Delay-Tolerant Networking   E. Birrane
Internet-Draft   Johns Hopkins Applied Physics Laboratory
Intended status: Experimental   J. Mayer
Expires: December 28, 2016   INSYEN AG
                                                          June 26, 2016

Asynchronous Management Protocol
draft-birrane-dtn-amp-03

Abstract

This document describes an Asynchronous Management Protocol (AMP) in conformance with the 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 reduces 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.

Table of Contents

1. Introduction

This document specifies an Asynchronous Management Protocol (AMP) that provides application-layer network management service 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 time and state measurements.

Networks whose communication links are frequently challenged by physical or administrative effects cannot guarantee the low-latency, duplex data communications necessary to support sessions and other synchronous communication. For such networks, a new 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.

- 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 unidirectional links.

- 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 minimizes the size of a message whenever practical, to include packing and unpacking binary data, variable-length fields, and pre-configured data definitions.

- Absolute and Custom 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.

- Autonomous, 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.

- 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

- 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 data types defined by the AMP are self-terminating. This means that, given an indefinite-length octet stream, each data type 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.

- 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)". 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

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 outside of the scope of this protocol.

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 [RFC2119].

2. Terminology

Note: The terms "Actor", "Agent", "Application Data Model", "Atomic Data", "Computed Data", "Control", "Literal", "Macro", "Manager", "Report Template", "Report Entry", and "Rule" are used without modification from the definitions provided in [AMA].

Additional terms critical to understanding the proper operation of the AMP are as follows.

- 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. MIDs are defined in Section 3.3.

- Report (RPT) - An ordered collection of report entries gathered by an Agent and provided to one or more Managers. Reports represent the fundamental unit of data exchange from an Agent to a Manager within the AMP. Report messages are defined in Section 7.4.

- State-Based Rule (SRL) - A rule in the AMP whose action is performed if a defined predicate evaluates to true. SRLs are defined in Section 4.7.

- Time-Based Rule (TRL) - A rule in the AMP whose action is performed at regular intervals. SRLs are defined in Section 4.6.

3. Data Model

This section identifies the data types used to capture information within the AMP.

3.1. Primitive Types

Primitive 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, 32/64-bit floating point values, and strings, as outlined in Table 1.

AMP Type	Bit Width	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.
STR	Varies	NULL-terminated series of characters in UTF-8 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 SDNVs 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)

A timestamp value can represent either a relative or absolute time within the AMP. An AMP relative time is defined as the number of seconds between two AMP events (such as the receipt of a control by an agent and the execution of that control). An AMP absolute time is defined as UTC time using the Unix/POSIX Epoch.

Since timestamps are a common component in AMP messages and controls, they should be made as small as possible. Therefore, timestamps in AMP do not add a special flag to determine whether the given time is an absolute or relative time. Instead, AMP defines a simple formula to unambiguously determine the type of time represented without increasing the overall size of a timestamp.

AMP uses September 9th, 2012 as the timestamp epoch (UTC time 1347148800). Times less than this value MUST be considered a relative time. 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 unambiguously with the following pseudocode.

```
IF (timestamp < 1347148800) THEN
   absolute_time = current_time + timestamp
ELSE
   absolute_time = timestamp
```

3.2. Compound Types

Compound types are data types defined as an aggregation of other data types.

3.2.1. 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

3.2.2. Data Collection (DC)

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

3.2.3. Typed Data Collection (TDC)

The Typed Data Collection (TDC) is a special kind of DC which encodes type information as the first BLOB in the collection. The TDC data type is used to capture typical "TLV" (type, length, value) information in the AMP.

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

The TDC fields are defined as follows.

BLOBs
This represents the number of BLOBS that comprise the TDC. Since the TDC has one BLOB for each data item in the collection, plus one additional BLOB for type information,

the # BLOBs value MUST be equal to one more than the number of data items in the collection.

Type BLOB
Each BYTE in the Type BLOB represents a type enumeration of a corresponding Data BLOB. For example, the 3rd BYTE in the Type BLOB holds the type enumeration of the 3rd Data BLOB in the TDC. Since there is exactly 1 byte per data item in the Type BLOB, the overall size of this BLOB MUST be the total number of data items in the TDC.

Data BLOB
The Nth Data BLOB holds the Nth data value in the collection.

