Workgroup: Delay-Tolerant Networking
Internet-Draft: draft-birrane-dtn-ari-01
Published: 12 March 2023
Intended Status: Standards Track
Expires: 13 September 2023
Authors: E.J. Birrane   E.A. Annis   B. Sipos
JHU/APL   JHU/APL   JHU/APL

Asynchronous Resource Identifier

Abstract

This document defines the structure, format, and features of the naming scheme for the objects defined in the Delay-Tolerant Networking (DTN) Application Data Model (ADM), in support of challenged network management solutions described in the Delay-Tolerant Networking Autonomous Management Architecture (AMA).

This document defines a new Asynchronous Resource Identifier (ARI), based on the structure of a common URI, meeting the needs for a concise, typed, parameterized, and hierarchically organized set of data elements.

Status of This Memo

Copyright Notice

Table of Contents

1. Introduction

The unique limitations of Delay-Tolerant Networking transport capabilities [RFC4838] necessitate increased reliance on individual node behavior. These limitations are considered part of the expected operational environment of the system and, thus, contemporaneous end-to-end data exchange cannot be considered a requirement for successful communication.

The primary DTN transport mechanism, Bundle Protocol version 7, (BPv7) [RFC9171], standardizes a store-and-forward behavior required to communicate effectively between endpoints that may never co-exist in a single network partition. BPv7 might be deployed in static environments, but the design and operation of BPv7 cannot presume that to be the case.

Similarly, the management of any BPv7 protocol agent (BPA) (or any software reliant upon DTN for its communication) cannot presume to operate in a resourced, connected network. Just as DTN transport must be delay-tolerant, DTN network management must also be delay-tolerant.

The DTN Autonomous Management Architecture (DTN AMA) [I-D.ietf-dtn-ama] outlines an architecture that achieves this result through the self-management of a DTN node as configured by one or more remote managers in an asynchronous and open-loop system. An important part of this architecture is the definition of a conceptual data schema for defining resources configured by remote managers and implemented by the local autonomy of a DTN node.

The DTN Asynchronous Management Model (DTN AMM) [I-D.birrane-dtn-adm] defines a logical schema that can be used to represent data types and structures, autonomous controls, and other kinds of information expected to be required for the local management of a DTN node. The DTN AMM further describes a physical data model, called the Application Data Model, that can be defined in the context of applications to create resources in accordance with the DTN AMM logical schema. These named resources can be predefined in moderated publications or custom-defined as part of the operational management of a network or a node.

Every AMM resource must be uniquely identifiable. To accomplish this, an expressive naming scheme is required. The AMM Resource Identifier (ARI) provides this naming scheme. This document defines an ARI, based on the structure of a URI, meeting the needs for a concise, typed, parameterized, and hierarchically organized naming convention.

1.1. Scope

The ARI scheme is based on the structure of a URI [RFC3986] in accordance with the practices outlined in [RFC8820].

ARIs are designed to support the identification requirements of the DTN AMM logical schema. As such, this specification will discuss these requirements to the extent necessary to explain the structure and use of the ARI syntax.

This specification does not constrain the syntax or structure of any existing URI (or part thereof). As such, the ARI scheme does not impede the ownership of any other URI definition and is therefore clear of the concerns presented in [RFC7320].

This specification does not discuss the manner in which ARIs might be generated, populated, and used by applications. The operational utility and configuration of ARIs in a system are described in other documents associated with DTN management, to include the AMA and AMM specifications.

This specification does not describe the way in which path prefixes associated with an ARI are standardized, moderated, or otherwise populated. Path suffixes may be specified where they do not lead to collision or ambiguity.

This specification does not describe the mechanisms for generating either standardized or custom ARIs in the context of any given application, protocol, or network.

This specification does not describe the ways in which an ARI could be encoded into other formats, to include compressed binary formats. However, the design of the ARI syntax discusses compressibility to the extent that the design impacts the ability to create such encodings.

1.2. Use of ABNF

This document defines text structure using the Augmented Backus-Naur Form (ABNF) of [RFC5234]. The entire ABNF structure can be extracted from the XML version of this document using the XPath expression:

```
'//sourcecode[@type="abnf"]'
```

The following initial fragment defines the top-level rules of this document's ABNF.

```
start = ari
```

From the document [RFC3986] the definitions are taken for pchar, path-absolute, and path-noscheme. From the document [RFC5234] the definition is taken for digit.

1.3. Use of CDDL

This document defines Concise Binary Object Representation (CBOR) structure using the Concise Data Definition Language (CDDL) of [RFC8610]. The entire CDDL structure can be extracted from the XML version of this document using the XPath expression:

```
'//sourcecode[@type="cddl"]'
```

The following initial fragment defines the top-level symbols of this document's CDDL, which includes the example CBOR content.

```
start = ari

; Limited sizes to fit the AMP data model
int32 = (int .lt 2147483648) .ge -2147483648
uint32 = uint .lt 4294967296
int64 = (int  .lt 9223372036854775808) .ge -9223372036854775808
uint64 = uint .lt 18446744073709551616
```

This document does not rely on any CDDL symbol names from other documents.

1.4. Terminology

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

Additionally, the following terms are used in this document:

Agent: An entity being managed in the AMA as defined in [I-D.ietf-dtn-ama]. It is expected to be accessible by its Managers over a DTN.

Manager: An entity managing others in the AMA as defined in [I-D.ietf-dtn-ama]. It is expected to be accessible by its Agents over a DTN.

Application Data Model (ADM): Definitions of pre-planned objects being managed on remote agents across challenged networks. An ADM is versioned, but a single version of an ADM cannot change over time once it is registered. This is similar in function to an SMI MIB or an YANG module.

Operational Data Model (ODM): The operational configuration of an Agent, exclusive of the pre-planned objects defined by ADMs. These objects are dynamic configuration applied at runtime, either by Managers in the network or by autonomy on the Agent.

Asynchronous Resource Identifier (ARI): An identifier for any ADM or ODM managed object, as well as ad-hoc managed objects and literal values. ARIs are syntactically conformant to the Uniform Resource Identifier (URI) syntax documented in [RFC3986] and using the scheme name "ari". This is similar in function to an SMI OID or an YANG XPath expression along with parameters.

Namespace A moderated, hierarchical taxonomy of namespaces that describe a set of ADM scopes. Specifically, an individual ADM namespace is a specific sequence of ADM namespaces, from most general to most specific, that uniquely and unambiguously identify the namespace of a particular ADM.

2. ARI Purpose

ADM resources are referenced in the context of autonomous applications on an agent. The naming scheme of these resources must support certain features to inform AMA processing in accordance with the ADM logical schema.

This section defines the set of unique characteristics of the ARI scheme, the combination of which provides a unique utility for naming. While certain other naming schemes might incorporate certain elements, there are no such schemes that both support needed features and exclude prohibited features.

2.1. Resource Parameterization

The ADM schema allows for the parameterization of resources to both reduce the overall data volume communicated between DTN nodes and to remove the need for any round-trip data negotiation.

Parameterization reduces the communicated data volume when parameters are used as filter criteria. By associating a parameter with a data source, data characteristic, or other differentiating attribute, DTN nodes can locally process parameters to construct the minimal set of information to either process for local autonomy or report to remote managers in the network.

Parameterization eliminates the need for round-trip negotiation to identify where information is located or how it should be accessed. When parameters define the ability to perform an associative lookup of a value, the index or location of the data at a particular DTN node can be resolved locally as part of the local autonomy of the node and not communicated back to a remote manager.

2.2. Compressible Structure

The ability to encode information in very concise formats enables DTN communications in a variety of ways. Reduced message sizes increase the likelihood of message delivery, require fewer processing resources to secure, store, and forward, and require less resources to transmit.

While the encoding of an ARI is outside of the scope of this document, the structure of portions of the ARI syntax lend themselves to better compressibility. For example, DTN ADM encodings support the ability to identify resources in as few as 3 bytes by exploiting the compressible structure of the ARI.

The ARI syntax supports three design elements to aid in the creation of more concise encodings: enumerated forms of path segments, relative paths, and patterning.

2.2.1. Enumerated Path Segments

Because the ARI structure includes paths segments with stable enumerated values, each segment can be represented by either its text name or its integer enumeration. For human-readability in text form the text name is preferred, but for binary encoding and for comparisons the integer form is preferred. It is a translation done by the entity handling an ARI to switch between preferred representations (see Section 6); the data model of both forms of the ARI allows for either.

2.2.2. Relative Paths

Hierarchical structures are well known to support compressible encodings by strategically enumerating well-known branching points in a hierarchy. For this reason, the ARI syntax uses the URI path to implement a naming hierarchy.

Supporting relative paths allow for the ARI namespace to be shortened relative to a well-known prefix. By eliminating the need to repeat common path prefixes in ARIs (in any encoding) the size of any given ARI can be reduced.

This relative prefix might be relative to an existing location, such as the familiar "../item" or relative to a defined nickname for a particular path prefix, such as "{root}/item".

2.2.3. Patterning

Patterning in this context refers to the structuring of ARI information to allow for meaning data selection as a function of wildcards, regular expressions, and other expressions of a pattern.

Patterns allow for both better compression and fewer ARI representations by allowing a single ARI pattern to stand-in for a variety of actual ARIs.

This benefit is best achieved when the structure of the ARI is both expressive enough to include information that is useful to pattern match, and regular enough to understand how to create these patterns.

3. ARI Logical Structure

This section describes the components of the ARI scheme to inform the discussion of the ARI syntax in Section 4. At the top-level, an ARI is one of two classes: literal or object reference. Each of these classes is defined in the following subsections.

3.1. Names, Enumerations, Comparisons, and Canonicalizations

Within the ARI logical model, there are a number of domains in which items are identified by a combination of text name and integer enumeration: ADMs, ODMs, literal types, object types, and objects. In all cases, within a single domain the text name and integer enumeration SHALL NOT be considered comparable. It is an explicit activity by any entity processing ARIs to make the translation between text name and integer enumeration (see Section 6).

Text names SHALL be restricted to begin with an alphabetic character followed by any number of other characters, as defined in the id-text ABNF symbol. This excludes a large class of characters, including non-printing characters. When represented in text form, the text name for ODMs is prefixed with a "!" character to disambiguate it from an ADM name (see Section 3.3).

For text names, comparison and uniqueness SHALL be based on case-insensitive logic. The canonical form of text names SHALL be the lower case representation.

Integer enumerations for ADMs and ODMs SHALL be restricted to a magnitude less than 2**63 to allow them to fit within a signed 64-bit storage. The ADM registration in Table 5 reserves high-valued code points for private and experimental ADMs, while the entire domain of ODM code points (negative integers) is considered private use. Integer enumerations for primitive types and object types SHALL be restricted to a magnitude less than 2**31 to allow them to fit within a signed 32-bit storage. The registrations in Table 3 and Table 4 respectively Integer enumerations for objects (within an ADM or ODM) SHALL be restricted to a magnitude less than 2**31 to allow them to fit within a signed 32-bit storage, although negative-value object enumerations are disallowed.

For integer enumerations, comparison and uniqueness SHALL be based on numeric values not on encoded forms. The canonical form of integer enumerations in text form SHALL be the shortest length decimal representation.

3.2. Literals

Literals represent a special class of ARI which are not associated with any particular ADM or ODM. A literal has no other name other than its value, but literals may be explicitly typed in order to force the receiver to handle it in a specific way.

Because literals will be based on the CBOR data model [RFC8949] and its extended diagnostic notation, a literal has an intrinsic representable data type as well as an AMP data type. The CBOR primitive types are named CDDL symbols as defined in Section 3.3 of [RFC8610].

When converting from AMP primitive types, the chosen CBOR type SHALL be determined by the mapping in Table 1. Additionally, when handling typed literal ARIs any combination of AMP primitive type and CBOR primitive type not in Table 1 SHALL be considered invalid. This restriction is enforced by the CDDL defined in Section 5. Additionally, when handling a literal of AMP type CBOR the well-formed-ness of the CBOR contained SHOULD be verified before the literal is treated as valid.

AMP Primitive Type	**Used CBOR Primitive Type**
BOOL	bool
BYTE	uint
INT	int
UINT	uint
VAST	int
UVAST	uint
REAL32	float
REAL64	float
TV	int
TS	int

AMP Primitive Type	Used CBOR Primitive Type
STR	tstr
LABEL	tstr
BYTESTR	bstr
CBOR	bstr

Table 1: Literal Types to CBOR Primitives

When interpreting an untyped literal ARI, the implied AMP primitive type SHALL be determined by the mapping in Table 2.

CBOR Primitive Type	Implied AMP Primitive Type
bool	BOOL
uint	UVAST
nint	VAST
float16, float32	FLOAT32
float64	FLOAT64
bstr	BYTESTR
tstr	STR

Table 2: Literal Implied and Allowed Types

3.3. Object References

Object references are composed of two parts: object identity and optional parameters. The object identity can be dereferenced to a specific object in the ADM/ODM, while the parameters provide additional information for certain types of object and only when allowed by the parameter "signature" from the ADM/ODM.

The object identity itself contains the components, described in the following subsections: namespace, object type, and object name. When encoded in text form (see Section 4), the identity components correspond to the URI path segments.

3.3.1. Namespace

ADM resources exist within namespaces to eliminate the possibility of a conflicting resource name, aid in the application of patterns, and improve the compressibility of the ARI. Namespaces **SHALL NOT** be used as a security mechanism to manage access. An Agent or Manager **SHALL NOT** infer security information or access control based solely on namespace information in an ARI.

Namespaces have two possible forms; one more human-friendly and one more compressible:

Text form: This form corresponds with a human-readable identifier for either an ADM or a ODM namespace. The text form is not compressible and needs to be converted to a numeric namespace based on a local registry. A text form namespace SHALL contain only URI path segment characters.

Numeric form: This form corresponds with a compressible value suitable for on-the-wire encoding between Manager and Agent. Sorting and matching numeric namespaces is also faster than text form. A numeric form namespaces SHALL be small enough to be represented as a 64-bit signed integer.

Independent to the form of the namespace is the issuer of the namespace, which is one of:

ADM namespace: When a namespace is associated with an ADM, its text form SHALL begin with an alphabetic character and its numeric form SHALL be a positive integer. All ADM namespaces are universally unique and, except for private or experimental use, SHOULD be registered with IANA (see Table 5).

ODM namespace: When a namespace is not associated with an ADM, its text form SHALL begin with a bang character "!" and its numeric form SHALL be a negative integer. These namespaces do not have universal registration and SHALL be considered to be private use. It is expected that runtime ODM namespaces will be allocated and managed per-user and per-mission.

3.3.2. Object Type

Due to the flat structure of an ADM, as defined in [I-D.birrane-dtn-adm], all managed objects are of a specific and unchanging type from a set of available managed object types. The preferred form for object types in text ARIs is the text name, while in binary form it is the integer enumeration (see Section 6).

The following subsection explains the form of those object identifiers.

3.3.3. Object Name

An object is any one of a number of data elements defined for the management of a given application or protocol that conforms to the ADM logical schema.

Within a single ADM or runtime namespace and a single object type, all managed objects have similar characteristics and all objects are identified by a single text name or integer enumeration. The preferred form for object names in text ARIs is the text name, while in binary form it is the integer enumeration. Any ADM-defined object will have both name and enumeration, while a runtime-defined object can have either but not both. Conversion between the two forms requires access to the original ADM, and its specific revision, in which the object was defined.

3.3.4. Parameters

The ADM logical schema allows many object types to be parameterized when defined in the context of an application or a protocol.

If two instances of an ADM resource have the same namespace and same object type and object name but have different parameter values, then those instances are unique and the ARIs for those instances MUST also be unique. Therefore, parameters are considered part of the ARI syntax.

The ADM logical schema defines two types of parameters: Formal and Actual. The terms formal parameter and actual parameter follow common computer programming vernacular for discussing function declarations and function calls, respectively.

Formal Parameters:
Formal parameters define the type, name, and order of the information that customizes an ARI. They represent the unchanging "definition" of the parameterized object. Because ARIs represent a *use* of an object and not its definition, formal parameters are not present in an ARI.

Actual Parameters:
Actual parameters represent the data values used to distinguish different instances of a parameterized object.

An actual parameter MUST specify a value and MAY specify a type. If a type is provided it MUST match the type provided by the formal parameter. An actual parameter MUST NOT include NAME information.

Including type information in an actual parameters allows for explicit type checking of a value, which might otherwise be implicitly cast.

There are two ways in which the value of an actual parameter can be specified: parameter-by-value and parameter-by-name.

Parameter-By-Value: This method involves directly supplying the value as part of the actual parameter. It is the default method for supplying values.

Parameter-By-Name: This method involves specifying the name of some other parameter and using that other parameter's value for the value of this parameter. This method is useful when a parameterized ARI contains another parameterized ARI. The contained object's actual parameter can be given as the name of the containing ARI's parameter. In that way, a containing ARI's parameters can be "flowed down" to all of the objects it contains.

4. ARI Text Form

This section defines how the data model explained in Section 3 is encoded as text conforming to the URI syntax of [RFC3986]. The most straightforward text form of ARI uses an explicit scheme and an absolute path (starting with an initial slash "/"), which requires no additional context to interpret its structure.

When used within the context of a base ARI, the URI Reference form of Section 4.4 can be used. In all other cases an ARI must be an absolute-path form and contain a scheme.

While this text description is normative, the ABNF schema in this section provides a more explicit and machine-parsable text schema. The scheme name of the ARI is "ari" and the scheme-specific part of the ARI follows one of the two forms corresponding to the literal-value ARI and the object-reference ARI.

```
ari = absolute-ari / relative-ari

absolute-ari = "ari:" ari-ssp
ari-ssp = ari-ssp-literal / ari-ssp-objref

; A text name must start with an alphabetic character
id-text = ALPHA *pchar
; An integer enumeration must contain only digits
id-num = 1*DIGIT
```

4.1. URIs and Percent Encoding

Due to the intrinsic structure of the URI, on which the text form of ARI is based, there are limitations on the syntax available to the scheme-specific-part [RFC7595]. One of these limitations is that each path segment can contain only characters in the pchar ABNF symbol defined in [RFC3986]. For most parts of the ARI this restriction is upheld by the values themselves: ADM/ODM names, type names, and object names have a limited character set as well. For literals and nested parameters though, the percent encoding of Section 2.4 of [RFC3986] is needed.

In the ARI text examples in this document the URIs have been percent-decoded for clarity, as might be done in an ARI display and editing tool. But the actual encoded form of the human-friendly ARI ari:"text" is ari:%22text%22. Outside of literals, the safe characters which are not be percent-encoded are the structural delimiters /()[], used for parameters and ARI collections.

One other aspect of convenience for human editing of text-form ARIs is linear white space. The current ABNF pattern, staying within the URI pattern, do not allow for whitespace to separate list items or otherwise. A human editing an ARI could find it convenient to include whitespace following commas between list items, or to separate large lists across lines. Any tool that allows this kind of convenience of editing SHALL collapse any white space within a single ARI before encoding its contents.

4.2. Literals

Based on the structure of Section 3.2, the text form of the literal ARI contains only a URI path with an optional AMP primitive type. A literal has no concept of a namespace or context, so the path is always absolute. When the path has two segments, the first is the AMP primitive type and the second is the encoded literal value. When the path has a single segment it is the encoded literal value. As a shortcut, an ARI with only a single path segment is necessarily an untyped literal so the leading slash can be elided.

An ARI encoder or decoder SHALL handle both text name and integer enumeration forms of the primitive type. When present and able to be looked up, the primitive type SHOULD be a text name.

The literal value SHALL be the percent encoded form of the CBOR extended diagnostic notation text of Appendix G of [RFC8610]. When untyped, the decoded literal value SHALL be one of the primitive types named by the lit-notype CDDL symbol of Section 5.2. When typed, the decoded literal value MAY be any valid CBOR item conforming to the AMP primitive type definition.

Some example of the forms for a literals are below. These first are untyped primitive values:

```
ari:true
ari:"text"
ari:10
```

And these are typed values:

```
ari:/UINT/10
ari:/LABEL/"name"
ari:/CBOR/<<10>>
```

The literal-value ARI has a corresponding ABNF definition of:

```
; The primitive type name is optional
ari-ssp-literal = ["/" lit-type] ["/"] lit-value
; Type is restricted to valid AMP primitive types
lit-type = id-text / id-num
; The value is percent-encoded CBOR Diagnostic syntax
lit-value = *pchar
```

4.3. Object References

Based on the structure of Section 3.3, the text form of the object reference ARI contains a URI with three path segments corresponding to the namespace-id, object-type, and object-id. Those three segments (excluding parameters as defined below) are referred to as the object identity.

An ARI encoder or decoder SHALL handle both text name and integer enumeration forms of the namespace-id, object-type, and object-id.

The final segment containing the object-id MAY contain parameters enclosed by parentheses "(" and ")". There is no semantic distinction between the absence of parameters and the empty parameter list. The parameter list SHALL be separated by comma characters ",". Each parameter item SHALL be either an ARI or an ARI collection. Within a parameter item, ARI collections SHALL be indicated by enclosing square brackets "[" and "]". The ARI collection list SHALL be separated by comma characters ",". Each parameter item is handled recursively as the text form of ARI.

The parameters as a whole SHALL be the percent encoded form of the constituent ARIs, excluding the structural delimiters /()[],. Implementations are advised to be careful about the percent encoded vs. decoded cases of each of the nested ARIs within parameters to avoid duplicate encoding or decoding. It is recommended to dissect the parameters and ARI collections in their encoded form first, and then to dissect and percent decode each separately and recursively.

```
ari:/adm-a/EDD/someobj
ari:/adm-a/CTRL/otherobj(true,3)
ari:/adm-a/CTRL/otherobj("a param",/UINT/10)
ari:/41/-1/0
```

The object-reference ARI has a corresponding ABNF definition of:

```
ari-ssp-objref = obj-ident [paramlist]
; The object identity can be used separately than parameters
obj-ident = "/" ns-id "/" obj-type "/" obj-id

; A comma-separated list of parameters with enclosure
paramlist = "(" param *("," param) ")"
param = ari / ac

ns-id = ns-adm / ns-odm
ns-adm = id-text / id-num
ns-odm = ("!" id-text) / ("-" id-num)
; Type is restricted to valid AMP primitive types
obj-type = id-text / ("-" id-num)
obj-id = id-text / id-num

; A comma-separated list of any form of ARI with enclosure
ac = "[" ari *("," ari) "]"
```

4.4. URI References

The text form of ARI can contain a URI Reference, as defined in Section 3 of [RFC3986], which can only be resolved using a base URI using the algorithm defined in Section 5 of [RFC3986]. When resolving nested ARI content, the base URI of any interior resolution is the next-outer ARI in the nested structure. The outermost ARI SHALL NOT be a URI Reference because it will have no base URI to resolve with.

Because a relative-path ARI with no path separators is considered to be an untyped literal, an ARI reference SHALL contain at least one path separator. For the case where the ARI reference is to a sibling object from the base URI the relative path SHOULD be of the form "./" to include the path separator.

When resolving nested ARI content, the parameters of the URI reference SHALL be preserved in the resolved ARI. This behavior is equivalent to the query parameter portion when resolving a generic URI reference.

```
; Relative ARI must be resolved before interpreting
relative-ari = path-nonempty [paramlist]

; Non-empty absolute or relative path
path-nonempty = path-absolute / path-noscheme
```

4.5. Patterns

Because each of the text form use path segments to delimit the components of the absolute ARI, and due to the restrictions of the ARI path segment content, it is possible for URI reserved characters to be able to provide wildcard-type patterns. Although the form is similar, an ARI Pattern is not itself an ARI and they cannot be used interchangeably. The context used to interpret and match an ARI Pattern SHALL be explicit and separate from that used to interpret and dereference an ARI.

The ARI Pattern SHALL NOT ever take the form of a URI Reference; only as an absolute URI. An ARI Pattern SHALL NOT ever contain parameters, only identity.

An ARI Pattern has no optional path segments. When used as a literal ARI pattern the path SHALL have two segments. When used as an object-reference ARI pattern the path SHALL have three segments.

The single-wildcard is the only defined segment pattern and a segment can either be a real ID or a single wildcard.

```
ari-pat = "ari:" ari-pat-ssp
ari-pat-ssp = ari-pat-literal / ari-pat-objref

ari-pat-literal = "/" id-pat "/" id-pat
ari-pat-objref = "/" id-pat "/" id-pat "/" id-pat

; The non-wildcard symbol is the same as ARI syntax
id-pat = wildcard / (*pchar)
wildcard = "*"
```

5. ARI Binary Form

This section defines how the data model explained in Section 3 is encoded as a binary sequence conforming to the CBOR syntax of [RFC8949]. Within this section the term "item" is used to mean the CBOR-decoded data item which follows the logical model of CDDL [RFC8610].

The binary form of the URI is intended to be used for machine-to-machine interchange so it is missing some of the human-friendly shortcut features of the ARI text form from Section 4. It still follows the same logical data model so it has a one-for-one representation of all of the styles of text-form ARI.

A new CBOR tag TBD999999 has been registered to indicate that an outer CBOR item is a binary-form ARI. This is similar in both syntax and semantics to the "ari" URI scheme in that for a nested ARI structure, only the outer-most ARI need be tagged. The inner ARIs are necessarily interpreted as such based on the nested ARI schema of this section.

While this text description is normative, the CDDL schema in this section provides a more explicit and machine-parsable binary schema.

```
; An ARI can be tagged if helpful
ari = ari-notag / #6.999999(ari-notag)
ari-notag = lit-ari / ari-objref
```

5.1. Intermediate CBOR

The CBOR item form is used as an intermediate encoding between the ARI data and the ultimate binary encoding. When decoding a binary form ARI, the CBOR must be both "well-formed" according to [RFC8949] and "valid" according to the CDDL model of this specification. Implementations are encouraged, but not required, to use a streaming form of CBOR encoder/decoder to reduce memory consumption of an ARI handler. For simple implementations or diagnostic purposes, a two stage conversion between ARI--CBOR and CBOR--binary can be more easily understood and tested.

5.2. Literals

Based on the structure of Section 3.2, the binary form of the literal ARI contains a data item along with an optional AMP primitive type. In order to keep the encoding as short as possible, the untyped literal is encoded as the simple value itself. Because the typed literal and the object-reference forms uses CBOR array framing, this framing is used to disambiguate from the pure-value encoding of the lit-notype CDDL symbol.

When present, the primitive type SHALL be an integer enumeration. When untyped, the decoded literal value SHALL be one of the primitive types named by the lit-notype CDDL symbol. When typed, the decoded literal value MAY be any valid CBOR item conforming to the AMP primitive type definition.

Some example of the forms for a literal are below. These first are untyped primitive values:

```
true
```

```
"text"
```

```
10
```

And these are typed values:

```
[4, 10]
```

```
[15, <<10>>]
```

The literal-value ARI has a corresponding CDDL definition of:

```
lit-ari = lit-typeval / lit-notype

lit-notype = bool / int / float / tstr / bstr

lit-typeval = $lit-typeval .within lit-typeval-struct
lit-typeval-struct = [
  lit-type: (int32 .ge 0),
  lit-value: any
]

; FIXME: will expand with assigned types
$lit-typeval /= [1, bool]
$lit-typeval /= [2, uint .size 1] ; 1-byte
$lit-typeval /= [4, int32] ; 4-byte
$lit-typeval /= [5, uint32] ; 4-byte
$lit-typeval /= [6, uint64] ; 8-byte
$lit-typeval /= [7, int64] ; 8-byte
$lit-typeval /= [8, float16 / float32]
$lit-typeval /= [9, float64]
$lit-typeval /= [10, tstr]
$lit-typeval /= [11, bstr]

$lit-typeval /= [12, int]
$lit-typeval /= [13, int]
$lit-typeval /= [14, tstr .regexp "[A-Za-z].*"]
$lit-typeval /= [15, bstr .cbor any]
```

5.3. Object References

Based on the structure of Section 3.3, the binary form of the object reference ARI is a CBOR-encoded item. An ARI SHALL be encoded as a CBOR array with at least three items corresponding to the namespace-id, object-type, and object-id. Those three items are referred to as the object identity. The optional fourth item of the array is the parameter list.

The namespace-id SHALL be present only as an integer enumeration. The object-type SHALL be present only as an integer enumeration. The object-id SHALL be present as either a text name or an integer enumeration. The processing of text name object identity components by an Agent is optional and SHALL be communicated to any associated Manager prior to encoding any ARIs for that Agent.

When present, the parameter list SHALL be a CBOR array containing either ARI or ARI collection items. The CBOR tag 41 (meaning a homogeneous array per [IANA-CBOR]) SHALL be used to indicate that a parameter item is an ARI collection. All other, untagged parameter items SHALL be handled as an ARI.

An example object reference without parameters is:

```
[41, -1, 0]
```

Another example object reference with parameters is:

```
[41, -2, 3, ["a param", [4, 10]]]
```

The object-reference ARI has a corresponding CDDL definition of:

```
ari-objref = [obj-ident, ?params]
obj-ident = (
  ns-id,
  obj-type,
  obj-id,
)
ns-id = int64
obj-type = $obj-type-reg .within (int32 .lt 0)
obj-id = (int32 .ge 0) / tstr
params = [*ari-or-ac]

ari-or-ac = ari / ac
ac = #6.41([*ari])

; FIXME: will expand with assigned types
$obj-type-reg = nint
```

5.4. URI References

TBD

5.5. Patterns

TBD

6. Transcoding Considerations

When translating literal types into text form and code point lookup tables are available, the primitive type SHOULD be converted to its text name. When translating literal types from text form and code point lookup tables are available, the primitive type SHOULD be converted from its text name. The conversion between AMP primitive type name and enumeration requires a lookup table based on the registrations in Table 3.

When translating literal values into text form, it is necessary to canonicalize the CBOR extended diagnostic notation of the item. The following applies to generating text form from CBOR items:

* The canonical text form of CBOR bool values SHALL be the forms identified in Section 8 of [RFC8949].

* The canonical text form of CBOR int and float values SHALL be the decimal form defined in Section 8 of [RFC8949].

* The canonical text form of CBOR tstr values SHALL be the definite-length, non-concatenated form defined in Section 8 of [RFC8949].

* The canonical text form of CBOR bstr values SHALL be the definite-length, base16 ("h" prefix), non-concatenated form defined in Section 8 of [RFC8949].

* When presenting the AMP primitive type of CBOR the values SHALL be the embedded CBOR form defined in Appendix G.3 of [RFC8610].

When translating object references into text form and code point lookup tables are available, any enumerated item SHOULD be converted to its text name. When translating object references from text form and code point lookup tables are available, any enumerated item SHOULD be converted from its text name. The conversion between AMP object-type name and enumeration requires a lookup table based on the registrations in Table 4. The conversion between name and enumeration for either namespace-id or object-id require lookup tables based on ADMs and ODMs known to the processing entity.

7. Interoperability Considerations

DTN challenged networks might interface with better resourced networks that are managed using non-DTN management protocols. When this occurs, the federated network architecture might need to define management gateways that translate between DTN and non-DTN management approaches.

NOTE: It is also possible for DTN management be used end-to-end because this approach can also operate in less challenged networks. The opposite is not true; non-DTN management approaches should not be assumed to work in DTN challenged networks.

Where possible, ARIs should be translatable to other, non-DTN management naming schemes. This translation might not be 1-1, as the features of the ADM may differ from features in other management naming schemes. Therefore, it is unlikely that a single naming scheme can be used for both DTN and non-DTN management.

8. Security Considerations

Because ADM and ODM namespaces are defined by any entity, no security or permission meaning can be inferred simply from the expression of namespace.

9. IANA Considerations

This section provides guidance to the Internet Assigned Numbers Authority (IANA) regarding registration of schema and namespaces related to the ADM Resource Identifier (ARI), in accordance with BCP 26 [RFC1155].

9.1. URI Schemes Registry

This document defines a new URI scheme "ari" in Section 4. A new entry has been added to the "URI Schemes" registry [IANA-URI] with the following parameters.

Scheme name:
ari

Status:
Permanent

Applications/protocols that use this scheme name:
The scheme is used by AMP Managers and Agents to identify managed objects.

Contact:
IETF Chair <chair@ietf.org>

Change controller:
IESG <iesg@ietf.org>

Reference:
Section 4 of [This document].

9.2. CBOR Tags Registry

This document defines a new CBOR tag TBD999999 in Section 5. A new entry has been added to the "CBOR Tags" registry [IANA-CBOR] with the following parameters.

Tag:
TBD999999

Data Item:
multiple

Semantics:
Used to tag a binary-form DTNMP ARI

Reference:
Section 5 of [This document].

9.3. DTN Management Protocol Registry

This document defines a new sub-registry "Primitive Types" within the "DTN Management Protocol" registry [IANA-DTNMP] containing the following initial entries. Enumerations in this sub-registry are non-negative integers representable as CBOR uint type with an argument shorter than 4-bytes. The registration procedure for this sub-registry is Specification Required.

Enumeration	Name	Reference	Description
TBD1	BOOL	[This document]	A native boolean value.
TBD2	BYTE	[This document]	An 8-bit unsigned integer.
TBD4	INT	[This document]	A 32-bit signed integer.
TBD5	UINT	[This document]	A 32-bit unsigned integer.
TBD6	VAST	[This document]	A 64-bit signed integer.
TBD7	UVAST	[This document]	A 64-bit unsigned integer.
TBD8	REAL32	[This document]	A 32-bit [IEEE.754-2019] floating point number.
TBD9	REAL64	[This document]	A 64-bit [IEEE.754-2019] floating point number.
TBD10	STR	[This document]	A text string composed of characters.
TBD11	BYTESTR	[This document]	A byte string composed of 8-bit values.
TBD12	TV	[This document]	
TBD13	TS	[This document]	
TBD14	LABEL	[This document]	A text label of a parent object parameter. This is only valid in a nested parameterized ARI.
TBD15	CBOR	[This document]	A byte string containing an encoded CBOR item. The structure is opaque to the Agent but guaranteed well-formed for the ADM using it.
TBD16 to 65279			*Unassigned*
65280 to 2147483647		[This document]	Enumerations that are 2**16-2**8 and larger are reserved for private or experimental use.

Table 3: Primitive Types

This document defines a new sub-registry "Managed Object Types" within the "DTN Management Protocol" registry [IANA-DTNMP] containing the following initial entries. Enumerations in this sub-registry are negative integers representable as CBOR nint type with an argument shorter than 4-bytes. The registration procedure for this sub-registry is Specification Required.

Enumeration	**Name**	**Reference**	**Description**
-*TBD1*	MDAT	[This document]	ADM Metadata
-*TBD2*	CONST	[This document]	Constant
-*TBD3*	CTRL	[This document]	Control
-*TBD4*	EDD	[This document]	Externally Defined Data
-*TBD5*	MAC	[This document]	Macro
-*TBD6*	OPER	[This document]	Operator
-*TBD7*	RPTT	[This document]	Report Template
-*TBD8*	SBR	[This document]	State-Based Rule
-*TBD9*	TBLT	[This document]	Table Template
-*TBD10*	TBR	[This document]	Time-Based Rule
-*TBD11*	VAR	[This document]	Variable
TBD12 to 65279			*Unassigned*
65280 to 2147483647		[This document]	Enumerations that are 2**16-2**8 and larger are reserved for private or experimental use.

Table 4: Managed Object Types

This document defines a new sub-registry "Application Data Models" within the "DTN Management Protocol" registry [IANA-DTNMP] containing the following initial entries. Enumerations in this sub-registry are non-negative integers representable as CBOR uint type with an argument shorter than 8-bytes. The registration procedure for this sub-registry is Specification Required.

Enumeration	Name	Reference	Notes
0		[This document]	Value zero is reserved.
1 to 4294967296			*Unassigned*
4294967296 and larger		[This document]	Enumerations that are larger than 32-bit are reserved for private or experimental use.

Table 5: Application Data Models

The Operational Data Models code points are all private use, so do not need to have an IANA registry defined.

10. References

10.1. Normative References

[IANA-CBOR] IANA, "Concise Binary Object Representation (CBOR) Tags", <https://www.iana.org/assignments/cbor-tags/>.

[IANA-DTNMP] IANA, "Delay-Tolerant Networking (DTN) Management Protocol", <https://www.iana.org/assignments/TBD/>.

[IANA-URI] IANA, "Uniform Resource Identifier (URI) Schemes", <https://www.iana.org/assignments/uri-schemes/>.

[IEEE.754-2019] IEEE, "IEEE Standard for Floating-Point Arithmetic", IEEE IEEE 754-2019, DOI 10.1109/IEEESTD.2019.8766229, 18 July 2019, <https://ieeexplore.ieee.org/document/8766229>.

[RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, DOI 10.17487/RFC2119, March 1997, <https://www.rfc-editor.org/info/rfc2119>.

[RFC3986] Berners-Lee, T., Fielding, R., and L. Masinter, "Uniform Resource Identifier (URI): Generic Syntax", STD 66, RFC 3986, DOI 10.17487/RFC3986, January 2005, <https://www.rfc-editor.org/info/rfc3986>.

[RFC5234] Crocker, D., Ed. and P. Overell, "Augmented BNF for Syntax Specifications: ABNF", STD 68, RFC 5234, DOI 10.17487/RFC5234, January 2008, <https://www.rfc-editor.org/info/rfc5234>.

[RFC7595] Thaler, D., Ed., Hansen, T., and T. Hardie, "Guidelines and Registration Procedures for URI Schemes", BCP 35, RFC 7595, DOI 10.17487/RFC7595, June 2015, <https://www.rfc-editor.org/info/rfc7595>.

[RFC8174]
Leiba, B., "Ambiguity of Uppercase vs Lowercase in RFC 2119 Key Words", BCP 14, RFC 8174, DOI 10.17487/RFC8174, May 2017, <https://www.rfc-editor.org/info/rfc8174>.

[RFC8949] Bormann, C. and P. Hoffman, "Concise Binary Object Representation (CBOR)", STD 94, RFC 8949, DOI 10.17487/RFC8949, December 2020, <https://www.rfc-editor.org/info/rfc8949>.

[RFC9171] Burleigh, S., Fall, K., and E. Birrane, III, "Bundle Protocol Version 7", RFC 9171, DOI 10.17487/RFC9171, January 2022, <https://www.rfc-editor.org/info/rfc9171>.

10.2. Informative References

[RFC1155] Rose, M. and K. McCloghrie, "Structure and identification of management information for TCP/IP-based internets", STD 16, RFC 1155, DOI 10.17487/RFC1155, May 1990, <https://www.rfc-editor.org/info/rfc1155>.

[RFC4838] Cerf, V., Burleigh, S., Hooke, A., Torgerson, L., Durst, R., Scott, K., Fall, K., and H. Weiss, "Delay-Tolerant Networking Architecture", RFC 4838, DOI 10.17487/RFC4838, April 2007, <https://www.rfc-editor.org/info/rfc4838>.

[RFC7320] Nottingham, M., "URI Design and Ownership", RFC 7320, DOI 10.17487/RFC7320, July 2014, <https://www.rfc-editor.org/info/rfc7320>.

[RFC8610] Birkholz, H., Vigano, C., and C. Bormann, "Concise Data Definition Language (CDDL): A Notational Convention to Express Concise Binary Object Representation (CBOR) and JSON Data Structures", RFC 8610, DOI 10.17487/RFC8610, June 2019, <https://www.rfc-editor.org/info/rfc8610>.

[RFC8820] Nottingham, M., "URI Design and Ownership", BCP 190, RFC 8820, DOI 10.17487/RFC8820, June 2020, <https://www.rfc-editor.org/info/rfc8820>.

[I-D.ietf-dtn-ama] Birrane, E. J., Annis, E., and S. Heiner, "Asynchronous Management Architecture", Work in Progress, Internet-Draft, draft-ietf-dtn-ama-03, 25 October 2021, <https://datatracker.ietf.org/doc/html/draft-ietf-dtn-ama-03>.

[I-D.birrane-dtn-adm] Birrane, E. J., DiPietro, E., and D. Linko, "AMA Application Data Model", Work in Progress, Internet-Draft, draft-birrane-dtn-adm-03, 2 July 2018, <https://datatracker.ietf.org/doc/html/draft-birrane-dtn-adm-03>.

Appendix A. Examples

The examples in this section rely on the ADM and ODM definitions in Table 6 and Table 7 respectively.

Enumeration	Name
10	adm10
20	adm20

Table 6: Example ADMs

Enumeration	Name
-10	odm10

Table 7: Example ODMs

Given those namespaces, the example objects are listed in Table 6 where the Namespace column uses the ARI text form convention.

Namespace	Object Type	Enumeration	Name	Signature
adm10	EDD	3	num_bytes	()
adm10	CTRL	2	do_thing	(AC targets, UINT count)
adm10	RPTT	1	rpt_with_param	(ARI var, STR text)
!odm10	VAR	1	my_counter	()

Table 8: Example Objects

Each of the following examples illustrate the comparison of ARI forms in different situations, covering the gamut of what can be expressed by an ARI.

A.1. Typed Literal

This is the literal value 4 interpreted as a 32-bit unsigned integer. The ARI text (which is identical to its percent-encoded form) is:

```
ari:/UINT/4
```

which is translated to enumerated form:

```
ari:/5/4
```

and converted to CBOR item:

```
[5, 4]
```

and finally to the binary string of:

```
0x820504
```

A.2. Complex CBOR Literal

This is a literal value embedding a complex CBOR structure. The CBOR diagnostic expression being encoded is

```
{"test": [3, 4.5]}
```

which is CBOR-encoded to a byte string and percent-encoded to the URL path:

```
ari:/CBOR/h%27A164746573748203F94480%27
```

which is translated to enumerated form:

```
ari:/15/h%27A164746573748203F94480%27
```

and converted to CBOR item (note the byte string is no longer text-encoded):

```
[15, h'A164746573748203F94480']
```

and finally to the binary string of:

```
0x820F4BA164746573748203F94480
```

A.3. Non-parameterized Object Reference

This is a non-parameterized num_bytes object in the ADM namespace. The ARI text (which is identical to its percent-encoded form) is:

```
ari:/adm10/edd/num_bytes
```

which is translated to enumerated form:

```
ari:/10/-4/3
```

and converted to CBOR item:

```
[10, -4, 3]
```

and finally to the binary string of:

```
0x830A2303
```

A.4. Parameterized Object Reference

This is an parameterized do_thing object in the ADM namespace. Additionally, the parameters include two relative-path ARI References to other objects in the same ADM, which are resolved after text-decoding. The ARI text (which is identical to its percent-encoded form) is:

```
ari:/adm10/ctrl/do_thing([../edd/num_bytes,/!odm10/var/my_counter],3)
```

which is translated to enumerated and resolved form:

```
ari:/10/-3/2([/10/-4/3,/-10/-11/1],3)
```

and converted to CBOR item:

```
[10, -3, 2, [
  41([
    [10, -4, 3],
    [10, -11, 1]
  ]),
  3
]]
```

and finally to the binary string of:

```
0x840A220282D82982830A2303830A2A0103
```

A.5. Recursive Structure with Percent Encodings

This is a complex example having nested ARIs, some with percent-encoding needed. The human-friendly (but not valid URI) text for this case is:

```
ari:/adm10/rptt/rpt_with_param("text")
```

which is percent encoded to the real URI:

```
ari:/adm10/rptt/rpt_with_param(%22text%22)
```

which is translated to enumerated form:

```
ari:/10/-7/1(%22text%22)
```

and converted to CBOR item:

```
[10, -7, 1, ["text"]]
```

and finally to the binary string of:

```
0x840A2601816474657874
```

Authors' Addresses

Edward J. Birrane, III
The Johns Hopkins University Applied Physics Laboratory
11100 Johns Hopkins Rd.
Laurel, MD 20723
United States of America

Phone: +1 443 778 7423
Email: Edward.Birrane@jhuapl.edu

Emery Annis
The Johns Hopkins University Applied Physics Laboratory

Email: Emery.Annis@jhuapl.edu

Brian Sipos
The Johns Hopkins University Applied Physics Laboratory

Email: brian.sipos+ietf@gmail.com