Delay-Tolerant Networking
Internet-Draft
Intended status: Experimental
Expires: February 18, 2016

E. Birrane
Johns Hopkins Applied Physics Laboratory

August 17, 2015

Asynchronous Management Protocol
draft-birrane-dtn-amp-00

Abstract

This document describes an Asynchronous Management Protocol (AMP) conformant with an Asynchronous Management Architecture (AMA). The AMP provides monitoring and configuration services between managing devices (Managers) and managed devices (Agents), some of which may operate on the far side of high-delay or high-disruption links. The AMP minimizes 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. The AMP accomplishes this without requiring mobile code and generally reduces the processor, memory, and storage requirements of implementing Managers and Agents.

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 http://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 February 18, 2016.

Copyright Notice

Copyright (c) 2015 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 (http://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
1.1. Overview 3
1.2. Technical Notes 5
1.3. Scope 5
1.3.1. Protocol Scope 5
1.3.2. Specification Scope 6
1.4. Requirements Language 6
2. Terminology 6
3. Data Model 7
3.1. Basic Types 7
3.1.1. Standard Numeric Types 7
3.1.2. Self-Delimiting Numeric Value (SDNV) 8
3.1.3. Timestamp (TS) 8
3.2. Compound Types 9
3.2.1. String (STR) 9
3.2.2. Binary Large Object (BLOB) 9
3.2.3. Data Collection (DC) 9
3.2.4. Typed Data Collection (TDC) 10
3.3. Naming Structures 11
3.4. Special Types 15
3.4.1. MID Collections (MC) 16
3.4.2. Expressions (EXPR) 16
3.4.3. Predicate (PRED) 16
3.4.4. Definition (DEF) 16
4. AMP Structures 17
4.1. AMA Overview 17
4.2. Primitive Data 18
4.2.1. Definition 19
4.2.2. Processing 19
4.3. Computed Data 19
4.3.1. Definition 20
4.3.2. Processing 21
4.4. Report 21
4.4.1. Definition 21
4.4.2. Processing 22
4.5. Control 23
4.5.1. Definition 23
4.5.2. Processing 24
4.6. Time-Based Rule (TRL) 24

4.6.1. Definition 25
4.6.2. Processing 25
4.7. State-Based Rule (SRL) 26
4.7.1. Definition 26
4.7.2. Processing 27
4.8. Macro 28
4.8.1. Definition 28
4.8.2. Processing 29
4.9. Literal 29
4.9.1. Definition 30
4.9.2. Processing 30
4.10. Operator 30
4.10.1. Definition 31
4.10.2. Processing 31
5. Data Type IDs and Enumerations 31
6. Application Data Model Template 33
6.1. Overview 33
6.2. Template 33
6.2.1. ADM Metadata 33
6.2.2. ADM Information Capture 34
7. The Agent ADM 35
8. Functional Specification 36
8.1. Message Group Format 36
8.2. Message Format 36
8.3. Register Agent (0x00) 38
8.4. Data Report (0x12) 38
8.5. Perform Control (0x20) 39
9. IANA Considerations 39
10. Security Considerations 39
11. References 40
11.1. Informative References 40
11.2. Normative References 40
Author's Address 40

1. Introduction

This document specifies an Asynchronous Management Protocol (AMP) that provides application-layer network management services over links where propagation delays or link disruptions prevent timely communications between a network operator and a managed device. The AMP is conformant to the Asynchronous Management Architecture [AMA].

1.1. Overview

Network management protocols define the messages that implement management functions amongst managed and managing devices in a network. These functions include the definition, production, and reporting of performance data, the application of administrative

policy, and the configuration of behavior based on local time and state measurements.

Networks whose communication links are frequently challenged by physical or administrative effects may not provide the guarantee of low-latency, duplex data communications necessary to support sessions and other synchronous communication. For such networks, an asynchronous management protocol is required which provides familiar network management services in the absence of sessions and operator-in-the-loop control.

AMP accomplishes the network management function using open-loop, intelligent-push, asynchronous mechanisms that better scale as link challenges scale. The protocol is designed to support several desirable properties outlined in [AMA] and briefly listed below.

- o Intelligent Push of Information - The intelligent push of information eliminates the need for round-trip data exchange. This is a necessary consequence of operating in an open-loop system. AMP is designed to operate even in networks of solely uni-directional links.

- o Small Message Sizes - Smaller messages require smaller periods of viable transmission for communication, incur less re-transmission cost, and consume less resources when persistently stored en-route in the network. AMP minimizes the size of a message whenever practical, to include packing and unpacking binary data, variable-length fields, and pre-configured data definitions.

- o Fine-grained, Flexible Data Identification - Fine-grained identification allows data in the system to be explicitly addressed while flexible data identification allows users to define their own customized, addressed data collections. In both cases, the ability to define precisely the data required removes the need to query and transmit large data sets only to filter/ downselect desired data at a receiving device.

- o Stateless Operation - AMP does not rely on session establishment or round-trip data exchange to perform network management functions. Wherever possible, the AMP is designed to be stateless. Where state is required, the AMP provides mechanisms to support transactions and graceful degradation when nodes in the network fail to synchronize on common definitions.

- o Compatibility with Low-Latency Network Management Protocols - AMP adopts an identifier approach compatible with the Managed Information Base (MIB) format used by Internet management protocols such as the Simple Network Management Protocol (SNMP),

thus enabling management interfaces between challenged networks and unchallenged networks (such as the Internet).

1.2. Technical Notes

- o Multi-byte values presented in this specification are to be transmitted in network-byte order.

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

- o Illustrations of byte fields in this specification consist of the name of the field, the type of the fields between []'s, and if the field is optional, the text "(opt)". 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 appear first, followed by Field 2 (if present).

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

Figure 1: Byte Field Formatting Example

1.3. Scope

1.3.1. 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 support and other network/session layer functions. Therefore, these items are not discussed in the scope of this document.

1.3.2. Specification Scope

This document describes the format of the AMP messages exchanged amongst managing and managed devices in a challenged network. This document further describes the rationale behind key design decisions to the extent that such a description informs the operational deployment and configuration of an AMP implementation. This document does not address specific data configurations of AMP-enabled devices, nor does it discuss the interface between AMP and other management protocols, such as SNMP.

1.4. 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 RFC 2119 [RFC2119].

2. Terminology

Note: The terms "Actor", "Agent", "Application Data Model", "Atomic Data", "Computed Data", "Controls", "Macros", "Managers", and "Reports" are exactly conformant with those definitions as provided in [AMA]. Brief versions of these definitions are provided in this document as an aid to the reader, but the AMA specification should be read for a complete understanding of these concepts.

This section identifies those terms critical to understanding the proper operation of the AMP. Whenever possible, these terms align in both word selection and meaning with their analogs from other management protocols and with the AMA.

- Actor - A software service running on either managed or managing devices implementing an end-point in the AMP. Actors may implement the "Manager" role, "Agent" role, or both.

- Agent Role - A role within the AMP, associated with a managed device.

- Application Data Model (ADM) - The set of predefined data definitions, reports, literals, operations, and controls given to an AMP actor to manage a particular application or protocol. AMP actors support multiple ADMs, one for each application/protocol being managed.

- Atomic Data - Globally unique, managed data definitions whose definition does not change based on the configuration of an AMP Actor.

- o Computed Data - Data computed dynamically by an AMP Actor.

- o Controls - Operations that may be undertaken by an AMP Actor to change the behavior, configuration, or state of a protocol or application managed by the AMP.

- o Macros - A named, ordered collection of controls.

- o Managed Item Definition (MID) - A parameterized structure used to uniquely identify all data and control definitions within the AMP. MIDs are a super-set of Object Identifiers (OIDs) and the mechanism by which the AMP maintains data compatibility with other management protocols.

- o Manager - A role within the AMP associated with a managing device. AMP managers also provide gateways to non-AMP management protocols as part of conditioning the data returned from agents. Managers interact with one or more agents located on the same device and/or on remote devices in the network.

- o Report - A named, ordered collection of data items gathered by one or more AMP Agents and provided to one or more AMP Managers. Reports may contain atomic data, computed data, and other reports. Individual data within a report are not named; the report itself is named to reduce the size of the report message.

3. Data Model

This section identifies the data types used to capture information within the AMP. The data model consists of the description of basic type definitions, compound type definitions, naming schemes, and type enumerations.

3.1. Basic Types

Basic types are those that are not comprised of any other set of types known to the AMP.

3.1.1. Standard Numeric Types

The AMP supports types for unsigned bytes, 32/64-bit signed and unsigned integers, and 32/64-bit floating point values, as outlined in Table 1.

AMP ID	BitWidth	Description
BYTE	8	unsigned byte value
INT	32	Signed integer in 2's complement
UINT	32	Unsigned integer in 2's complement
VAST	64	Signed integer in 2's complement
UVAST	64	Unsigned integer in 2's complement
REAL32	32	Single-precision, 32-bit floating point value in IEEE-754 format.
REAL64	64	Double-precision, 64-bit floating point value in IEEE-754 format.

Table 1: Standard Numeric Types

3.1.2. Self-Delimiting Numeric Value (SDNV)

The data type "SDNV" refers to a Self-Delimiting Numerical Value (SDNV) described in [RFC6256]. SDNVs are used in the AMP to capture any data items that are expected to be 8 bytes or less in total length. AMP Actors MAY reject any value encoded in an SDNV that is greater than 8 bytes in length.

One popular use of ADNVs in the AMP is to compress the representation of 32/64-bit integer values. This simplifies the AMP by not having to additionally support 8/16-bit versions of integers without incurring significant transmission waste when encoding small numbers into 32/64-bit representations.

3.1.3. Timestamp (TS)

Timestamps represent time within the AMP. For compatibility with a variety of protocols, the use of UTC time is selected to represent all time values in the AMP. However, timestamps in AMP may represent either absolute or relative time based on the associated AMP Epoch.

AMP uses September 9th, 2012 as the timestamp epoch (UTC time 1348025776). Times less than this value MUST be considered as relative times. Values greater than or equal to this epoch MUST be considered as absolute times. In all cases, the AMP timestamp is encoded as an SDNV to avoid the 32-bit 2038 UTC rollover problem.

The absolute time associated with a timestamp can be calculated by an AMP Actor with the following pseudocode.

```
IF (timestamp < 1348025776) THEN
   absolute_time = current_time + timestamp
ELSE
   absolute_time = timestamp
```

3.2. Compound Types

3.2.1. String (STR)

The AMP supports the string type as an ordered collection of byte values assumed terminated by a special NULL-terminator value of 0. Within the AMP, the String type is almost always deprecated in favor of the BLOB type (which asserts the data value first).

3.2.2. Binary Large Object (BLOB)

A Binary Large Object (BLOB) is an ordered collection of bytes prefaced by the number of bytes making up the BLOB. The format of a BLOB is illustrated in Figure 2. BLOBs are used in the AMP to capture variable data sets that are too large to efficiently store in an SDNV.

Binary Large Object Format

```
+---------+--------+--------+     +--------+
| # Bytes | BYTE 1 | BYTE 2 | ... | BYTE N |
|  [SDNV] | [BYTE] | [BYTE] |     | [BYTE] |
+---------+--------+--------+     +--------+
```

Figure 2: Binary Large Object Format

3.2.3. Data Collection (DC)

Often, multiple BLOBs must be communicated in the AMP. A Data Collection (DC) is an ordered set of BLOBs, prefaced by the number of BLOBs making up the collection. The format of a DC is illustrated in Figure 3.

Data Collection

```
+---------+--------+--------+     +--------+
| # BLOBs | BLOB 1 | BLOB 2 | ... | BLOB N |
|  [SDNV] | [BLOB] | [BLOB] |     | [BLOB] |
+---------+--------+--------+     +--------+
```

Figure 3: Data Collection Format

3.2.4. Typed Data Collection (TDC)

When Data Collections are used to convey parameters for AMP controls, or otherwise used to capture the "return" values of controls, the data SHOULD be annotated with the associated type for each data in the collection. The Typed Data Collection (TDC) provides this augmentation. A TDC is a regular DC with one additional BLOB added to the collection to capture type information. This "Type BLOB" stores the type of each other entry in the DC in a byte.

The TDC format is illustrated in Figure 4

Typed Data Collection

```
+---------+-----------+-------------+     +-------------+
| # BLOBs | TYPE BLOB | DATA BLOB 1 | ... | DATA BLOB N |
|  [SDNV] |   [BLOB]  |   [BLOB]    |     |    [BLOB]   |
+---------+-----------+-------------+     +-------------+
```

Figure 4: Typed Data Collection Format

For example, consider a Data Collection of 3 BLOBs, with BLOB 1 having type UINT, BLOB 2 having type VAST, and BLOB 3 having type SDNV. The DC structure has no way of capturing this type information. The corresponding TDC would have 4 BLOBs. BLOB 1 would have length 3 and contain the enumerations for UINT, VAST, and SDNV - each encoded in one byte. BLOBs 2, 3, and 4 would be the original data. This example is illustrated in Figure 5.

Typed Data Collection Example

```
     DC                                          TDC
+---------------------+           +--------------------------------+
| # BLOBs = {3}       |           | # BLOBs = {4}                  |
+---------------------+           +--------------------------------+
| BLOB 1 = {DATA1,...}|--------+  | TYPE BLOB = {UINT, VAST, SNDV} |
+---------------------+        |  +--------------------------------+
| BLOB 2 = {DATA2,...}|-----+  +->| DATA BLOB 1 = {DATA1,...}      |
+---------------------+     |     +--------------------------------+
| BLOB 3 = {DATA3,...}|--+  +---->| DATA BLOB 2 = {DATA2,...}      |
+---------------------+  |        +--------------------------------+
                         +------->| DATA BLOB 3 = {DATA3,...}      |
                                  +--------------------------------+
```

Figure 5: Typed Data Collection Example

3.3. Naming Structures

Application, protocol, and user data defined and exchanged within the AMP must be uniquely identifiable both within a network and (when AMP is used in an overlay) across networks. This section describes the "Managed Identifier" (MID) which is the single data structure used to provide unique naming for all AMP data structures. The MID is a variable-length structure that encapsulates all of the information and annotation necessary to identify an AMP structure.

The MID consists of a "core" unique identifier and several useful annotations to assist with filtering, access control, and parameterization. The unique identifier at the core of a MID is based on the Object Identifier (OID) and its Basic Encoding Rules (BER) as identified in the ITU-T X.690 standard. The specifics of the OID encoding are provided below.

The AMP uses the OID as its fundamental unit of identification to allow its Agents and Managers to more easily interface with other management schemes (such as SNMP) at management boundaries between challenged and un-challenged networks.

The MID structure is comprised of up to four fields, as illustrated in Figure 6.

MID format

```
+--------+--------+--------+--------+
| Flags  | Issuer |   OID  |  Tag   |
| [BYTE] | [SDNV] |[VARIED]| [SDNV] |
|        | (opt)  |        | (opt)  |
+--------+--------+--------+--------+
```

Figure 6: Managed Identifier Format

The MID fields are defined as follows.

Flags

Flags are used to describe the type of component identified by the MID, identify which optional fields in the MID are present, and the encoding used to capture the component's OID. The layout of the flag byte is illustrated in Figure 7.

MID Flag Format

```
+-----+---+---+-----+------+
| OID |TAG|ISS| CAT | TYPE |
+-----+---+---+-----+------+
| 7 6 | 5 | 4 | 3 2 | 1  0 |
+-----+---+---+-----+------+
 MSB                    LSB
```

Figure 7

MID Type (TYPE)

The type of the component encapsulated by the MID, enumerated as data (0), control (1), literal (2), or operator (3).

MID Category (CAT)

The category of the component encapsulated by the MID, enumerated as atomic (0), computed (1), and collection (2).

Issuer Present (ISS)

Whether the issuer field is present (1) or not (0) for this MID. If this flag has a value of 1 then the issuer field MUST be present in the MID. Otherwise, the issuer field MUST NOT be present in the MID.

Tag Present (TAG)

Whether the tag field is present (1) or not (0) for this MID. If this flag has a value of 1 then the tag

field MUST be present in the MID. Otherwise, the tag field MUST NOT be present.

OID Type (OID)

Whether the contained OID field represents an full OID (0), a parameterized OID (1), a compressed full OID (2), or a compressed, parameterized OID (3). OID types are described below.

For example, a MID flag byte of 0x00 indicates an atomic data value with no issuer or tag field encapsulating a full OID. A MID flag byte of 0x94 indicates a computed data value with an issuer field, but no tag field encapsulating a compressed OID.

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 assure MID uniqueness. This value, for example, may come from a global registry of organizations, an issuing node address, or some other network-unique marking.

OID

The core of a MID is its encapsulated OID. Aside from the flag byte, this is the only other mandatory element within a MID. The AMP defines four types of OID references: Full OIDs, Parameterized OIDs, Compressed Full OIDs, and Compressed Parameterized OIDs.

Full OID

This is a binary representation of the full OID associated with the named value. The OID is encoded using a modified form of the ASN.1 Basic Encoding Rules (BER) for Object Identifiers (type value of 0x06). In the standard ASN.1 encoding, four octet sets are defined: identifier octets, length octets, contents octets, and end-of-contents octets. An AMP Full OID does not use the identifier, length, or end-of-contents octets. Instead, an AMP Full OID is comprised of two fields: the length in bytes of the encoded OID captured in an SDNV followed by the OID contents octets. It should be noted that this effectively encodes the "OID" as a Data Collection. The Full OID format is illustrated in Figure 8.

```
                    +----------+
                    | Full OID |
                    |   [DC]   |
                    +----++----+
                         ||
                         ||
 ____________________/  \__________________________
/                                                  \
 +------------+---------+---------+     +---------+
 | OID Length | Octet 1 | Octet 2 | ... | Octet N |
 |   [SDNV]   | [BYTE]  |  [BYTE] |     |  [BYTE] |
 +------------+---------+---------+     +---------+
```

Figure 8: Full OID Format

Parameterized OID

The parameterized OID is represented as the non-parameterized portions of the OID followed by one or more parameters. Parameterized OIDs are used to templatize the specification of data items and otherwise provide parameters to controls without requiring potentially unmanageable growth of a Full OID namespace. The format of a parameterized OID is given in Figure 9.

```
      +----------+------------+
      | FULL OID | Parameters |
      |   [DC]   |    [DC]    |
      +----------+-----++-----+
                         ||
                         ||
    __________________/  \_______________________
   /                                              \
    +----------+--------+--------+     +--------+
    | # Params | Parm 1 | Parm 2 |     | Parm N |
    |  [SDNV]  | [BLOB] | [BLOB] | ... | [BLOB] |
    +----------+--------+--------+     +--------+
```

Figure 9: Parameterized OID Format

Compressed OID

Since many related OIDs share a common and lengthy hierarchy there is opportunity for significant message size savings by defining a shorthand for commonly-used portions of the OID tree. A partial OID is a tuple consisting of a nickname for a pre-defined portion of the OID tree (as an SDNV), followed by a relative OID. Nicknames are defined in

the formal Application Data Models of managed applications and protocols. They must be managed to ensure there is no collision in the nickname namespace. The format of a compressed OID is given in Figure 10.

```
+----------+--------------+
| Nickname | Relative OID |
|  [SDNV]  |     [DC]     |
+----------+--------------+
```

Figure 10: Compressed OID Format

Compressed Parameterized OID

A compressed, parameterized OID is similar to a compressed OID. In this instance, the tuple contained in this field is the nickname for the pre-defined portion of the OID tree (as an SDNV) followed by a parameterized OID whose hierarchy begins at the place identified by the nickname. The format of a compressed OID is given in Figure 11.

Compressed Parameterized OID Format

```
+----------+--------------+------------+
| Nickname | Relative OID | Parameters |
|  [SDNV]  |     [DC]     |    [DC]    |
+----------+--------------+------------+
```

Figure 11: Compressed Parameterized OID Format

Tag

A value used to disambiguate multiple MIDs with the same OID/Issuer combination. The definition of the tag is left to the discretion of the MID issuer. Proper name objects do not require a tag as their OIDs are guaranteed to be globally unique. Options for tag values include an issuer-known version number or a hashing of the data associated with a non-proper-name MIDs. The tag field MUST NOT be present for the atomic category.

3.4. Special Types

In addition to the data types already mentioned, the following special data types are also defined.

3.4.1. MID Collections (MC)

A MID collection is comprised of a value identifying the number of MIDs in the collection, followed by each MID, as illustrated in Figure 12.

```
+--------+-------+     +-------+
| # MIDs | MID 1 | ... | MID N |
| [SDNV] | [MID] |     | [MID] |
+--------+-------+     +-------+
```

Figure 12: MID Collection

3.4.2. Expressions (EXPR)

Expressions apply operations to data and literal values to generate new data values. The expression type in AMP is a collection of MIDs that represent a postfix notation stack of Data, Literal, and Operation types. For example, the infix expression A * (B * C) is represented as the sequence A B C * *. The format of an expression is illustrated in Figure 13.

```
+------------+
| Expression |
|    [MC]    |
+------------+
```

Figure 13

Expression
: An expression is represented in the AMP as a MID collection, where each MID in the ordered collection represents the data, literals, and operations that comprise the expression.

3.4.3. 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". Similar to an expression, a predicate is represented as a MID collection (MC).

3.4.4. Definition (DEF)

Several data structured within the AMP are defined as a named collection of other named entities. For example, a Macro is a named collection of Controls; a Report is a named collection of other data items; a computed data definition is a named expression which, from

above, is a collection of MIDs representing the postfix notation of the calculating function.

The Definition (DEF) type captures instances where the AMP refers to a named, typed collection of information. The format of a DEF is illustrated in Figure 14.

```
+--------+----------+------------+
| Def ID | Def Type | Definition |
| [MID]  |  [BYTE]  |    [MC]    |
+--------+----------+------------+
```

Figure 14: Definition Format

Def ID
: The name of the item being defined. For example, the identifier for the Macro, Report, or Computed Data Item.

Def Type
: The enumeration representing the type for this definition.

Definition
: The MC that captures the definition. This will be context-sensitive based on the structure being defined.

4. AMP Structures

This section identifies the basic structures that comprise the data items exchanged in the AMP, including the definition of these structures based on the aforementioned data model and the behavior of AMP Actors when interacting with these structures.

4.1. AMA Overview

The AMA defines a series of logical components that should be included as part of an asynchronous management protocol. These components are summarized from the AMA in the following table.

Component	Summary Description
Primitive Data	A typed, measured value whose definition does not change.
Computed Data	A value computed from primitive data and other computed data.
Report	Collection of primitive and/or computed data and/or other reports.
Control	Parameterized opcode for an action that can be taken by an Agent.
Rule	A pre-configured response to a pre-defined state on an Agent.
Macro	An ordered collection of controls.
Literal	A constant used when evaluated rules or computing data.
Operator	An opcode representing a mathematical function known to an Agent.

AMP Logical Components

The AMP implements these logical components in largely a one-to-one fashion with the exception that the AMP defines two types of autonomous rules: time-based and state-based. This section describes the format of these data structures in the context of the aforementioned AMP data types. NOTE: The expression of these structures is only to describe how these structures appear in messages exchanged between and amongst Agents and Managers in a challenged network. Individual software applications may choose their own most efficient internal representation of these structures.

4.2. Primitive Data

Primitive Data are pre-defined as part of ADMs for various applications and protocols. These represent values that are directly measured by firmware on Agents rather than computed as a function of other data in the system.

4.2.1. Definition

The representation of these data is simply their identifying MIDs. The representation of a primitive data item is illustrated in Figure 15.

```
+-------+
|  ID   |
| [MID] |
+-------+
```

Figure 15: Primitive Data Format

ID

This is the MID identifying the primitive data. Since primitive data are always defined solely in the context of an ADM, this MID MUST NOT have either an ISSUER field or a TAG field. This ID MUST NOT encapsulate a parameterized OID. The low nibble of the MID flag byte for primitive data is always 0x0.

4.2.2. Processing

Managers

- o Store the MID for each known Primitive Data definition.
- o Associate a data type to each known Primitive Data definition.
- o Encode Primitive Data MIDs in Controls to Agents, as appropriate.

Agents

- o Store the MID for each known Primitive Data definition.
- o Associate a data type to each known Primitive Data definition.
- o Calculate the value of a Primitive Data definition when required, such as when generating a Report or evaluating an Expression.

4.3. Computed Data

Computed data are either as defined in an ADM or tactically by operators in a particular network. Computed data differ from Primitive Data in that they are completely described by other known data in the system (either other Computed Data, or other Primitive Data). For example, letting P# be a Primitive Data item and C# be a

Computed Data item, the following are examples of Computed Data definitions.

C1 = P1 * P2

C2 = C1 + P3

4.3.1. Definition

Computed data are defined by the triplet (TYPE, ID, EXPR) as illustrated in Figure 16. Note that this format is identical to the DEFINITION data type. As such, a computed data definition is captured in a DEF.

```
                        +---------------+
                        | Computed Data |
                        |     [DEF]     |
                        +-------++------+
                                ||
                                ||
                    ____________/  \_______________
                   /                               \
                    +-------+--------+------------+
                    |  ID   |  TYPE  | Definition |
                    | [MID] | [BYTE] |   [EXPR]   |
                    +-------+--------+------------+
```

Figure 16: Computed Data Format

ID

This is the MID identifying the computed data. When a computed data item is defined in an ADM this MID MUST NOT have either an ISSUER field or a TAG field. When the computed data is defined outside of an ADM, the MID MUST have an ISSUE field and, optionally, a TAG field. This ID MUST NOT encapsulate a parameterized OID. The low nibble of the MID flag byte for computed data is always 0x4.

TYPE

This is the resultant type of the computed data, and acts as a static cast for the result of the expression. Note, it is possible to specify a type different then the resultant type of an expression. For example, if an expression adds two single-precision floating point numbers, the computed data may have an integer type associated with it. This byte is populated with the enumeration of the associated type and MUST be defined as one of the numeric data types, as outlined in Section 5.

Definition
The computed data value is computed by an expression. Notably, an EXPR is simply a MC with the restriction that the MIDs represent a series of primitive data, computed data, literals, and operations that form a postfix expression.

4.3.2. Processing

Managers

- o Store the MID for each ADM-defined Computed Data definition.
- o Send requests to Agents to add, list, describe, and remove custom Computed Data definitions.
- o Remember custom Computed Data definitions.
- o Encode Computed Data MIDs in Controls to Agents, as appropriate.

Agents

- o Store the MID for each ADM-defined Computed Data definition.
- o Calculate the value of Computed Data when required, such as during rule evaluation, calculating other Computed Data values, and generating Reports.
- o Add, Remove, List, and Describe custom Computed Data definitions.

4.4. Report

Reports capture information generated by Agents and transmitted to Managers in the AMP. Since the AMP is an asynchronous protocol there is no explicit association between the contents of a report and a generating action by either a Manager or an Agent.

Note that reports in the AMP are NOT key-value pairs. The definition of the report is assumed known by the Agent producing the report and the Manager receiving the report. Similarly, when a report captured the return of a control, the format of the return value is assumed known to the Agent and Manager. This approach results in a significant space saving when communicating reports in the network.

4.4.1. Definition

A report is a named, typed data collection. Each BLOB within the typed data collection is referred to as a report entry. Reports are generated in one of three scenarios: (1) when a report is explicitly

requested, (2) when a rule determines that a report should be generated, and (3) when a control runs and generates a return value that is sent as a report.

When a report is generated by command or by rule, then the report is given a unique identifier either by the ADM defining the report, or by an operator defining a report as part of network configuration. When a report is generated ad-hoc to capture the return value of a control, the report identifier is the same as the identifier of the control whose return value it captures.

The definition of a report is illustrated in Figure 17.

```
                  +-------+---------+
                  |  ID   | Entries |
                  | [MID] |  [TDC]  |
                  +-------+---++----+
                              ||
                              ||
  __________________________/  \__________________________________
 /                                                                 \
   +-----------+-------------+---------+---------+     +---------+
   | # Entries | Entry Types | Entry 1 | Entry 2 |     | Entry N |
   |  [SDNV]   | [BLOB]      | [BLOB]  |  [BLOB] | ... | [BLOB]  |
   +-----------+-------------+---------+---------+     +---------+
```

Figure 17: Report Format

ID

This is the MID identifying the report. When a report is defined in an ADM this MID MUST NOT have either an ISSUER field or a TAG field. When the report is defined outside of an ADM, the MID MUST have an ISSUE field and, optionally, a TAG field. This ID MUST NOT encapsulate a parameterized OID. The low nibble of the MID flag byte for computed data is always 0x8.

Entries

This is the typed data collection containing all of the report entries that comprise the report.

4.4.2. Processing

Managers

o Store the MID for each ADM-defined Report definition.

- o Send requests to Agents to add, list, describe, and remove custom Report definitions.

- o Remember custom Report definitions when processing reports received by Agents.

- o Encode Report MIDs in Controls to Agents, as appropriate.

Agents

- o Store the MID for each ADM-defined Report definition.

- o Populate Reports for transmission to Managers when appropriate, such as when providing a result of running a control or when directed to generate a report by a control or rule.

- o Add, Remove, List, and Describe custom Report definitions.

- o Agents SHOULD collect multiple Report entries into a single Report for transmission to a Manager rather than sending multiple, individual Reports to a Manager.

4.5. Control

A Control represents a pre-defined (possibly parameterized) opcode that can be run on an Agent. Controls in the AMP are always defined in the context of an ADM. There is no concept of an operator-defined control, as controls represent the opcodes of actions taken directly by the firmware. Since Controls are pre-configured in Agents and Managers as part of ADM support, their representation is simply the MID that identifies them, similar to Primitive Data.

One difference between the definition of Controls and Primitive Data is that Controls may accept parameters. This is accomplished by using a parameterized OID in the MID identifying the Control.

4.5.1. Definition

The format of a Control is illustrated in Figure 18.

```
+-------+
|  ID   |
| [MID] |
+-------+
```

Figure 18: Control Format

ID

This is the MID identifying the Control. Since Controls are always defined solely in the context of an ADM, this MID MUST NOT have either an ISSUER field or a TAG field. The low nibble of the MID flag byte for Controls is always 0x1.

4.5.2. Processing

Managers

- o Store the MID for each ADM-defined Control definition.
- o Store the number of parameters and each parameter type for parameterized controls.
- o Encode Control MIDs in other Controls to Agents, as appropriate.

Agents

- o Store the MID for each ADM-defined Control definition.
- o Implement Controls in firmware and run Controls with appropriate parameters when necessary in the context of control messages and autonomous rule execution.
- o Communicate "return" values from Controls back to Managers in a report where such return values are defined for a Control.

4.6. Time-Based Rule (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 Report R1 every 10 hours ending after 20 times.

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

4.6.1. Definition

The format of a TRL is illustrated in Figure 19.

```
+-------+-------+--------+--------+--------+
|  ID   | START | PERIOD | COUNT  | ACTION |
| [MID] | [TS]  | [SDNV] | [SDNV] |  [MC]  |
+-------+-------+--------+--------+--------+
```

Figure 19: Time-Based Rule Format

ID
: This is the MID identifying the TRL. When a report is defined in an ADM this MID MUST NOT have either an ISSUER field or a TAG field. When the report is defined outside of an ADM, the MID MUST have an ISSUE field and, optionally, a TAG field. This ID MUST NOT encapsulate a parameterized OID. The low nibble of the MID flag byte for computed data is always 0x5.

START
: The time at which the TRL should start to be evaluated. This will mark the first running of the action associated with the TRL.

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

COUNT
: The number of times the TRL action will be run. The special value of 0 indicates the TRL should continue running the action indefinitely.

ACTION
: The Macro representing the collection of controls to run as part of the action. This is captured as a MC data type with the constraint that every MID within the MC represent a control or another Macro.

4.6.2. Processing

Managers

- Send requests to Agents to add, list, describe, and remove custom TRL definitions.

- o Remember custom TRL definitions when processing reports received by Agents.

- o Encode TRL MIDs in Controls to Agents, as appropriate.

Agents

- o Run the actions associated with TRLs in accordance with their start time and period.

- o Add, Remove, List, and Describe custom TRL definitions.

4.7. State-Based Rule (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, and for every PERIOD seconds thereafter, an ACTION should be run by the Agent if some CONDITION evaluates to true, until the CONDITION has been evaluated 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 Computed Data C1 > 10, produce Report R1 no more than 20 times.

Starting at the given absolute time, whenever C2 + (P1 * C3) != C4, run Macro M1 no more than 36 times.

4.7.1. Definition

The format of a SRL is illustrated in Figure 20.

```
+-------+-------+-----------+--------+--------+
|  ID   | START | CONDITION | COUNT  | ACTION |
| [MID] | [TS]  |   [PRED]  | [SDNV] |  [MC]  |
+-------+-------+-----------+--------+--------+
```

Figure 20: State-Based Rule Format

ID

This is the MID identifying the SRL. When a report is defined in an ADM this MID MUST NOT have either an ISSUER field or a TAG field. When the report is defined outside of an ADM, the MID MUST have an ISSUE field and, optionally, a TAG field. This ID MUST NOT encapsulate a parameterized OID.

The low nibble of the MID flag byte for computed data is always 0x5.

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 capturing the internal Agent state evaluation which, if true, results in the SRL running the associated action. The predicate is an EXPR which evaluates to "false" if the expression result is 0 and evaluates to "true" in any other case. An expression, itself, is simply a MC representing a series of Primitive Data, Computed Data, Literal, and/or Operator MIDs that are ordered to create a valid postfix expression.

COUNT
: The number of times the SRL condition will be run. The special value of 0 indicates the SRL should continue evaluating the condition indefinitely.

ACTION
: The Macro representing the collection of controls to run as part of the action. This is captured as a MC data type with the constraint that every MID within the MC represent a control or another Macro.

4.7.2. Processing

Managers

- Send requests to Agents to add, list, describe, and remove custom SRL definitions.
- Remember custom SRL definitions when processing reports received by Agents.
- Encode SRL MIDs in Controls to Agents, as appropriate.

Agents

- Run the actions associated with SRLs in accordance with their start time and evaluation of their predicate.
- Add, Remove, List, and Describe custom SRL definitions.

4.8. Macro

Macros in the AMP are simply ordered collections of MIDs (an MC) that describe Controls or other Macros. When run by an Agent, each MID in the MC is run in order. Similar to a Computed Data definition, a Macro is a named collection of MIDs and is represented by a DEF structure.

While the MIDs representing any given control may be parameterized, the MID associated with a Macro MAY NOT be parameterized.

4.8.1. Definition

The format of a Macro is illustrated in Figure 21.

```
                              +-------+
                              | Macro |
                              | [DEF] |
                              +---++--+
                                  ||
                                  ||
                     _____________/  \_______________
                    /                                \
                     +-------+--------+------------+
                     |  ID   |  TYPE  | Definition |
                     | [MID] | [BYTE] |    [MC]    |
                     +-------+--------+------------+
```

Figure 21: Macro Format

ID
: This is the MID identifying the Macro. When a Macro item is defined in an ADM this MID MUST NOT have either an ISSUER field or a TAG field. When the computed data is defined outside of an ADM, the MID MUST have an ISSUE field and, optionally, a TAG field. This ID MUST NOT encapsulate a parameterized OID. The low nibble of the MID flag byte for Macro is always 0x9.

TYPE
: The data type for a Macro is always the special type for Macro as defined in Section 5. Macros do not have a specific return type otherwise.

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

4.8.2. Processing

Managers

- o Store the MID for each ADM-defined Macro definition.
- o Send requests to Agents to add, list, describe, and remove custom Macro definitions.
- o Encode Macro MIDs in Controls to Agents, as appropriate.

Agents

- o Store the MID for each ADM-defined Macro definition.
- o Remember custom Macro definitions and run Macros when appropriate, such as when responding to a run-macro command or when executing the action of a TRL or SRL.
- o Add, Remove, List, and Describe custom Macro definitions.

4.9. Literal

Literals in the AMP represent constants either as defined in an ADM or as specified by a user. Examples of constants that could be defined in an ADM include common mathematical values such as PI or well-known Epochs such as the UNIX Epoch. Examples of constants that could be user-defined as part of expressions include simple numerical values, such as 5 in the expression (A > 5). In both cases, Literals MUST ALWAYS be defined in an ADM, with user-definable Literal values being provided through the MID parameterization mechanism as follows.

The ADM definition of a Literal MUST include the type of the Literal value. Since ADM definitions are preconfigured on Agents and Managers in an AMA the type information for a given Literal is therefore known by all Actors in the system.

If the MID identifying the Literal encapsulates a non-parameterized OID, then the value is given in the ADM and Agents and Managers can lookup this value in their set of pre-configured data.

If the MID identifying the Literal encapsulates a parameterized OID, then the parameters to the OID define the value of the Literal. Users wishing to create a new Literal will create a MID with whatever parameters are necessary to create the value. The documentation of the ADM defining the Literal MUST describe how parameters result in the calculation of the Literal value.

4.9.1. Definition

The format of a Literal is illustrated in Figure 22.

```
+-------+
|  ID   |
| [MID] |
+-------+
```

Figure 22: Control Format

ID

This is the MID identifying the Literal. When a Literal item is defined in an ADM this MID MUST NOT have either an ISSUER field or a TAG field. When the Literal is defined outside of an ADM, the MID MUST have an ISSUE field and, optionally, a TAG field. This ID MUST NOT encapsulate a parameterized OID. The low nibble of the MID flag byte for a literal is always 0x2.

4.9.2. Processing

Managers

- o Store the MID for each ADM-defined Literal definition.
- o Encode Literal MIDs in Controls to Agents, as appropriate.

Agents

- o Store the MID for each ADM-defined Literal definition.
- o Calculate the value of Literals where appropriate, such as when generating a Report or when evaluating an Expression.

4.10. Operator

Operators in the AMP 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 MID that identifies them, similar to Primitive Data and Controls.

The ADM definition of an Operator MUST specify how many parameters are expected. 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 would accept 10 parameters.

4.10.1. Definition

Operators are always evaluated in the context of an Expression. The format of an Operator is illustrated in Figure 23.

```
+-------+
|  ID   |
| [MID] |
+-------+
```

Figure 23: Operator Format

ID

This is the MID identifying the Operator. Since Operators are always defined solely in the context of an ADM, this MID MUST NOT have either an ISSUER field or a TAG field. The low nibble of the MID flag byte for Operators is always 0x3.

4.10.2. Processing

Managers

- o Store the MID for each ADM-defined Operator definition.
- o Encode Operator MIDs in Controls to Agents, as appropriate.

Agents

- o Store the MID for each ADM-defined Operator definition.
- o Store the number of parameters expected for each Operator.
- o Calculate the value of applying an Operator to a given set of parameters, such as when evaluating an Expression.

5. Data Type IDs and Enumerations

This section lists the data type IDs and enumerations for the data types outlined in this section. IDs are the text abbreviations used in this specification and in ADMs to identify data types. Enumerations associate data types with an unsigned integer value. These enumerations MUST be used whenever a data type is represented as a numerical representation, such as the case with the TYPE BLOB of a TDC.

Data Type	ID	Enumeration	Numeric
BYTE	BYTE	0	No
Signed 32-bit Integer	INT	1	Yes
Unsigned 32-bit Integer	UINT	2	Yes
Signed 64-bit Integer	VAST	3	Yes
Unsigned 64-bit Integer	UVAST	4	Yes
Single-Precision Floating Point	REAL32	5	Yes
Double-Precision Floating Point	REAL64	6	Yes
Character String	STR	7	No
Binary Large Object	BLOB	8	No
Self-Delineating Numerical Value	SDNV	9	No
Timestamp	TS	10	No
Data Collection	DC	11	No
Managed Identifier	MID	12	No
Managed Identifier Collection	MC	13	No
Expression	EXPR	14	No
Definition	DEF	15	No
Time-Based Rule	TRL	16	No
State-Based Rule	SRL	17	No
Typed Data Collection	TDC	18	No
Report	RPT	19	No
Macro	MACRO	20	No
Unknown Type	UNK	21	No

AMP Data Type IDs and Enumerations

6. Application Data Model Template

6.1. Overview

An application data model (ADM) specifies the set of AMP components associated with a particular application/protocol. The purpose of the ADM is to provide a guaranteed interface for the management of an application or protocol over AMP that is independent of the nuances of its software implementation. In this respect, the ADM is conceptually similar to the Managed Information Base (MIB) used by SNMP, but contains additional information relating to command opcodes and more expressive syntax for automated behavior.

Currently, the ADM is an organizing document and not used to automatically generate software. As such, the ADM template lists the kind of information that must be present in an ADM definition but does not address mechanisms for automating implementations.

Each ADM specifies the globally unique identifiers and descriptions for all data, controls, literals, and operators associated with the application or protocol managed by the ADM. Any implementation claiming compliance with a given ADM must compute all identified data, perform identified controls, and understand identified literals and operators.

6.2. Template

ADM definitions specify the metadata, data, controls, literals, and operators associated with a managed application or protocol.

6.2.1. ADM Metadata

ADM metadata consist of the items necessary to uniquely identify the ADM itself. The required metadata items include the following.

Item	Type	Description	Req.
Name	STR	The human-readable name of the ADM.	Y
Version	STR	Version of the ADM encoded as a string.	Y
OID Nickname N	OID	ADMs provide an ordered list of nicknames that can be used by other MIDs in the ADM definition to defined compressed OIDs. There can an arbitrary number of nicknames defined for an ADM.	N

Table 2: ADM Terminology

6.2.2. ADM Information Capture

The ADM Data Section consist of all components in the "data" category associated with the managed application or protocol. The information that must be provided for each of these items is as follows.

Name
: Every component in an ADM MUST be given a human-readable, consistent name that uniquely identifies the component in the context of the application or protocol. These names will be used by human-computer interfaces for manipulating components.

MID
: The managed identifier that describes this data item. MIDs in components identified by an ADM MUST NOT contain an issuer or a tag value. In cases where a partial OID is specified, the ADM OID prefix is presumed as the base. In cases where the OID is parameterized, the parameter values are not included in the MID definition in the ADM as parameters are provided at runtime by implementations.

OID
: A human-readable version of the OID encapsulated in the MID for the component (e.g., 1.2.3.4). When a nickname is used to represent an compressed OID, the nickname enumeration is included in this field enclosed by square brackets. For example, if OID nickname 0 refers to the OID prefix 1.2.3.4.5, then the OID 1.2.3.4.5.6 may be listed more compactly as [0].6

Description

Every component in an ADM MUST be given a human-readable, consistent description that provides a potential user with a compact, effective summary of the component.

Type
For components that evaluate to a data value, the data type for that value must be represented.

Parameters
For components with a parameterized OID, the ADM MUST provide the expected number of parameters. A value of 0 indicates that the OID has no parameters and MUST NOT be used for any MID which has a parameterized OID. When omitted, the number of parameters is considered 0.

Parameter N Name
Each parameter of a parameterized component must be given a name.

Parameter N Description
Each parameter of a parameterized component must be given a summary that describes how the parameter will be used by the application or protocol.

Parameter N Type
Each parameter of a parameterized component must be given a type that describes the structure capturing the parameter value. Notably, while parameters in the OID form something similar to a function prototype, there is no sense of function call or stack and therefore all parameters should be considered as pass-by-value.

7. The Agent ADM

The full set of Primitive Data, Computed Data, Reports, Controls, Rules, Macros, Literals, and Operators that can be understood by an AMP Agent have been separated into an AMP Agent ADM. Just as the AMP uses ADMs to manage applications and protocols, the ADM model is also used to implement the functionality of the Agent. As such, the AMP message specification is limited to three basic communications:

- Adding an Agent to the list of managed devices known to a Manager.

- Sending a Macro of one or more Controls to an Agent.

- Receiving a Report of one or more Report Entries from an Agent.

as outlined in Section 8.

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

8. Functional Specification

This section describes the format of the messages that comprise the AMP protocol. When discussing the format/types of data that comprise message fields, the following conventions are used.

8.1. 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 24. These message groups are assumed communicated amongst Agents and Managers as the payloads of encapsulating protocols which MAY provide additional security and data integrity features.

```
+--------+-----------+-----------+     +-----------+
| # Msgs | Timestamp | Message 1 | ... | Message N |
| [SDNV] |    [TS]   |   [VAR]   |     |   [VAR]   |
+--------+-----------+-----------+     +-----------+
```

Figure 24: AMP Message Group Format

Msgs
: The number of messages that are together in this message group.

Timestamp
: The creation time for this messaging group. This timestamp MUST be an absolute time. Individual messages may have their own creation timestamps based on their type, but the group timestamp also serves as the default creation timestamp for every message in the group.

Message N
: The Nth message in the group.

8.2. Message Format

Each message identified in the AMP specification adheres to a common message format, illustrated in Figure 25, consisting of a message header, a message body, and an optional trailer.

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

Figure 25: AMP Message Format

Header

The message header byte is shown in Figure 26. The header identifies a message context and opcode as well as flags that control whether a report should be generated on message success (Ack) and whether a report should be generated on message failure (Nack).

```
+--------+----+---+---------+-------+
|ACL Used|Nack|Ack| Context |Opcode |
+--------+----+---+---------+-------+
|    7   |  6 | 5 |   4  3  | 2 1 0 |
+--------+----+---+---------+-------+
 MSB                            LSB
```

Figure 26: AMP Common Message Header

Opcode

The opcode field identifies the opcode of the message, within the associated message context.

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 communicating this status. Otherwise, the actor MAY generate such a report 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 communicating this status. Otherwise, the Actor MAY generate such a report 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 identifiers 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.

8.3. Register Agent (0x00)

The Register Agent message is used to inform a AMP manager of the presence of another agent in the network.

```
+----------+
| Agent ID |
| [SDNV]   |
+----------+
```

Figure 27: Register Agent Message Body

Agent ID

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

8.4. Data Report (0x12)

Data reports include a listing of one or more data items collected from a managed device. These reports may include atomic data, computed data, or any report definition known to the generating device. Each message is a concatenation of ID/Data definitions with the overall message length assumed to be captured in the underlying transport container.

```
+------+----------+-------+-------+     +-------+
| Time | Num Rpts | RPT 1 | RPT 2 |     | RPT N |
| [TS] |  [SDNV]  | [RPT] | [RPT] | ... | [RPT] |
+------+----------+-------+-------+     +-------+
```

Figure 28: Data Report Message Body

Time
The time at which the report was generated by the AMP Actor.

Num Rpts
The number of reports in the data report message.

RPT N
The Nth report.

8.5. Perform Control (0x20)

The perform control method causes the receiving AMP Actor to apply one or more pre-configured controls provided in the form of a Macro.

```
+-------+-----------+
| Start |  Controls |
|  [TS] |    [MC]   |
+-------+-----------+
```

Figure 29: Perform Control Message Body

Start
The time at which the Macro should be run.

Controls
The collection of controls (Macro) to be run by the AMP Actor. The MID identifying a control will contain the parameters for the control (if any) through the use of a parameterized OID captured within the MIDs in the MC.

9. IANA Considerations

At this time, this protocol has no fields registered by IANA.

10. 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.

11. References

11.1. Informative References

[AMA] Birrane, E., "Asynchronous Management Architecture", draft-birrane-dtn-ama-00 (work in progress), August 2015.

[I-D.irtf-dtnrg-dtnmp] Birrane, E. and V. Ramachandran, "Delay Tolerant Network Management Protocol", draft-irtf-dtnrg-dtnmp-01 (work in progress), December 2014.

11.2. Normative References

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

[RFC6256] Eddy, W. and E. Davies, "Using Self-Delimiting Numeric Values in Protocols", RFC 6256, DOI 10.17487/RFC6256, May 2011, <http://www.rfc-editor.org/info/rfc6256>.

Author's Address

Edward J. Birrane
Johns Hopkins Applied Physics Laboratory

Email: Edward.Birrane@jhuapl.edu