For example, consider the following set of data values: {(UINT) 3, (REAL32) 3.14, (STR) "pi"}. The corresponding TDC would have 4 BLOBs. BLOB 1 would have length 3 and contain the enumerations for UINT, REAL32, and STR - encoded in one BYTE each. BLOBs 2, 3, and 4 would hold the original data. This example is illustrated in Figure 5.

Typed Data Collection Example

```
     Data Set                                          TDC
+---------------+           +---------------------------------+
| # Items = 3   |           | # BLOBs = {4}                   |
+---------------+           +---------------------------------+
| (UINT)   3    |--------+  | TYPE BLOB = {UINT, REAL32, STR} |
+---------------+        |  +---------------------------------+
| (REAL32) 3.14 |-----+  +->| DATA BLOB 1 = {3}               |
+---------------+     |     +---------------------------------+
| (STR)    "pi" |--+  +---->| DATA BLOB 2 = {3.14}            |
+---------------+  |        +---------------------------------+
                   +------->| DATA BLOB 3 = {"pi"}            |
                            +---------------------------------+
```

Figure 5: Typed Data Collection Example

The rationale for extracting data type information into a Type BLOB and placing that BLOB at the beginning of the TDC is to enable faster performance for type validators. With the Type BLOB, a validator can inspect one BLOB to ensure that the elements within the TDC match the expected type specifications. Without a Type BLOB, type information would need to be interspersed with data values throughout the TDC. In that case, a type validator would need to scan through the entire set of bytes comprising the TDC looking for type information. This would significantly alter the speed of type checking in the AMP.

The rationale for placing data values directly in a Data BLOB is to enable rapid navigation. As mentioned in Section 1.2, every data type defined in the AMP is deterministic in length. However, this determination may require deep inspection of the data in cases of variable-length headers and optional fields. By placing the data value in a Data BLOB, the length of the value may be asserted to allow a data parser to rapidly calculate the position of data item N in the TDC. The redundancy of storing a pre-calculated length for each data value when the data value length can be calculated from the data itself is a processing tradeoff made by AMP given the relative frequency with which the TDC is used to communicate Report and Control parameters.

3.2.4. Table (TBL)

A TBL is a names, typed, collection of tabular data with each row represented as a DC and each column defined by both a column name and a column type. Each row in the TBL MUST have the same length and the ith BLOB of each row DC MUST correspond to the ith column in the table.

The TBL format is illustrated in Figure 6

```
                             Table


      +-----------+-----------+--------+-------+     +-------+
      | Col Names | Col Types | # Rows | Row 1 |     | Row N |
      |   [DC]    |  [BLOB]   | [SDNV] | [DC]  | ... |  [DC] |
      +-----------+-----------+--------+-------+     +-------+
```

Figure 6: Table Format

The TBL fields are defined as follows.

Col Names
: Column names are captured as a DC with the ith entry in the DC representing the name of the ith column. This DC MUST have a number of entries equal to the number of columns in the table. Each entry in the DC is considered to be of type STR.
NOTE: It is being considered to make this field a TDC instead of a DC to allow individual Col Names to be of different data types, instead of making them always be strings.

Col Types

Similar to the Type BLOB of the TDC, the Col Types BLOB contains one BYTE for each column in the TBL and this BYTE holds the type enumeration for the column. Therefore, the Col Types BLOB MUST have a length equal to the number of columns in the table and each BYTE in the BLOB MUST contain a correct enumeration of an AMP type. This field MUST contain one of the AMP data structure enumerations identified in Section 5.

Rows
This field captures the number of rows in the TBL. If the number of rows in the TBL is set to 0, that indicates there is no additional data after this field.

Row 1 .. Row N
Each row in the TBL is represented by a DC, with the ith BLOB in the DC representing the data in the ith column of the TBL. Each row DC MUST have a number of BLOBs equal to the number of columns.

The Figure below illustrates a table of data relating to months of the year on the left and the corresponding populated TBL structure for this table on the right.

```
                                 +---------------------------------------+
+-----+------+------+     +----->| Col Names DC = {"Month","Ord","Days"}|
|Month| Ord  | Days |-----+      +---------------------------------------+
|(STR)|(UINT)|(UINT)|----------->| Col Types BLOB = {STR, UINT, UINT}   |
+-----+------+------+            +---------------------------------------+
|Jan  |   1  |  31  |---------+  | Num Rows = 3                          |
+-----+------+------+         |  +---------------------------------------+
|Oct  |  10  |  31  |-----+  +-->| Row 1 DC = {"Jan", 1, 31}             |
+-----+------+------+     |      +---------------------------------------+
|June |  6   |  30  |--+  +----->| Row 2 DC = {"Oct", 10, 31}            |
+-----+------+------+  |         +---------------------------------------+
                       +-------->| Row 3 DC = {"June", 6, 30}            |
                                 +---------------------------------------+
```

Figure 7: Table Example

3.3. Managed Identifiers (MIDs)

Structures 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) used to provide unique naming for the AMP structures defined in Section 4. The MID is a variable-length structure with optional fields.

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 use of OIDs in the MID structure allows Agents and Managers to interface with other management schemes (such as SNMP) at management boundaries between challenged and unchallenged networks.

The MID consists of a mandatory flag BYTE, a mandatory OID, and optional annotations to assist with filtering, access control, and parameterization. The MID structure is illustrated in Figure 8.

```
                           MID format

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

Figure 8: Managed Identifier Format

The MID fields are defined as follows.

Flags

Flags are used to describe the type of structure 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 9.

```
                         MID Flag Format

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

Figure 9

STRUCT ID

The lower nibble of the MID flag identifies the kind of data structure being identified by this identifier. This field MUST contain one of the AMP data structure enumerations identified in Section 5.

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 a full OID (0), a parameterized OID (1), a compressed full OID (2), or a compressed, parameterized OID (3).

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 MID 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 MID defined as part of an ADM.

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, which are defined as follows.

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 followed by the OID contents octets. It should be noted that this matches, exactly, the

definition of the BLOB type. The Full OID format is illustrated in Figure 10.

```
                   +----------+
                   | Full OID |
                   | [BLOB]   |
                   +----++----+
                        ||
                        ||
 _____________________/  \_________________________
/                                                  \
 +------------+---------+---------+     +----------+
 | OID Length | Octet 1 | Octet 2 | ... | Octet N  |
 |   [SDNV]   | [BYTE]  |  [BYTE] |     |  [BYTE]  |
 +------------+---------+---------+     +----------+
```

Figure 10: Full OID Format

Parameterized OID
The parameterized OID is represented as a Full 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 11.

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

Figure 11: 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, followed by a relative OID. Nicknames are defined in Section 3.4. The format of a compressed OID is given in Figure 12.

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

Figure 12: 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 13.

Compressed Parameterized OID Format

```
+----------+-------------+------------+
| Nickname | Relative OID | Parameters |
|  [SDNV]  |   [BLOB]     |   [TDC]    |
+----------+-------------+------------+
```

Figure 13: 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. Options for tag values include an issuer-known version number or a hashing of the data associated with the MID. The tag field MUST NOT be present for any MID defined as part of an ADM.

3.4. Nicknames

There are several strategies for reducing the overall size of an OID in an operational system. The AMP method for OID size reduction is to publish global enumerations that represent strategic nodes in an OID tree. This published, global enumeration is called a Nickname.

As mentioned in the discussion of compressed OIDs above, a nickname is used in lieu of a portion of the OID tree. ADMs may define their own nicknames so long as their definitions do not conflict with the definitions of nicknames in other ADMs. AMP does not provide the ability to assign nicknames dynamically.

Like other numeric types, nicknames are encoded as SDNVs allowing them to be of arbitrary length. For example, 3 bytes of SDNV can encode over 2 million nicknames. Assuming ADMs are allotted 10 nicknames each, this approach can accommodate over 200,000 ADMs before requiring a 4th byte for nickname information.

Additionally, since nicknames are globally unique, neither an AMP Agent or Manager is ever required to expand a compressed OID to assert uniqueness or perform other identification. It is recommended that compressed OIDs be used whenever possible.

3.5. Parameters

Parameterized OIDs provide a powerful mechanism for customizing behavior for certain AMP structures. Parameterized values in AMP are formally defined in ADMs with a well-known, static typing. When an ADM specifies that an identified AMP structure may be parameterized, the specification MUST list the number of expected parameters and the type associated with each parameter. When a particular instance of a parameterized AMP structure is generated by an Agent or a Manager, the MID identifying that instance MUST contain a parameterized OID and the parameters associated with the OID MUST match in number and type the specification.

3.5.1. Optional Parameters

When parameterizing an AMP structure, some parameters may be optional with default values defined if parameters are omitted. The use of optional parameters helps keep MID values small when using default values for parameters is a common case, rather than forcing all parameters to be provided all the time.

Since each individual parameter in a TDC is represented as a BLOB, a parameter can be omitted by specifying a length of 0 BYTES for the Data BLOB holding the parameter. If a parameter is omitted and is not considered optional by the parameterized AMP structure, this MUST be considered an error.

3.5.2. Parameter Evaluation

The type value associated with the TDC in a parameter list is only used to provide type-checking safety to ensure that the given parameters match expected parameter types. It is important to understand that the types in the parameter TDC DO NOT define the parameterized interface - only the ADM defines the typed interface.

Parameters within the TDC may be represented in one of two ways: the parameter itself (parameter by value), or an expression used to determine the parameter (parameter by evaluation).

3.5.2.1. Parameter By Value

When specifying a parameter using a value, the BYTE representing the parameter type MUST be set to the expected parameter type and the BLOB representing the parameter contents MUST be the parameter value.

For example, consider a parameterized OID that takes 1 parameter, which it expects to be an unsigned integer (UINT). When populating this parameter by value, the type of the populated parameter field MUST be UINT and the parameter value MUST be the unsigned integer.

3.5.2.2. Parameter By Evaluation

When the value of a parameter is likely to change, an Expression (EXPR) may be substituted for the parameter value. When it comes time to interpret the parameter value, the current value of the Expression is calculated and used as the parameter value.

A parameter defined by evaluation MUST be of type EXPR, and the type of the EXPR must be equal to the expected type of the parameter. Expressions and Expression types are discussed in Section 3.6.2.

NOTE: If the expected type of the parameter is already EXPR, and a parameter of type EXPR is provided, then the system MUST treat the situation as if it were a parameter by value. AMP DOES NOT support an EXPR which references another EXPR as doing so leads to significant confusion in implementations and the possibility of circular reference.

3.5.2.3. Identifying Parameter Approach

The determination of whether a parameter has been provided by value or by evaluation is made by comparing the given type of the parameter to the expected type of the parameter.

If the parameter type and the expected type match, then the parameter MUST be considered by value. If the parameter type is an EXPR and the EXPR type matches the expected type, then the parameter MUST be considered by evaluation of the EXPR. In any other case, the parameter MUST be considered invalid as being from a type mismatch.

3.6. Special Types

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

3.6.1. 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 14.

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

Figure 14: MID Collection

3.6.2. Expressions (EXPR)

Expressions apply mathematical operations to values to generate new values on an Agent. The EXPR type in AMP is a collection of MIDs that represent a postfix notation stack of data, Literal, and Operator 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 15.

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

Figure 15

Type
: The enumeration representing the type of the result of the evaluated expression.

Expression
: An expression is represented in the AMP as a MID collection, where each MID in the ordered collection represents the data, Literals, and/or Operations that comprise the Expression.

3.6.3. 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".

4. AMP Structures

This section identifies the AMP structures that implement the AMA logical data model.

4.1. AMA Overview

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

AMA Component	Summary Description	AMP Structure
Atomic Data	A typed, measured value whose definition and value determination occurs externally to the AMP.	Externally Defined Data
Computed Data	A typed, computed value whose definition and value determination occurs within the AMP.	Variable
Report Entry	Collection of Atomic and/or Computed data and/or other Reports.	Report Entry
Control	Parameterized opcode for any action that can be taken by an Agent.	Control
Rule	A pre-configured response to a pre-defined time or state on an Agent.	State-Based Rule, Time-Based Rule
Macro	An ordered collection of Controls.	Macro
Literal	A constant used when evaluating Rules or determining the value of Computed Data.	Literal
Operator	An opcode representing a mathematical function known to an Agent.	Operator

AMP Logical Components

The AMP implements these logical components in largely a one-to-one fashion with a few exceptions. This section describes the format of these structures in the context of the aforementioned AMP data types. NOTE: The expression of these structures is only to describe how they appear in messages exchanged between and amongst Agents and Managers. