Abstract

This document describes a concise binary encoding of components of the Asynchronous Management Model (AMM) and a binary protocol for the exchange of these items over a network. The intent of this protocol is to provide an efficient exchange of management encoding that conserves computing resources for embedded devices and energy necessary for transmission of protocol data units. AMP is designed to reduce the number of transmitted bytes, operates without sessions or (concurrent) two-way links, and functions autonomously when there is no timely contact with a network operator.

Status of This Memo

This Internet-Draft is submitted in full conformance with the provisions of BCP 78 and BCP 79.

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF). Note that other groups may also distribute working documents as Internet-Drafts. The list of current Internet-Drafts is at https://datatracker.ietf.org/drafts/current/.

Internet-Drafts are draft documents valid for a maximum of six months and may be updated, replaced, or obsoleted by other documents at any time. It is inappropriate to use Internet-Drafts as reference material or to cite them other than as "work in progress."

This Internet-Draft will expire on December 6, 2018.

Copyright Notice

Copyright (c) 2018 IETF Trust and the persons identified as the document authors. All rights reserved.

This document is subject to BCP 78 and the IETF Trust’s Legal Provisions Relating to IETF Documents (https://trustee.ietf.org/license-info) in effect on the date of publication of this document. Please review these documents carefully, as they describe your rights and restrictions with respect
to this document. Code Components extracted from this document must include Simplified BSD License text as described in Section 4.e of the Trust Legal Provisions and are provided without warranty as described in the Simplified BSD License.

Table of Contents

1. Introduction .................................................. 3
2. Requirements Language ........................................ 3
3. Scope .................................................................. 3
   3.1. Protocol Scope ............................................ 3
   3.2. Specification Scope ....................................... 3
4. Terminology ...................................................... 4
5. Constraints and Assumptions ..................................... 4
6. Technical Notes .................................................. 5
7. AMP-Specific Concepts .......................................... 6
   7.1. Nicknames (NN) ............................................. 6
   7.1.1. Motivation for AId Compression ....................... 7
   7.1.2. ADM Enumeration ...................................... 7
   7.1.3. ADM Template Area Enumeration ....................... 7
   7.1.4. Nickname Definition .................................... 8
   7.1.5. ADM Enumeration Considerations ..................... 9
8. Encodings ....................................................... 9
   8.1. CBOR Considerations ...................................... 9
   8.2. AMM Models ............................................... 10
   8.2.1. Primitive and Derived Types ......................... 10
   8.2.2. Derived Types ......................................... 12
   8.2.3. Type-Name-Value (TNV) ................................. 12
   8.2.4. ADM Identifier (AId) .................................. 13
     8.3. Collections ............................................. 17
     8.3.1. TNV Collection ....................................... 17
     8.3.2. Types Then Value Collection (TTVC) ............... 18
     8.3.3. AId Collections (AC) ................................ 19
     8.3.4. Expressions (EXPR) .................................. 20
     8.3.5. Predicate (PRED) ..................................... 20
   8.4. ADM Object Definition Encodings ......................... 20
     8.4.1. Externally Defined Data (EDD) ....................... 21
     8.4.2. Constants (CONST) ................................... 21
     8.4.3. Controls (CTRL) ...................................... 22
     8.4.4. Macros (MAC) ......................................... 22
     8.4.5. Operators (OP) ....................................... 23
     8.4.6. Report Templates (RPTT) ............................ 24
     8.4.7. Report (RPT) ......................................... 24
     8.4.8. State-Based Rules (SRL) ............................. 25
     8.4.9. Table Templates (TBLT) ............................. 27
     8.4.10. Tables (TBL) ......................................... 28
     8.4.11. Time-Based Rules (TRL) ............................ 29
     8.4.12. Variables (VAR) ..................................... 30
1. Introduction

Network management in challenged and resource constrained networks must be accomplished differently than the network management methods in high-rate, high-availability networks. The Asynchronous Management Architecture (AMA) [I-D.birrane-dtn-ama] provides an overview and justification of an alternative to "synchronous" management services such as those provided by NETCONF. In particular, the AMA defines the need for a flexible, robust, and efficient autonomy engine to handle decisions when operators cannot be active in the network. The logical description of that autonomous model and its major components is given in the AMA Data Model (ADM) [I-D.birrane-dtn-adm].

The ADM presents an efficient and expressive autonomy model for the asynchronous management of a network node, but does not specify any particular encoding. This document, the Asynchronous Management Protocol (AMP), provides a compact, binary encoding of ADM objects and specifies a protocol for the exchange of these encoded objects.

2. Requirements Language

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 [RFC2119].

3. Scope

3.1. Protocol Scope

The AMP provides data monitoring, administration, and configuration for applications operating above the data link layer of the OSI networking model. While the AMP may be configured to support the
management of network layer protocols, it also uses these protocol stacks to encapsulate and communicate its own messages.

It is assumed that the protocols used to carry AMP messages provide addressing, confidentiality, integrity, security, fragmentation, and other network/session layer functions. Therefore, these items are outside of the scope of this document.

3.2. Specification Scope

This document describes the format of messages used to exchange data models between managing and managed devices in a network. The rationale for this type of exchange is outside of the scope of this document as it is covered in [I-D.birrane-dtn-ama]. The description and explanation of the data models exchanged is also outside of the scope of this document as it is covered in [I-D.birrane-dtn-adm].

This document does not address specific data configurations of AMP-enabled devices, nor does it discuss the interface between AMP and other management protocols.

4. Terminology


5. Constraints and Assumptions

The desirable properties of an asynchronous management protocol, as specified in the AMA, are summarized here to represent design constraints on the AMP specification.

- Intelligent Push of Information - Nodes in a challenged network cannot guarantee concurrent, bi-directional communications. Some links between nodes may be strictly uni-directional. AMP should "push" data to nodes rather than "pull" data from nodes.

- Small Message Sizes - Smaller messages require smaller periods of viable transmission for communication, incur less retransmission cost, and consume fewer resources when persistently stored enroute in the network. AMP should minimize message size wherever practical, to include binary data representations and assuming pre-defined data definitions and schemas.
Absolute and Custom Data Identification - All data in the system must be uniquely addressable, to include operator-specified information. AMP should provide an efficient mechanism for encoding these unique identifiers.

Autonomous, Stateless Operation - There is no reliable concept of session establishment or round-trip data exchange in asynchronous networks. AMP should be stateless. Where helpful, AMP should provide a mechanism for transactional ordering of commands within a single AMP protocol data unit. AMP should degrade gracefully when nodes in the network fail to converge on common definitions.

Compatibility with Low-Latency Network Management Protocols - AMP encoding should be compatible with identifiers and encodings used by synchronous network management protocols. While this compatibility does not need to be embedded in the AMP encoding itself, a deterministic, mechanical translation to and from other encodings should be supported.

6. Technical Notes

Unless otherwise specified, multi-byte values in this specification are expected to be transmitted in network byte order (Big Endian).

Character encodings for all text-based data types will use UTF-8 encodings.

All encodings and AMP-specific types defined by the AMP are self-terminating. This means that, given an indefinite-length octet stream, each encoding can be unambiguously decoded from the stream without requiring additional information such as a length field separate from the data type definition.

Bit-fields in this document are specified with bit position 0 holding the least-significant bit (LSB). When illustrated in this document, the LSB appears on the right.

In order to describe the encoding of data models specified in [I-D.birrane-dtn-adm], this specification must refer to both the data object being encoded and to the encoding of that data object. When discussing the encoded version of a data object, this specification uses the notation "E(data_object)" where E() refers to a conceptual encoding function. This notation is only provided as a means of clarifying the text and imposes no changes to the actual wire coding. For example, this specification will refer to the "macro" data object as "Macro" and to the encoding of a Macro as "E(Macro)".
Illustrations of fields in this specification consist of the name of the field, the type of the field between [’s, and if the field is optional, the text "(opt)". Field order is deterministic and, therefore, fields MUST be transmitted in the order in which they are specified. In cases where an optional field is not present, then the next field will be considered for transmission. An example is shown in Figure 1 below. In this illustration two fields (Field 1 and Field 2) are shown, with Field 1 of Type 1 and Field 2 of Type 2. Field 2 is also listed as being optional. Byte fields are shown in order of receipt, from left-to-right. Therefore, when transmitted on the wire, Field 1 will be received first, followed by Field 2 (if present).

```
+----------+----------+
| Field 1  | Field 2  |
| [TYPE 1] | [TYPE 2] |
|          | (opt)    |
+----------+----------+
```

Figure 1: Byte Field Formatting Example

When types are documented in this way, the type always refers to the encoding of that type. The E() notation is not used as it is to be inferred from the context of the illustration.

7. AMP-Specific Concepts

The AMP specification provides an encoding of items comprising the asynchronous data model. As such, AMP defines very few structures of its own. This section identifies those few data structures that are unique to the AMP and required for it to perform appropriate and efficient encodings of model elements.

7.1. Nicknames (NN)

In the AMP, a "Nickname" is the compression of a shared portion of an ADM Identifier (AId). In this case, compression refers to a structured enumeration of ADMS and types within those ADMS. In this structured enumeration, each ADM is enumerated and each element in the standard ADM template is enumerated and the nickname is a number derived as a function of the two.
7.1.1. Motivation for AId Compression

AIds uniquely identify data model elements. In cases where the data model being identified is simple (such as a UINT), the size of the encoding of the AId will dominate the total size of the encoded data item. To reduce the overall size of data exchanged over AMP, the encoding of the AId must be as efficient as possible.

A potentially large source of inefficiency in the AId structure stems from the fact that data items are grouped by (1) the ADM in which they are defined and (2) the section within the ADM. In certain circumstances, groups of model elements will have AIds that have only minor differences. Consider the following set of AIDs.

DTN:adml.type1.item_1 DTN:adml.type1.item_2 ...
DTN:adml.type1.item_1974

In this example, the encoding of the "DTN:adml.type1." portion of each individual AId results in inefficient bandwidth utilization.

7.1.2. ADM Enumeration

An ADM enumeration is an unsigned integer in the range of 0 to \((2^{64})/20\). This range provides effective support for thousands of trillions of ADMs.

The formal set of ADMs, similar to SNMP MIBs and NETCONF YANG models, will be moderated and published. Additionally, a set of informal ADMs may be developed on a network-by-network or on an organization-by-organization bases.

Since informal ADMs exist within a predefined context (a network, an organization, or some other entity) they do not have individual ADM enumerations. As such, any informal ADM will be given the special enumeration "0", which will stand for "informal ADM" and will rely on other context information provided in the AId encoding (see Section 8.2.4).

Formal ADMs are presumed to be less ephemeral as they involve the time and effort and cost of moderation, publication, and maintenance. As such, a nickname will be allocated for each such formally defined ADM.

7.1.3. ADM Template Area Enumeration

An ADM Template Area Enumeration is an unsigned integer in the range of 0 – 19. This covers all of the standard areas for the ADMs.
Template as defined in [I-D.birrane-dtn-adm]. Each of these types are enumerated in Table 1.

<table>
<thead>
<tr>
<th>Area</th>
<th>Enumeration</th>
</tr>
</thead>
<tbody>
<tr>
<td>metadata</td>
<td>0</td>
</tr>
<tr>
<td>edds</td>
<td>1</td>
</tr>
<tr>
<td>tables</td>
<td>2</td>
</tr>
<tr>
<td>variables</td>
<td>3</td>
</tr>
<tr>
<td>report templates</td>
<td>4</td>
</tr>
<tr>
<td>controls</td>
<td>5</td>
</tr>
<tr>
<td>macros</td>
<td>6</td>
</tr>
<tr>
<td>operators</td>
<td>7</td>
</tr>
<tr>
<td>constants</td>
<td>8</td>
</tr>
<tr>
<td>reserved</td>
<td>9-19</td>
</tr>
</tbody>
</table>

Table 1: ADM Type Enumerations

7.1.4. Nickname Definition

As an enumeration, a Nickname is captured as a 64-bit unsigned integer (UVAST) calculated as a function of the ADM enumeration and the ADM type enumeration, as follows.

NN = ((ADM Enumeration) * 20) + (ADM Template Area Enumeration)

Considering the example set of AIds from Section 7.1.1, assuming that adm1 has ADM enumeration 9 and that typel refers to Macros and, as such, has ADM Template Area Enumeration 6, the shared nickname for the 1974 items would be: (6 * 20) + 6 = 126. In this particular example, each AId can be encoded in no more than 3 bytes: one byte to hold the nickname and up to two bytes to hold the item number (0 - 1974).
7.1.5. ADM Enumeration Considerations

The selection of formal ADM enumerations SHOULD take into consideration the nature of the applications and protocols to which the ADM applies. Those ADMs that are likely to be used in challenged networks SHOULD be allocated low enumeration numbers (e.g., those that will fit into 1-2 bytes) while ADMs that are likely to only be used in well-resourced networks SHOULD be allocated higher enumeration numbers. It SHOULD NOT be the case that ADM enumerations are allocated on a first-come, first-served basis. It is recommended that ADM enumerations should be labeled based on the number of bytes of the Nickname as a function of the size of the ADM enumeration. These labels are shown in Table 2.

<table>
<thead>
<tr>
<th>ADM Enum</th>
<th>NN Size</th>
<th>Label</th>
<th>Comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x1 - 0xCCC</td>
<td>1-2</td>
<td>Challenged</td>
<td>Constraints imposed by physical layer and power.</td>
</tr>
<tr>
<td></td>
<td>Bytes</td>
<td>Networks</td>
<td></td>
</tr>
<tr>
<td>0xCCD - 0xCCCCCCC</td>
<td>3-4</td>
<td>Congested</td>
<td>Constraints imposed by network traffic.</td>
</tr>
<tr>
<td></td>
<td>Bytes</td>
<td>Networks</td>
<td></td>
</tr>
<tr>
<td>&gt;=0xCCCCCCD</td>
<td>5-8</td>
<td>Resourced</td>
<td>Generally unconstrained networks.</td>
</tr>
<tr>
<td></td>
<td>Bytes</td>
<td>Networks</td>
<td></td>
</tr>
</tbody>
</table>

Table 2: ADM Enumerations Labels

8. Encodings

This section describes the binary encoding of logical data constructs using the Concise Binary Object Representation (CBOR).

8.1. CBOR Considerations

The following considerations act as guidance for CBOR encoders and decoders implementing the AMP.

- All AMP encodings are of definite length and, therefore, indefinite encodings MUST NOT be used.

- AMP encodings MUST NOT use CBOR tags. Identification mechanisms in the AMP capture structure and other information such that tags are not necessary.
Canonical CBOR MUST be used for all encoding. All AMP CBOR decoders MUST run in strict mode.

Encodings MUST result in smallest data representations. There are several cases where the ADM defines types with less granularity than CBOR. For example, ADM defines the UINT type to represent unsigned integers up to 32 bits in length. CBOR supports separate definitions of unsigned integers of 8, 16, or 32 bits in length. In cases where an ADM type MAY be encoded in multiple ways in CBOR, the smallest data representation MUST be used. For example, UINT values of 0-255 MUST be encoded as a uint8_t, and so on.

8.2. AMM Models

8.2.1. Primitive and Derived Types

The AMP supports types for unsigned bytes, 32/64-bit signed and unsigned integers, 32/64-bit floating point values, and strings, as outlined in Table 3.
<table>
<thead>
<tr>
<th>AMP Type</th>
<th>CBOR Major Type</th>
<th>CBOR Additional Value</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>BYTE</td>
<td>7 (Primitive)</td>
<td>0-23 or 24</td>
<td>BYTES are simple values in CBOR.</td>
</tr>
<tr>
<td>INT</td>
<td>0 (Positive), 1 (Negative)</td>
<td>0-23,24,25,26</td>
<td>INTs are encoded as positive or negative integers from (\text{(u)}\text{int8}_t) up to (\text{(u)}\text{int32}_t).</td>
</tr>
<tr>
<td>UINT</td>
<td>0</td>
<td>0-23, 24, 25, 26</td>
<td>UINTs are positive integers from (\text{uint8}_t) up to (\text{uint32}_t).</td>
</tr>
<tr>
<td>VAST</td>
<td>0 (Positive), 1 (Negative)</td>
<td>27</td>
<td>VASTs are encoding as (\text{(u)}\text{int64}_t).</td>
</tr>
<tr>
<td>UVAST</td>
<td>0</td>
<td>27</td>
<td>VASTs are encoding as (\text{uint64}_t).</td>
</tr>
<tr>
<td>REAL32</td>
<td>7 (Primitive)</td>
<td>26</td>
<td>Single-precision, 32-bit floating point value in IEEE-754 format.</td>
</tr>
<tr>
<td>REAL64</td>
<td>7 (Primitive)</td>
<td>27</td>
<td>Double-precision, 64-bit floating point value in IEEE-754 format.</td>
</tr>
<tr>
<td>STR</td>
<td>3 (Text String)</td>
<td>Varies</td>
<td>Uses CBOR encoding unmodified.</td>
</tr>
<tr>
<td>BOOL</td>
<td>7 (Primitive)</td>
<td>0 or 1</td>
<td>AV of 0 is considered FALSE. AV of 1 considered TRUE.</td>
</tr>
<tr>
<td>TS</td>
<td>0 (Positive)</td>
<td>Varies</td>
<td>Timestamp in the smallest of (\text{uint8}_t), (\text{uint16}_t), (\text{uint32}_t), or (\text{uint64}_t).</td>
</tr>
<tr>
<td>BLOB</td>
<td>2 (Byte String)</td>
<td>Varies</td>
<td>Uses CBOR encoding unmodified.</td>
</tr>
</tbody>
</table>

Table 3: Standard Numeric Types
8.2.2. Derived Types

The AMP supports types for unsigned bytes, 32/64-bit signed and unsigned integers, 32/64-bit floating point values, and strings, as outlined in Table 4.

<table>
<thead>
<tr>
<th>AMP Type</th>
<th>CBOR Major Type</th>
<th>CBOR Additional Value</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>TS</td>
<td>7 (Primitive)</td>
<td>0-23 or 24</td>
<td>BYTES are simple values in CBOR.</td>
</tr>
</tbody>
</table>

Table 4: Standard Numeric Types

8.2.3. Type-Name-Value (TNV)

TNV values are encoded as a CBOR array that comprises four distinct pieces of information: a set of flags, a type, an optional name, and an optional value. In the E(TNV) the flag and type information are compressed into a single value. The CBOR array MUST have length 2, 3, or 4 depending on the number of optional fields appearing in the encoding. The E(TNV) format is illustrated in Figure 2.

```
+---------+
|   TNV   |
| [ARRAY] |
+---------+

Array Header Flags/Type Name Value
[UINT]   [UINT] [TXT STR] [Varies]
(opt)     (opt)
```

Figure 2: E(TNV) Format

The E(TNV) fields are defined as follows.

Flags/Type

The first byte of the E(TNV) describes the type associated with the TNV and which optional components are present. The layout of this byte is illustrated in Figure 3.
E(TNV) Flag/Type Byte Format

+------+-------+-------------+
| Name | Value | Struct      |
| Flag | Flag  |    Type     |
+------+-------+-------------+
|  7   |   6   | 5 4 3 2 1 0 |
+------+-------+-------------+

Figure 3

Name Flag
This flag indicates that the TNV contains a name field. When set to 1 the Name field MUST be present in the E(TNV). When set to 0 the Name field MUST NOT be present in the E(TNV).

Value Flag
This flag indicates that the TNV contains a value field. When set to 1 the Value field MUST be present in the E(TNV). When set to 0 the Value field MUST NOT be present in the E(TNV).

Struct Type
This field lists the type associated with this TNV and MUST contain one of the types defined in [I-D.birrane-dtn-adm] with the exception that the type of a TNV MUST NOT be a TNV.

Name
This optional field captures the human-readable name for the TNV encoded as a CBOR text string (major type 3).

Value
This optional field captures the encoded value associated with this TNV. The value is encoded in accordance with AMP rules for encoding of items of the type of this TNV.

8.2.4. ADM Identifier (AId)

The AId, as defined in [I-D.birrane-dtn-adm], identifies an ADM object. There are two kinds of ADM objects that can be identified in this scheme: literal objects (of ADM type LITERAL) and all other ADM objects.
8.2.4.1. Encoding ADM Objects of Type LITERAL

A literal identifier is one that is literally defined by its value, such as the numbers (0, 3.14) and strings ("example"). AIds of type LITERAL do not have issuers or nicknames or parameters. They are simply typed basic values.

The E(AId) of an ADM of type LITERAL is encoded as a CBOR Byte String and consists of a mandatory flag BYTE and the value of the LITERAL.

The E(AId) structure for LITERALS is illustrated in Figure 4.

```
+--------+----------+
| Flags  | Value    |
| [BYTE] | [VARIES] |
+--------+----------+
```

Figure 4

These fields are defined as follows.

**Flags**

The Flags byte identifies the ADM object as being of type LITERAL and also captures the primitive type of the following value. The layout of this byte is illustrated in Figure 5.

```
+------------+-------------+
| VALUE TYPE | STRUCT TYPE |
| 7 6 5 4    | 3 2 1 0     |
+------------+-------------+
```

Figure 5

**Value Type**

The high nibble of the flag byte describes the type of the value of the AId being encoded. This type MUST be one of the ADM structure types defined in [I-D.birrane-dtn-adm]. as a "Primitive Type".

**Structure Type**
The lower nibble of the flag byte identifies the type of ADM Object being identified by the AId. In this instance, this value MUST be LITERAL, as defined in [I-D.birrane-dtn-adm].

Value

This field captures the CBOR encoding of the value. Values are encoded according to their Value Type as specified in the flag byte in accordance with the encoding rules provided in Section 8.2.1.

8.2.4.2. Encoding Non-Literal ADM Objects

All other AIds are defined in the context of ADM structures, templates, issuing organizations and may contain parameters and other meta-data. The AMP, as a binary encoding of this information meant to exchange ADM models in a machine-to-machine context removes human-readable information such as Name and Description. Additionally, this encoding adds other information to improve the efficiency of the AId encoding, such as the concept of Nicknames as defined in Section 7.1.

The E(AId) is encoded as a CBOR Byte String and consists of a mandatory flag BYTE, a mandatory ADM Resource Identifier (ARI), and optional annotations to assist with filtering, access control, and parameterization. The E(AId) structure is illustrated in Figure 6.

---

These fields are defined as follows.

Flags

Flags describe the type of structure and which optional fields are present in the encoding. The layout of the flag byte is illustrated in Figure 7.
E(AId) General Flag Byte Format

<table>
<thead>
<tr>
<th>NN</th>
<th>PARM</th>
<th>ISS</th>
<th>TAG</th>
<th>STRUCT TYPE</th>
</tr>
</thead>
<tbody>
<tr>
<td>7</td>
<td>6</td>
<td>5</td>
<td>4</td>
<td>3  2  1  0</td>
</tr>
</tbody>
</table>

Figure 7

Nickname (NN)
This flag indicates that ARI compression is used for this E(AId). When set to 1 the Nickname field MUST be present in the E(AId). When set to 0 the Nickname field MUST NOT be present in the E(AId). When an AId is user-defined, there are no semantics for Nicknames and, therefore, this field MUST be 0 when the Issuer flag is set to 1. Implementations SHOULD use Nicknames whenever possible to reduce the size of the E(AId).

Parameters Present (PARM)
This flag indicates that this AId can be parameterized and that parameter information is included in the E(AId). When set to 1 the Parms field MUST be present in the E(AId). When set to 0 the Parms field MUST NOT be present in the E(AId).

Issuer Present (ISS)
This flag indicates that this AId is defined in the context of a specific issuing entity. When set to 1 the Issuer field MUST be present in the E(AId). When set to 0 the Issuer field MUST NOT be present in the E(AId).

Tag Present (TAG)
This flag indicates that the AId is defined in the context of a specific issuing entity and that issuing entity adds additional information in the form of a tag. When set to 1 the Tag field MUST be present in the E(AId). When set to 0 the Tag field MUST NOT be present in the E(AId). This flag MUST be set to 0 if the Issuer Present flag is set to 0.

Structure Type (STRUCT TYPE)
The lower nibble of the E(AId) flag byte identifies the kind of data structure being identified. This
field MUST contain one of the ADM structure types defined in [I-D.birrane-dtn-adm].

Nickname (NN)
This optional field contains the Nickname as calculated according to Section 8.2.4.

ADM Resource Identifier (ARI)
This mandatory field contains the enumeration of the ADM object. For elements defined in an ADM Template (e.g., where the Issuer Flag is set to 0) this is the 0-based index into the ADM Template array holding this element. For all user-defined ADM objects, (e.g., where the Issuer Flag is set to 1) this value is as defined by the Issuing organization.

Parameters
The parameters field is represented as a Types Then Value Collection (TTVC) as defined in Section 8.3.2. The overall number of items in the collection represents the number of parameters. The types of the TTVC represent the types of each parameter, with the first listed type associated with the first parameter, and so on. The values, if present, represent the values of the parameters, with the first listed value being the value of the first parameter, and so on.

Issuer
This is a binary identifier representing a predetermined issuer name. The AMP protocol does not parse or validate this identifier, using it only as a distinguishing bit pattern to ensure uniqueness. This value, for example, may come from a global registry of organizations, an issuing node address, or some other network-unique marking. The issuer field MUST NOT be present for any AId defined as part of an ADM.

Tag
A value used to disambiguate multiple AIds with the same Issuer. The definition of the tag is left to the discretion of the Issuer. The Tag field MUST be present if the Tag Flag is set in the flag byte and MUST NOT be present otherwise.

8.3. Collections

8.3.1. TNV Collection
A TNV Collection (TNVC) is a series of multiple TNV values. This is simply encoded as a CBOR array with each element in the array represented by the encoding of a TNV in accordance with Section 8.2.3
8.3.2. Types Then Value Collection (TTVC)

A Types-Then-Value Collection (TTVC) provides a mechanism for communicating a typed set of values by separating the types from the values themselves. This construction is useful both for rapidly performing type verification and for efficiently omitted type information where appropriate.

Extracting type information to the "front" of the collection optimizes the performance of type validators. A validator can inspect the first array to ensure that element values match type expectations. If type information were distributed throughout the collection, as in the case with the TNVC, a type validator would need to scan through the entire set of data to validate each type in the collection. A TTVC SHOULD be used in lieu of a TNVC whenever type validation must be performed.

In certain circumstances, a set of values can be communicated without any type information when type information can be inferred from context. In these circumstances, separating types from values allows for an efficient way to omit type information when necessary.

The TTVC is encoded as a CBOR array with either one or two elements. If the array has 1 element, then it MUST be a CBOR array of values. If the TTVC array has 2 elements, then it MUST contain a CBOR byte string of type information followed by a CBOR array of values. In cases where both types and values are present, the number of types MUST be the same as the number of elements as the array of values. The order of types MUST correspond to the order of values in the second array.

The E(TTVC) format is illustrated in Figure 8.
These fields are defined as follows.

**Types**

The Types field is encoded as a CBOR byte string with each element of the array encoded as a byte. Each byte represents a type and MUST match a type enumeration as defined in [I-D.birrane-dtn-adm].

**Values**

The Values array is encoded as a CBOR array with each element of the array encoded as a CBOR byte string containing the CBOR encoding of the value, based on its type as required by this specification.

### 8.3.3. AId Collections (AC)

An AId collection is an ordered collection of AId values. It is encoded as a CBOR array with each element being an encoded AId, as illustrated in Figure 9.
8.3.4. Expressions (EXPR)

The Expression object encapsulates a typed postfix expression in which each operator MUST be of type OP and each operand MUST be the typed result of an operator or one of EDD, VAR, LIT, or CONST.

The Expression object is encoded as a CBOR byte array whose format is illustrated in Figure 10.

E(AC) Format

```
+--------+------------+
|  Type  | Expression |
| [BYTE] |    [AC]    |
+--------+------------+
```

Figure 10

Type

The enumeration representing the type of the result of the evaluated expression. This type MUST be defined in [I-D.birrane-dtn-adm] as a "Primitive Type".

Expression

An expression is represented in the AMP as an AIId collection, where each AIId in the ordered collection represents either an operand or operator in postfix form.

8.3.5. Predicate (PRED)

Predicates are Expressions whose values are interpreted as a Boolean. The value of zero MUST be considered "false" and all other values MUST be considered "true". Otherwise, Predicates are encoded and treated the same as expressions.

8.4. ADM Object Definition Encodings

The autonomy model codified in [I-D.birrane-dtn-adm] comprises multiple individual objects. This section describes the CBOR encoding of these objects.

Note: The encoding of an object refers to the way in which the complete object can be encoded such that the object as it exists on a Manager may be re-created on an Agent, and vice-versa. In cases where both a Manager and an Agent already have the definition of an object, then only the encoded AIId of the object needs to be communicated. This is the case for all objects defined in a
published ADM and any user-defined object that has been synchronized between an Agent and Manager.

8.4.1. Externally Defined Data (EDD)

Externally defined data (EDD) are solely defined in the ADMs for various applications and protocols. EDDs represent values that are calculated external to an AMA Agent, such as values measured by firmware.

The representation of these data is simply their identifying AIds. The representation of an EDD is illustrated in Figure 11.

\[ E(EDD) \text{ Format} \]

\[ +--------+ \]
\[ | \quad \quad | \]
\[ | \quad \text{ID} \quad | \]
\[ | \quad \text{[AId]} \quad | \]
\[ +--------+ \]

Figure 11

ID

This is the AId identifying the EDD. Since EDDs are always defined solely in the context of an ADM, this AId MUST NOT have an ISSUER field and MUST NOT have a TAG field. This AId MAY be defined with parameters.

8.4.2. Constants (CONST)

Unlike Literals, a Constant is an immutable, typed, named value. Examples of constants include PI to some number of digits or the UNIX Epoch.

Since ADM definitions are preconfigured on Agents and Managers in an AMA, the type information for a given Constant is known by all actors in the system and the encoding of the Constant needs to only be the name of the constant as the Manager and Agent can derive the type and value from the unique Constant name.

The format of a Constant is illustrated in Figure 12.
8.4.3. Controls (CTRL)

A Control represents a pre-defined and optionally parameterized opcode that can be run on an Agent. Controls in the AMP are always defined in the context of an AMA; there is no concept of an operator-defined Control. Since Controls are pre-configured in Agents and Managers as part of ADM support, their representation is simply the AId that identifies them, similar to EDDs.

The format of a Control is illustrated in Figure 13.

E(CTRL) Format

+-------+
| ID    |
| [AId] |
+-------+

Figure 13

ID
This is the AId identifying the Control. This AId MUST NOT have an ISSUER field and MUST NOT have a TAG field. This AId MAY have parameters.

8.4.4. Macros (MAC)

Macros in the AMP are ordered collections of AId (an AC) that contain Controls or other Macros. When run by an Agent, each AId in the AC MUST be run in order.

Any AMP implementation MUST allow at least 4 levels of Macro nesting. Implementations MUST prevent recursive nesting of Macros.
The AId associated with a Macro MAY contain parameters. Each parameter present in a Macro AId MUST contain type, name, and value information. Any Control or Macro encapsulated within a parameterized Macro MAY also contain parameters. If an encapsulated object parameter contains only name information, then the parameter value MUST be taken from the named parameter provided by the encapsulating Macro. Otherwise, the value provided to the object MUST be used instead.

The format of a Macro is illustrated in Figure 14.

```
E(MACRO) Format

+-------+------------+
|  ID   | Definition |
| [AId] |    [AC]    |
+-------+------------+
```

Figure 14

ID

This is the AId identifying the Macro. When a Macro is defined in an ADM this AId MUST NOT have an ISSUER field and MUST NOT have a TAG field. When the Macro is defined outside of an ADM, the AId MUST have an ISSUER field and MAY have a TAG field.

Definition

This is the ordered collection of AIds that identify the Controls and other Macros that should be run as part of running this Macro.

8.4.5. Operators (OP)

Operators are always defined in the context of an ADM. There is no concept of a user-defined operator, as operators represent mathematical functions implemented by the firmware on an Agent. Since Operators are pre-configured in Agents and Managers as part of ADM support, their representation is simply the AId that identifies them.

The ADM definition of an Operator MUST specify how many parameters are expected and the expected type of each parameter. For example, the unary NOT Operator ("!") would accept one parameter. The binary PLUS Operator ("+")) would accept two parameters. A custom function to calculate the average of the last 10 samples of a data item should accept 10 parameters.
Operators are always evaluated in the context of an Expression. The encoding of an Operator is illustrated in Figure 15.

E(OP) Format

+-------+
| ID    |
| [AId] |
+-------+

Figure 15

ID

This is the AId identifying the Operator. Since Operators are always defined solely in the context of an ADM, this AId MUST NOT have an ISSUER field and MUST NOT have a TAG field.

8.4.6. Report Templates (RPTT)

A Report Template is an ordered collection of identifiers that describe the order and format of data in any Report Entry built in compliance with the template. A template is a schema for a class of reports. They contain no actual values and may be defined in a formal ADM or configured by users in the context of a network deployment.

A Report Template is modeled as an AC, as each data definition in the template is identified by an AId.

E(RPTT) Format

+-----------------+
| Report Contents |
| [AC]            |
+-----------------+

Figure 16

8.4.7. Report (RPT)

A Report is a set of data values populated using a given Report Template. While Reports do not contain name information, they MAY contain type information in cases where recipients wish to perform type validation prior to interpreting the Report contents in the context of a Report Template. Reports are "anonymous" in the sense that any individual Report does not contain a unique identifier. Reports can be differentiated by examining the combination of (1) the
A Report object is comprised of the identifier of the template used to populate the report, an optional timestamp, and the contents of the report. A Report is encoded as a CBOR array with either 2 or 3 elements. If the array has 2 elements then the optional Timestamp MUST NOT be in the Report encoding. If the array has 3 elements then the optional timestamp MUST be included in the Report encoding. The Report encoding is illustrated in Figure 17.

\[
\begin{array}{ccc}
\text{Template} & \text{Timestamp} & \text{Entries} \\
\text{[AID]} & \text{[TS]} & \text{[TTVC]} \\
\text{(opt)} & & \\
\end{array}
\]

Figure 17

Template

This is the AId identifying the template used to interpret the data in this report.

This AId may be parameterized and, if so, the parameters MUST include a name field and have been passed-by-name to the template contents when constructing the report.

Timestamp

The timestamp marks the time at which the report was created. This timestamp may be omitted in cases where the time of the report generation can be inferred from other information. For example, if a report is included in a message group such that the timestamp of the message group is equivalent to the timestamp of the report, the report timestamp may be omitted and the timestamp of the included message group used instead.

Values

This is the collection of data values that comprise the report contents in accordance with the associated Report Template.

8.4.8. State-Based Rules (SRL)

A State-Based Rule (SRL) specifies that a particular action should be taken by an Agent based on some evaluation of the internal state of the Agent. A SRL specifies that starting at a particular START time
an ACTION should be run by the Agent if some CONDITION evaluates to true, until the ACTION has been run COUNT times. When the SRL is no longer valid it may be discarded by the agent.

Examples of SRLs include:

Starting 2 hours from receipt, whenever V1 > 10, produce a Report Entry for Report Template R1 no more than 20 times.

Starting at some future absolute time, whenever V2 != V4, run Macro M1 no more than 36 times.

An SRL object is encoded as a CBOR array with 5 elements as illustrated in Figure 18.

![Figure 18](image)

ID
This is the AId identifying the SRL. If this AId contains parameters they MUST include a name in support of pass-by-name to each element of the Action AC.

START
The time at which the SRL condition should start to be evaluated. This will mark the first evaluation of the condition associated with the SRL.

CONDITION
The Predicate which, if true, results in the SRL running the associated action.
The number of times the SRL action can be run. The special value of 0 indicates there is no limit on how many times the action can be run.

**ACTION**

The collection of Controls and/or Macros to run as part of the action. This is encoded as an AId Collection in accordance with [Section 8.3.3](#) with the stipulation that every AId in this collection MUST represent either a Control or a Macro.

### 8.4.9. Table Templates (TBLT)

A Table Template (TBLT) describes the types, and optionally names, of the columns that define a Table.

The TBLT Object is encoded as a CBOR array that MUST contain either 2 or 3 elements. If the array is of size 2, then the column names array MUST NOT be present in the encoding. If the array is of size 3 then the column names array MUST be present in the encoding. The format of the TBLT Object Array is illustrated in Figure 19.

**E(TBLT) Format**

```plaintext
+---------+
| TBLT    |
| [ARRAY] |
+---------+

/

+-------------------------++-------------------------+-------------------------+
| ID [AID] | Types [BYTE STR] | Names [ARRAY] (opt) |
+-------------------------++-------------------------+-------------------------+
```

**Figure 19**

The elements of the TBLT array are defined as follows.

**ID**

This is the AId of the table template encoded in accordance with [Section 8.2.4](#).

**Types**
The types field captures the data types of each column comprising the table template. Each type is represented by its enumeration as defined in [I-D.birrane-dtn-adm]. This field is encoded as a CBOR byte string with the length of the string equal to the number of columns in the template and each column type enumeration encoded as a byte. The Nth byte in the byte string represents the type of the Nth column.

Names

When present, column names are encoded as a CBOR array. This array MUST have the same number of elements as bytes in the Types byte string. Each element in the Names array is encoded as a CBOR text string and represents the string representation of the column name. The Nth text string in the array represents the name of the Nth column.

8.4.10. Tables (TBL)

A Table object describes the series of values associated with a Table Template.

A Table object is encoded as a CBOR array, with the first element of the array identifying the Table Template and each subsequent element identifying a row in the table. The format of the TBL Object Array is illustrated in Figure 20.

```
+---------+--------+     +--------+
| TBLT ID |  Row 1 |     |  Row N |
|  [AID]  | [TTVC] | ... | [TTVC] |
+---------+--------+     +--------+
```

Figure 20

The TBL fields are defined as follows.

Template ID (TBLT ID)

This is the AId of the table template describing the format of the table and is encoded in accordance with Section 8.2.4.
Row

Each row of the table is represented as a series of values with optional type information to aid in type checking table contents to column types. Each row is encoded as a TTVC and MAY include type information. AMP implementations should consider the impact of including type information for every row on the overall size of the encoded table. Each TTVC representing a row MUST contain the same number of elements as there are columns in the referenced Table Template.

8.4.11. Time-Based Rules (TRL)

A Time-Based Rule (TRL) specifies that a particular action should be taken by an Agent based on some time interval. A TRL specifies that starting at a particular START time, and for every PERIOD seconds thereafter, an ACTION should be run by the Agent until the ACTION has been run for COUNT times. When the TRL is no longer valid it MAY BE discarded by the Agent.

Examples of TRLs include:

Starting 2 hours from receipt, produce a Report Entry for Report Template R1 every 10 hours ending after 20 times.

Starting at the given absolute time, run Macro M1 every 24 hours ending after 365 times.

The TRL object is encoded as a CBOR array with 5 elements as illustrated in Figure 21.

E(TRL) Format

/+--------+
|        |
| TRL    |
| [ARRAY]|
/+--------+
|        |

/+----------------------------------------/+----------------------------------------/  +----------------------------------------/+----------------------------------------/  +----------------------------------------/+----------------------------------------/  +----------------------------------------/+----------------------------------------/  +----------------------------------------/+----------------------------------------/
| ID   | START | PERIOD | COUNT | ACTION |
| [AID] | [TS]  | [UINT] | [UINT] | [AC]   |

Figure 21
ID
This is the AId identifying the TRL and is encoded in accordance with Section 8.2.4. If this AId contains parameters they MUST include a name in support of pass-by-name to each element of the Action AC.

START
The time at which the TRL condition should start to be evaluated.

PERIOD
The number of seconds to wait between running the action associated with the TRL.

COUNT
The number of times the TRL action can be run. The special value of 0 indicates there is no limit on how many times the action can be run.

ACTION
The collection of Controls and/or Macros to run as part of the action. This is encoded as an AId Collection in accordance with Section 8.3.3 with the stipulation that every AId in this collection MUST represent either a Control or a Macro.

8.4.12. Variables (VAR)

Variable objects are transmitted in the AMP without the human-readable description.

Variable objects are encoded as a CBOR byte string whose format is illustrated in Figure 22.

This section describes the format of the messages that comprise the AMP protocol.

9.1. AMP Message Summary

The AMP message specification is limited to three basic communications:

Figure 22

- **ID**: This is the AId identifying the VAR and is encoded in accordance with Section 8.2.4. This AId MUST NOT include parameters.

- **Type**: This field captures the data type of the Variable encoded as a BYTE. This data type MUST be one of the data types defined in [I-D.birrane-dtn-adm]. It is possible to specify a type different than the resultant type of the initializing Expression. For example, if an Expression adds two single-precision floating point numbers, the VAR MAY have an integer type associated with it. In cases where this is the case, numeric conversions MUST be handled in accordance with [I-D.birrane-dtn-adm].

- **Initializer**: The initial value of the Variable is calculated by an Expression encoded in accordance with Section 8.3.4.
Table 5: ADM Message Type Enumerations

The entire management of a network can be performed using these three messages and the configurations from associated ADMs.

9.2. Message Group Format

Individual messages within the AMP are combined into a single group for communication with another AMP Actor. Messages within a group MUST be received and applied as an atomic unit. The format of a message group is illustrated in Figure 23. These message groups are assumed communicated amongst Agents and Managers as the payloads of encapsulating protocols which should provide additional security and data integrity features as needed.

A message group is encoded as a CBOR array with at least 2 elements, the first being the time the group was created followed by 1 or more messages that comprise the group. The format of the message group is illustrated in Figure 23.

AMP Message Group Format

```
+-------------+
| Message Group |
| [ARRAY]    |
+-------------+

/                                           \
+-----------++------------++------------+
| Timestamp | Message 1 | ... | Message N |
| [TS]      | [BYTE STR]|     | [BYTE STR]|
+-----------++------------++------------+

Figure 23
```
9.3. Message Format

Each message identified in the AMP specification adheres to a common message format, illustrated in Figure 24, consisting of a message header, a message body, and an optional trailer.

Each message in the AMP is encoded as a CBOR byte string formatted in accordance with Figure 24.

AMP Message Format

```
+--------+----------+----------+
| Header |   Body   | Trailer  |
| [BYTE] | [VARIES]| [VARIES] |
|        |          |  (opt.)  |
+--------+----------+----------+
```

Figure 24

Header

The message header BYTE is shown in Figure 25. The header identifies a message context and opcode as well as flags that control whether a Report Entry should be generated on message success (Ack) and whether a Report Entry should be generated on message failure (Nack).

AMP Common Message Header

```
+----------+-----+------+-----+----------+
| Reserved | ACL | Nack | Ack | Opcode   |
+----------+-----+------+-----+----------+
|  7   6   |  5  |   4  |  3  |  2  1  0 |
+----------+-----+------+-----+----------+
```

Figure 25
 Opcode
The opcode field identifies which AMP message is being represented.

 ACK Flag
The ACK flag describes whether successful application of the message must generate an acknowledgement back to the message sender. If this flag is set (1) then the receiving actor MUST generate a Report Entry communicating this status. Otherwise, the actor MAY generate such a Report Entry based on other criteria.

 NACK Flag
The NACK flag describes whether a failure applying the message must generate an error notice back to the message sender. If this flag is set (1) then the receiving Actor MUST generate a Report Entry communicating this status. Otherwise, the Actor MAY generate such a Report Entry based on other criteria.

 ACL Used Flag
The ACL used flag indicates whether the message has a trailer associated with it that specifies the list of AMP actors that may participate in the Actions or definitions associated with the message. This area is still under development.

 Body
The message body contains the information associated with the given message.

 Trailer
An OPTIONAL access control list (ACL) may be appended as a trailer to a message. When present, the ACL for a message identifies the agents and managers that can be affected by the definitions and actions contained within the message. The explicit impact of an ACL is described in the context of each message below. When an ACL trailer is not present, the message results may be visible to any AMP Actor in the network, pursuant to other security protocol implementations.

9.4. Register Agent

The Register Agent message is used to inform an AMP Manager of the presence of another Agent in the network.

The body of this message is the name of the new agent, encoded as illustrated in Figure 26.
Register Agent Message Body

+------------+
| Agent ID   |
| [BYTE STR] |
+------------+

Figure 26

Agent ID

The Agent ID MUST represent the unique address of the Agent in whatever protocol is used to communicate with the Agent.

9.5. Report Set

The Report Set message contains a set of 1 or more Reports produced by an AMP Agent and sent to an AMP Manager.

The body of this message contains information on the recipient of the reports followed by one or more Reports. The body is encoded as illustrated in Figure 27.

Report Set Message Body

+----------+---------+     +---------+
| RX Names | ENTRY 1 |     | ENTRY N |
| [ARRAY]  | [RPT]   | ... | [RPT]   |
+----------+---------+     +---------+

Figure 27

RX Names

This field captures the set of Managers that have been sent this report set. This is encoded as a CBOR array that MUST have at least one entry. Each entry in this array is encoded as a CBOR text string.

ENTRY N

The Nth Report encoded in accordance with Section 8.4.7.

9.6. Perform Control

The perform control message causes the receiving AMP Actor to run one or more pre-configured Controls provided in the message.

The body of this message is the start time for the controls followed by the controls themselves, as illustrated in Figure 28.
Perform Control Message Body

+-------+-----------+
| Start |  Controls |
| [TS]  |    [AC]   |
+-------+-----------+

Figure 28

Start
The time at which the Controls/Macros should be run.

Controls
The collection of AIds that represent the Controls and/or Macros to be run by the AMP Actor. Every AId in this collection MUST be either a Control or a Macro.

10. IANA Considerations

A Nickname registry needs to be established.

11. Security Considerations

Security within the AMP exists in two layers: transport layer security and access control.

Transport-layer security addresses the questions of authentication, integrity, and confidentiality associated with the transport of messages between and amongst Managers and Agents. This security is applied before any particular Actor in the system receives data and, therefore, is outside of the scope of this document.

Finer grain application security is done via ACLs provided in the AMP message headers.

12. References

12.1. Informative References

[I-D.birrane-dtn-ama]
Birrane, E., "Asynchronous Management Architecture",
draft-birrane-dtn-ama-06 (work in progress), October 2017.

12.2. Normative References
Appendix A.  Acknowledgements

The following participants contributed technical material, use cases, and useful thoughts on the overall approach to this protocol specification: Jeremy Pierce-Mayer of INSYEN AG contributed the concept of the typed data collection and early type checking in the protocol.

Author’s Address

Edward J. Birrane
Johns Hopkins Applied Physics Laboratory

Email: Edward.Birrane@jhuapl.edu