Robert Tupelo-Schneck and Sam Sun.

CONTENTS

DIGITAL OBJECT INTERFACE PROTOCOL SPECIFICATION VERSION 2.0

I INTRODUCTION

This document is a specification for the Digital Object Interface Protocol (DOIP), a core protocol of the Digital Object Architecture (DO Architecture; or DOA). The DO Architecture is a logical extension of the Internet architecture that addresses the need to support information management more generally than just conveying information in digital form from one location in the Internet to another. It is a *non-proprietary* architecture and is publicly available without charge. It is an outgrowth of earlier work on mobile programs,[1] and security for packet radio systems.[2] The DOIP is intended to enable interoperability across heterogeneous information systems.

2 DIGITAL OBJECT ARCHITECTURE

The DO Architecture introduces the concept of a digital object, which forms the basis for the architecture.[3] A digital object (DO) is a sequence of bits, or a set of sequences of bits, incorporating a work or portion of a work or other information in which a party has rights or interests, or in which there is value, each of the sequences being structured in a way that is interpretable by one or more computational facilities.[4] Each DO has, as an essential element, an associated unique persistent identifier, known as digital object identifier (referred to informally as a handle).

[1] R.E.Kahn and V.Cerf, "The Digital Library Project (Volume 1): The World of Knowbots [Draft]," CNRI (1988), *at* http://www.cnri.reston.va.us/kahn-cerf-88.pdf

[2] R.E.Kahn, "The organization of computer resources into a packet radio network," *Managing Requirements Knowledge, International Workshop* (1975), *at* https://www.computer.org/csdl/proceedings/afips/1975/5083/00/50830177.pdf

[3] R.E. Kahn and R. Wilensky, "A Framework for Distributed Digital Object Services," *International Journal on Digital Libraries* (2006), *at* https://www.doi.org/topics/2006_05_02_Kahn_Framework.pdf

[4] *See* "System for uniquely and persistently identifying, managing and tracking digital objects," U.S. Patent No. 6,135,646 (now expired).

For all practical purposes, the concept of a digital object is substantially similar t o the notion of "digital entity" as defined in ITU-T Recommendation X.1255[5] that is based largely on the Digital Object Architecture. The ITU-T Recommendation is available in other languages. An "entity" in that recommendation is defined as anything that can be separately and uniquely identified. It also describes a "digital entity" (DE) as an entity that is represented as, or converted to, a machine-independent data structure consisting of one or more elements that may be parsed by different information systems. In this specification, the terms digital object and digital entity are used interchangeably. A detailed description of DOs, the DO Data Model, DO interface protocol, and federated registries are presented in X.1255.

The DO Architecture specifies two core protocols and three basic components. As described briefly below, the three components are the identifier/resolution system, the repository system, and the registry system. In practice, the repository and registry components are modular and may be combined, as needed.

The first protocol, the Identifier/Resolution Protocol (IRP), also known in an earlier version as the Handle System Protocol, is used for creating, updating, deleting, and resolving digital object identifiers. As specified in the IRP, each identifier is associated with an identifier record containing relevant "state information" that clients can resolve to; and all identifiers are of the form prefix/suffix where, by default, the prefix may first be resolved to locate the specific identifier/ resolution service to be used and the suffix may be any bit sequence. An organization may run a resolution system for its own set of identifiers by having a prefix allotted to it, and any existing identifier may be converted to a digital object identifier by treating it as a suffix and prepending its allotted prefix. A system implementation based on an earlier version of the protocol was described in three RFCs; the document specifying the Identifier/Resolution Protocol will be available shortly.

The second protocol, the Digital Object Interface Protocol (DOIP), is defined for use by digital object services more generally, of which the repository and registry systems are specific instances. Digital object services are intended to implement the DOIP and its basic required features, as specified in this document. An earlier version of this protocol, based on the

[5] ITU-T Recommendation X.1255, "Framework for discovery of identity management information," approved on September 4, 2013, *at* http://handle.itu.int/11.1002/1000/11951

Repository Access Protocol (RAP) originally described by R.E. Kahn and R. Wilensky,[6] was made publicly available in 2009.[7]

THE IDENTIFIER/RESOLUTION SYSTEM

The identifier/resolution system is one of the three components comprising the Digital Object Architecture. This system enables several digital object services, including:

- allotment of unique identifiers to information in digital form structured as digital objects regardless of the location of such information or the technology used to serve such information;
- rapid resolution of the identifiers to current state information about the corresponding digital object, e.g., its location(s), access & usage policies, timestamps, and/or public keys; and
- administration of the identifier records that contain the state information.

THE REPOSITORY SYSTEM

The repository system is a digital object service that provides the necessary functionality to manage digital objects including the provision of access to such objects based on the use of identifiers, and with integrated security. Through the use of identifiers in the access protocol, the repository system abstracts away the details of the storage technologies from the clients, thus enabling a long-lived mechanism for depositing and accessing digital objects. Access to this system is enabled using the DOIP described below.

THE REGISTRY SYSTEM

The registry system is a specialized repository system intended to store metadata about digital objects rather than the digital information itself, and when used as a standalone component, typically stores metadata of digital objects that are managed by one or more repository systems. Access to this system is enabled using the DOIP as well.

[6] *Supra*, note 3, at section 3.1.

[7] Digital Object Protocol Specification, version 1.0, CNRI (November 12, 2009), *at* http://dorepository.org/documentation/Protocol_Specification.pdf

3 DIGITAL OBJECT INTERFACE PROTOCOL

The Digital Object Interface Protocol (DOIP) ver. 2.0 specifies a standard way for clients to interact with digital objects (DOs). It is assumed that such DOs are managed by DO Services, which we often refer to as DOIP services in this document, and that the protocol implementation is part of those services. In this context, a DOIP service itself is considered a digital object. By its very nature, a protocol is intended for enabling interaction between one or more other entities running the protocol and thus, in general, to support a specific form of process-to-process interaction in a network environment.

The DOIP makes use of the IRP for associating identifiers with different elements of the protocol. The maximum size of an identifier will vary over time, but, initially, the maximum size of identifiers as specified in the DOIP is 4096 bits.

The DOIP enables the provision of security using PKI[8] to validate digital objects, including for service/client authentication as well as for ensuring integrity via signatures. The inherent PKI support will also help clients and services leverage encryption. Access control of DOs using identifiers to designate an approved access control list and a PKI challenge response test is assumed by the protocol. Basic operations that clients may invoke on the services are defined; and the protocol inherently supports the addition of operations.

DOIP can be tunneled through any secure communications protocol and the DOIP itself can be used to determine the choice of such protocol. A minimum requirement is that TLS[9] be supported for network communications. In addition to transport security, several other specifications are also leveraged by the protocol: one is the means by which serialization is achieved, for which JSON[10] can be used. The rest of this document assumes that JSON is used for serialization unless otherwise stated. A second, as indicated above, is the use of PKI for encryption and decryption of DOs including authentication of other system resources; but this capability, although relying on techniques external to the protocol, is enabled with the use of

[8] PKI-Public Key Infrastructure, *see* https://www.ssh.com/pki
[9] T.Dierks and C.Allen, "The Transport Layer Security (TLS) Protocol," RFC 2246 (1999), *at* https://www.ietf.org/rfc/rfc2246.txt; T.Dierks and E.Rescorla, TLS, RFCs 4346 & 5246 (2006 & 2008), *at* https://www.ietf.org/rfc/rfc4346.txt and https://tools.ietf.org/html/rfc5246
[10] T.Bray, Ed., "The JavaScript Object Notation Format," RFC 8259 (Dec. 2017), *at* https://tools.ietf.org/html/rfc8259

digital object identifiers. Other external specifications include Unicode[11] (specifically UTF-8 encoding[12]), TCP,[13] MIME,[14] X509,[15] JWS[16] and JWK.[17]

Each DO must specify its type. Core types are defined for this purpose; and types are extensible to allow for the creation of new types. One important function of types is to enable a DOIP service to identify allowable operations. Types are allotted identifiers, and each type is therefore associated with an identifier record that can be accessed by use of the IRP. The semantic and other structuring specifics of type records are not specified in the DOIP. It is assumed that groups or organizations with domain expertise will take responsibility for creating types in their domain and for specifying the semantic and serialization of type records.

The various DOIP-related constructs and interactions are defined next.

4 SERIALIZATION OF DIGITAL OBJECTS

A digital object (DO) as communicated between digital object services and clients *must* conform to the agreed form of serialization. Client software that incorporates DOIP software may be part of another DOIP service. The minimum required serialization of a DO is specified in Appendix A. At a high-level, a DO consists of an identifier, a type, optional and open-ended attributes, plus optional elements. The identifier of the DO *must* be unique and resolvable as specified in the IRP.

Clients interact with any DOs by establishing DOIP connections to each DO's respective DO Service. To do so, clients will need to acquire that DO Service's information to establish a network connection to it. This information is called the DO Service Information. The specific

[11] The Unicode Standard, http://www.unicode.org/standard/standard.html
[12] F.Yergeau, "UTF-8, a transformation format of ISO 10646," RFC 2279 (1998), *at* https://www.ietf.org/rfc/rfc2279.txt
[13] M.Duke, R.Braden, W.Eddy, E.Blanton, and A.Zimmermann, "A Roadmap for Transmission Control Protocol (TCP) Specification Documents," RFC 7414 (2015), *at* http://tools.ietf.org/html/rfc7414
[14] N.Freed and N.Borenstein, "Multipurpose Internet Mail Extensions (MIME)," RFCs 2045 & 2046 (1996), *at* https://tools.ietf.org/html/rfc2045 and https://tools.ietf.org/html/rfc2046; K.Moore, MIME, RFC 2047 (1996), *at* https://www.ietf.org/rfc/rfc2047.txt; N.Freed and J.Klensin, MIME, "Media Type Specifications and Registration Procedures," IETF, RFCs 4288 & 4289 (2005), *at* https://tools.ietf.org/html/rfc4288 and https://tools.ietf.org/html/rfc4289
[15] D.Cooper, S.Santesson, S.Farrell, S.Boeyem, R. Housley and W.Polk, "Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL) Profile," RFC 5280 (2008), *at* https://tools.ietf.org/html/rfc5280
[16] M.Jones, J.Bradley and N.Sakimura, "JSON Web Signature (JWS)," RFC 7515 (2015), *at* https://tools.ietf.org/html/rfc7515; M.Jones, JWS, "Unencoded Payload Option," RFC 7787 (2016), *at* https://tools.ietf.org/html/rfc7797
[17] *Supra,* note 16, RFC 7515.

values encoded in the Service Information are described in the types section of this document as the 0.TYPE/DOIPServiceInfo type.

The client *shall* resolve the DO identifier using the IRP. The resulting information shall contain either the Service Information associated with the 0.TYPE/DOIPServiceInfo type or receive a redirection to another identifier. In the case the client receives a redirection, it will resolve the new identifier using the IRP and any additional redirection into a Service Information record.

Clients *may* cache any Service Information for expediting future interactions with the DOIP service. Details of the identifier records are stated in the discussion on Core Types in Appendix D of this specification.

A DOIP service, acting as a proxy, *may* allow operations to be invoked on DOs managed by other DOIP services. The proxy service invokes the client-specified operation on the service that manages the identified DO and responds back the results to the client. The proxy service *may* also cache the response for expeditiously responding to such future requests.

The following section specifies how clients communicate with DOIP services for purposes of invoking operations on DOs managed by such services.

5 OPERATIONS

The DOIP operations are categorized as Basic and Extended. Basic operations *must* be properly interpreted by every DOIP service and shall be built into those services a priori. Extended operations *may* be implemented by DOIP services as they choose, provided that adequate security is enforced in retrieving, validating and executing such operations.

All DOIP operations, whether basic or extended, *must* have unique resolvable identifiers as specified in the IRP.

5.1 BASIC OPERATIONS

Every DOIP service *must* properly interpret the following Basic Operations if properly communicated by the client: hello, create, access, update, delete, search, and listOperations. The service *may* deny or allow a client to perform such an operation depending on the service's internal policies including those that pertain to access control. See Appendix B for details of the Basic Operations.

5.2 EXTENDED OPERATIONS

The DOIP services *may* support operations beyond the basic ones; and identifiers of such operations shall be resolvable as specified in the IRP. Those operations are not part of the basic DOIP specification, but the manner in which they are carried out is no different from those of the basic operations.

Extended operation specific functionality may be built into the service implementation, if desired. Alternatively, a DOIP service *may* provide runtime environments that retrieve, validate, and execute code managed in special DOs that pertain to extended operations. In either case, the operation details shall be managed as a DO as discussed in Appendix C. The purpose of representing the operation as a DO is to disseminate information about how to invoke the extended operation.

Extended DO operations can be developed to add specific ways to access digital information or to leverage different security mechanisms such as encryption, role based access control or proof of work techniques.

6 TYPES

Types are represented as unique identifiers that are resolvable to a "type record" as specified in the IRP and used for categorizing DOs that clients may encounter when interacting with DOIP services. In particular, a DO that is returned by a DOIP service *must* include a type for the DO as a whole. The type informs DOIP services what operations *can* be invoked against the DO. Core types of the DOIP are stated in Appendix D.

Types *may* be extended from Core Types by DOIP service implementations. There is no specific structure stipulated for an extension of Core Types other than to specify the parent type. Implementations may include added structure, however.

This linking between types enables clients that are aware of a few types to be able to process DOs categorized under more refined types, and vice versa. All extended types *must* indicate a reference to the type from which it is extended. In particular, an extended type using its identifier record *shall* indicate the parent type in a handle value: specifically, the parent type shall be specified in a data field that corresponds to the (identifier value) type "0.TYPE/Type".

Types themselves *may* be managed as DOs to provide additional information that clients may use to process those DOs. This specification does not prescribe the nature and structure of such additional information, and leaves it for implementations.

7 COMMUNICATION PROTOCOL

7.1 COMMUNICATION SECURITY

DOIP network communications *should* take place over the secure transport protocol connection specified in the DO Service Information, such as TLS, for instance, or by use of such other communication/security protocol as the parties may agree. Basic Operations carried out via a network shall take place over the default secure transport protocol, while Extended Operations may take place over the default protocol or over other security protocols as the participants may agree, or as the DOs may otherwise require. In the case where TLS is used, during a TLS handshake, the server *must* present a certificate as required by TLS; this certificate *must* contain the identifier of the DOIP service as well as the public key of the DOIP service. A client *shall* check that the public key corresponds to the expected public key by resolving the identifier using the IRP. The TLS connection itself validates that the server has the corresponding private key.

The client *may* provide a certificate in the same manner to authenticate to the DOIP service; however, this is not required when requests are made anonymously or for use of Extended Operations. In particular, for Extended Operations, the client may authenticate using a different mechanism that supports an alternative method for authentication. The optional property “authentication” shall be used in such cases. In particular, whenever a client certificate is provided, the identifier in the client certificate *must* match the ClientId stated in the DOIP request. The ClientId should be left blank for making requests anonymously, although not all anonymous requests will necessarily be fulfilled.
TLS certificates are based on the X.509 specification,[18] which defines a number of terms, including the ones underlined in the two bullets shown below. The DOIP service certificate as well as the client certificate shall be interpreted as follows (using terms defined in the X.509 specification):

[18] S*upra,* note 15.

- The identifier is in the subject field, as the first entry of attribute UID, if present, otherwise as the first entry of attribute CN.
- The public key is the subject public key.

There is no restriction on who can issue a certificate. If desired, a certificate *may* be self-signed. Clients and DOIP services *shall* validate a certificate by resolving the identifier in the certificate to retrieve the public key and match the key against the one stated in the certificate. The identifier will be interpreted as the DOIP service identifier in the case of the server certificate and must match the DOIP service identifier the client intended to contact; the identifier will be interpreted as the client identifier in the case of a client certificate, and must match the ClientId stated in the DOIP request.

Clients and DOIP services *may* use other additional criteria (as jointly agreed) for validating the other party; for example, the prefix of the identifier presented in the certificate may be used to indicate the certificate issuer.

7.2 REQUEST/RESPONSE INTERACTIONS

A client *may* send multiple DOIP requests in sequence over a single connection, once the TLS connection (or such other connection as agreed) is established, and the service public key is validated by a client and, optionally, vice versa. The DOIP service *shall* respond to each such request and such responses *shall* include the requestId to which it corresponds.

A DOIP request or response consists of multiple *segments* as defined here. Each segment is either:

- A JSON segment, which contains UTF-8 encoded text in the JSON data interchange format, generally over multiple lines, terminated by a newline character; or
- a bytes segment, which contains arbitrary bytes. A bytes segment must begin with a single line of text starting with the at-sign character ("@") optionally followed by whitespace terminated by a newline; this line is followed by the bytes in a chunked encoding, which consists of zero or more chunks. Each chunk has:
    - a line of text starting with the UTF-8 representation of a positive decimal number (the size of the chunk excluding the delimiters), optionally followed by whitespace, and terminated by a newline character;
    - as many bytes as indicated by the size, optionally followed by whitespace, and followed by a newline character; or

- an empty segment, which indicates the end of the request or response.

At the end of each segment must be a single line of UTF-8 text starting with the hash sign (“#”) optionally followed by whitespace and terminated by a newline character. A JSON segment is terminated by the first hash sign immediately following a newline. A bytes segment is terminated by the first hash sign that appears instead of a chunk size. An empty segment terminates the request or response.

A DOIP request or response can be parsed by reading a sequence of segments. If a segment starts with the character "@", it is a bytes segment; if a segment starts with the character "#", it is an empty segment indicating the end; otherwise, the segment is the default segment (initially a JSON segment) or such other segment as shall be agreed between the parties.

A DOIP request or response *must* lead with a JSON segment.

7.2.1 REQUEST

A valid DOIP request is produced as a JSON segment with the following properties:

a. requestId: the identifier of the request provided by the client; shall be unique within a given DOIP session so clients can distinguish between DOIP service responses. The requestId shall be a string not exceeding 4096 bits.
b. clientId: the identifier of the client.
c. targetId: the identifier of the DO on which the operation is to be invoked; the DOIP service could itself be the target.
d. operationId: the identifier of the operation to be performed.
e. attributes: optional array of JSON properties; operation stipulated.
f. authentication: optional JSON object used by clients to authenticate.
g. input: arbitrary information handled by the operation. If absent, the input for the operation is all segments that follow this segment. If present, the input is the JSON object corresponding to the "input" property, and there must be no further non-empty segments in the request.

Appendix B defines specific values for the above properties for the various basic operations.

7.2.2 RESPONSE

A valid DOIP response is produced as a JSON segment with the following properties:

a. requestId: the identifier of the request to which this is a response. The DOIP service must include in its response the requestId provided by the client.
b. status: an identifier that indicates the status of the request. Must be from the codes listed in the Status Codes subsection below.
c. attributes: optional array of JSON properties; operation stipulated.
d. output: arbitrary information returned to the client, depending on the operation. If absent, the output for the operation is all segments that follow this response object. If present the output is the JSON object corresponding to the output property, and there must be no further non-empty segments in the request.

Appendix B defines specific values for the above properties for the various basic operations.

The terms JSON object and JSON property conform to the JSON specification. The term JSON segment is as defined in the Exchanges subsection above.

7.3 STATUS CODES

Status codes *shall* have associated unique identifiers resolvable as specified in the IRP. The following basic status codes are applicable. Additional status codes may be used by implementations and be supplied within attributes, but a basic code *must* be supplied in the status property of any DOIP response.

- 0.DOIP/Status.001: The operation was successfully processed.
- 0.DOIP/Status.101: The request was invalid in some way.
- 0.DOIP/Status.102: The client did not successfully authenticate.
- 0.DOIP/Status.103: The client successfully authenticated, but is unauthorized to invoke the operation.
- 0.DOIP/Status.104: The digital object is not known to the service to exist.
- 0.DOIP/Status.105: The client tried to create a new digital object with an identifier already in use by an existing digital object.
- 0.DOIP/Status.200: The service declines to execute the extended operation.
- 0.DOIP/Status.500: Error other than the ones stated above occurred.

8 IDENTIFIERS

As described above, the DOIP defines four forms of identifiers: one for operations, one for types, one for status information and one for DOs. Implementations can define their own set of identifiers, as appropriate, as long as they are resolvable as specified in the IRP. That said, whenever DOIP is used in specific environments where external resolutions of identifiers become unnecessary, either because the anticipated clients are already aware of the information in the identifier records, or the exposure of such information in the identifier records poses security risks or other concerns, such identifiers may not be resolvable via the IRP. It is to be noted, however, that when DOIP Services are made available in the Internet, it is anticipated that such identifiers shall resolve to records that contain minimum information as specified in this document.

The convention used for the basic set is to use the prefix 0.DOIP for operations as well as status information, and the prefix 0.TYPE for types. In this version of the protocol, the identifiers have semantics; in subsequent versions, it is intended that such identifiers will also have non-semantic representations as well.

APPENDIX A: DO SERIALIZATION

Any DO communicated via DOIP must be serialized using the DOIP Serialization. The order of serialization of the various fields as defined next are not important. The first segment described below is the JSON serialization of the digital object except for the *data* portion within elements. The data portions follow the first segment, with each data portion serialized as two segments: a segment in JSON serialization expressing an object with a property "id" which indicates the id of the element, followed by a bytes segment consisting of the bytes of the data from the element. A final JSON segment contains signatures as described in Appendix E. The signatures segment is required for DOs of type 0.TYPE/DOIPServiceInfo and 0.TYPE/DOIPOperation, and is otherwise optional.

Unless otherwise stated, all fields are required. All values other than the data portion *must* be encoded in UTF-8.

- ***id***: the identifier of the DO.
- ***type***: the DO type. *Must* be 0.TYPE/DO or its extension. See Types section.
- ***attributes (optional)***: one or more fields (key-value pairs) serialized as a JSON (sub) object.
- ***elements (optional)***: one or more elements serialized as an array in the default serialization, with each element consisting of:
  - ***id***: identifier of the element; *must* be unique within a DO.
  - ***length (optional)***: length of the data portion.
  - ***type***: *shall* be a type as defined in this spec or a MIME type.
  - ***attributes (optional)*****:** one or more fields serialized as an object using the default serialization, or as a JSON (sub) object.
  - ***data***: serialized as separate segments, as stated above.
- ***signatures (conditional)***: Required for DOs of type 0.TYPE/DOIPServiceInfo and 0.TYPE/DOIPOperation; otherwise optional. The field is an array of strings in the default serialization; each string is in JWS format[19] with an unencoded detached payload. The header of each JWS must specify the signing entity by referencing an identifier in the "kid" field which can be resolved to obtain the signer's public key. The content of the payload is the serialized digital object, omitting the signatures and canonicalized as described in Appendix E.

[19] *Supra,* note 16, RFC 7515.

APPENDIX B: BASIC OPERATIONS

The basic DOIP operations are listed below. The input and output of successful responses are described. Failure responses, in addition to the non-successful "status", *may* have as output a JSON object with property "message" with a human-readable description of the error.

- **0.DOIP/Op.Hello**: An operation to allow a client to get information about the DOIP service.
  - Request attributes: none
  - Input: empty
  - Response attributes: none
  - Output: the default serialization of the DOIP Service Information as a DO.
- **0.DOIP/Op.Create**: An operation to create a digital object within the DOIP service. The target of a creation operation is the DOIP service itself.
  - Request attributes: none
  - Input: a serialized digital object. The default serialization may be used if the object lacks element data; otherwise the serialization is a multi-segment DO serialization as described above. The "id" can be omitted to ask the DOIP service to automatically choose the id.
  - Response attributes: none
  - Output: the default serialization of the created object omitting element data. Notably, includes the identifier of the object (even if chosen by the client) and any changes to the object automatically performed by the DOIP service.
- **0.DOIP/Op.Retrieve**: An operation to retrieve (some parts of the) information represented by the target DO.
  - Request attributes:
  - "element": if specified, retrieves the data for that element
  - "includeElementData": if present, returns the serialization of the DO including all element data
  - Input: none
  - Response attributes: none
  - Output: the default output is the serialization of the object using the default serialization without element data. If "element" was specified, the output is a single bytes segment with the bytes of the specified element. If "includeElementData" was specified, the output is the full serialized DO.

- **0.DOIP/Op.Update**: An operation to update (some parts of the) information represented by the target DO.
  - Request attributes: none
  - Input: a serialized digital object. The default serialization may be used if the object lacks element data (or if no element data is to be changed); otherwise the serialization is a multi-segment DO serialization as described above. Elements which are not intended to be changed can be omitted from the input.
  - Response attributes: none
  - Output: the default serialization of the created object omitting element data. Notably, includes any changes to the object automatically performed by the DOIP service.
- **0.DOIP/Op.Delete**: An operation to remove the target DO from the management of the DOIP service.
  - Request attributes: none
  - Input: none
  - Response attributes: none
  - Output: none
- **0.DOIP/Op.Search**: An operation to discover digital objects by searching metadata contained in the set of digital objects managed by the DOIP service.
  - Request attributes:
  - "query": the search query to be performed, in a textual representation
  - "pageNum": the page number to be returned, starting with 0
  - "pageSize": the page size to be returned; if missing or negative, all results will be returned; if zero, no results are returned, but the "size" is still returned
  - "sortFields": a comma-separated list of sort specifications, each of which is a field name optionally followed by ASC or DESC
  - "type": either "id", to return just object ids, or "full", to return full object data (omitting element data); defaults to "full"
  - Input: none
  - Response attributes: none
  - Output: an object based on the default serialization with top-level properties:
  - "size": the number of results across all pages
  - "results": a list of results, each of which is either a string (the object id) or the default serialization of an object omitting element data.
- **0.DOIP/Op.ListOperations**: An operation to request the list of operations that can be invoked on the target DO.

- Request attributes: none
- Input: none
- Response attributes: none
- Output: a serialized list of strings based on the default serialization, each of which is an operation id that the target DO supports.

The following identifiers designate parameters that are useful and/or necessary for DOIP Operations.

- 0.DOIP/Request: This identifier shall be used to describe the specifics of the DOIP request for extended operation.
- 0.DOIP/Response: This identifier shall be used to describe the specifics of the DOIP response for extended operation.
- 0.DOIP/OperationReference: This identifier shall be used to designate one DOIP operation being similar to another DOIP operation.
- 0.DOIP/Transport: This identifier may be used to specify the DOIP transport protocol used by the DO Service; it resolves to an extended DOIP type. If no transport is specified, then TCP/IP is assumed. If the DOIP uses TLS for instance, it may also be specified in this field.
- O.DOIP/Encoding: This identifier may be used to provide information that is used by the DOIP Service to specify the encoding used by the DOIP; it resolves to an extended DOIP type.
- 0.DOIP/AccessControl: This identifier may be used to provide information that specifies the access control operation.

APPENDIX C: DIGITAL OBJECT FOR EXTENDED OPERATION

The minimum structure of an extended operation DO is as follows:

- ***id***: the identifier of the extended operation.
- ***type***: *must* be 0.TYPE/DOIPOperation
- ***attributes***: an array of values containing the following properties.
- ***serviceId***: the identifier of the service where this DO is managed.
    - ***0.DOIP/Request***: One or more key-value pairs used for describing the expected values for each of the properties listed in the Request Section 6.2.1 of this specification. One key *must* be 'human-readable' to suggest that the description of the DOIP request is useful for humans. Other forms of descriptions that simply automation may additionally be used.
    - ***0.DOIP/Response***: One or more key-value pairs used for describing the expected values for each of the properties listed in the Response Section 6.2.2 of this specification. One key *must* be 'human-readable' for reasons stated above.
    - ***0.DOIP/OperationReference***: An optional field to reference another operation identifier to establish similarity of operation implementations.
- ***signatures***: As specified in Appendix A.

APPENDIX D: CORE TYPES

Types, in this context, are intended for DOIP services and related clients to learn of operations that are appropriate to be invoked against a DO. In particular, each DO specifies its type, and that type shall inform DOIP services what operations to perform. Clients shall learn of those applicable operations from DOIP services. Some of these types are intended to be available to all DOIP services, while others may not. In particular, types that are part of encrypted digital information would not be available for general use by DOIP services in the clear.

Types are associated with unique identifiers that are resolvable as specified in the IRP. The identifier record associated with any type, at a minimum, will specify its parent type. In particular, a type indicates in its identifier record, using one value whose data field associated with 0.TYPE/Type shall be the type of its parent.

Core types necessary for DOIP operations are defined here. 0.TYPE/Type is the root of the types. All other core types extend from 0.TYPE/Type. Types that DOIP implementations may create shall extend either from 0.TYPE/Type or its extensions as defined here.

Core types including their intrinsic relationships (presented as a hierarchy) are defined below:

- 0.TYPE/Type: the root of the types.
    - 0.TYPE/DO: the type that shall be used by DOs. DOIP services may use types extended from this type to convey DO specializations.
        - 0.TYPE/DOIPServiceInfo: the type that shall be used to convey the DO service information.
        - 0.TYPE/DOIPOperation: the type that shall be used to designate that a DO represents an extended operation.

In addition to specifying the parent, a type may stipulate other characteristics to be represented in the identifier record. The description below denotes wherever such stipulations apply for core types:

- 0.TYPE/DO: This is a generalized DO type. DOs that correspond to this type shall include, in their identifier record, one value whose data field associated with "0.TYPE/DOIPServiceInfo" shall either be the service identifier of the DOIP service that manages the DO in question or the actual service information for DOs that service themselves, as defined next.

- 0.TYPE/DOIPServiceInfo: This is an extended DO type. DOs that correspond to this type shall conform to the 0.TYPE/DO stipulation stated above, plus include, in their handle record, one handle value whose data field associated with “0.TYPE/DOIPServiceInfo” is the service information record, which is a DO conforming to the serialization specified in Appendix A, and contains the following attributes:
  - serviceName (optional): the name of the DOIP service.
  - serviceDescription (optional): the description of the service.
  - ipAddress (required): the IP address of the service.
  - port (required): the listening port on the IP address.
  - protocol (required): the default shall be ‘TCP’.
  - protocolVersion (required): highest version of the DOIP protocol supported.
  - publicKey (required): the public key expressed in JWK format as a default.
  - any number of other fields (optional)

The signature included in this DO shall be signed using the privateKey of the DOIP service.

Note that a DO returned by the DOIP service during a retrieve operation if invoked on the DO that corresponds to the DOIP service shall conform to the above specification.

- 0.TYPE/DOIPOperation: This is an extended DO type and is applicable for extended operations that provides the code which may be executed by runtime environments offered by DOIP services. DOs corresponding to this type shall conform to Appendix C.

APPENDIX E: DO SIGNATURES

As described in Appendix A, a serialized DO includes JSON segments and, as applicable, bytes segments. The serialized DO may also include yet another JSON segment with the signature of the (rest of the) DO. This Appendix describes how to generate that signature segment and how to verify the signature.

The signature segment shall be a JSON object with properties “bytesAlg” and “signatures”. The “bytesAlg” in part describes how to generate the payload for purposes of producing a signature. In particular, the “bytesAlg” property describes how the “bytes segments” of the DO will be canonicalized in the payload that is to be signed. The “bytesAlg” property itself is a JSON object containing a property “hashAlg”. The “hashAlg” property could take one of two values:

- “none”, meaning that the bytes segments are included as-is in the signed payload or
- “SHA-256”, meaning that each bytes segment is replaced with the bytes of its SHA-256 digest.

If the “bytesAlg” property is omitted, the value “none” is assumed.

The payload of the JWS signatures is a literal sequence of all of the segments of the DO as serialized according to Appendix A, excluding the signatures segment, with the bytes segments processed according to the value of the "bytesAlg" as defined above.

The “signatures” property shall have one or more values of a JWS as described in RFC 7515. Either the compact serialization or the JSON serialization can be used for generating the JWS. If multiple signatures are to be generated for the same payload, perhaps because there are multiple signers, then the JSON serialization should be preferred. The header of each JWS must specify the signing entity by referencing an identifier in the "kid" field which can be resolved to obtain the signer's public key.

In order to check a signature, the JWS should be confirmed to be well-signed using the private key corresponding to the resolved public key. Then the serialized DO must be checked for equivalency with the serialized DO in the signed payload. In particular, the order may be changed, in either the JSON properties in any of the JSON segments, or in the JSON array of elements in the first JSON segment, or in the overall order of the data elements. Therefore, each JSON segment should be checked for identity with the corresponding JSON segment in the signed payload. Likewise, each bytes segment should be confirmed to correspond to the corresponding bytes segment in the signed payload after transforming it according to the “bytesAlg”.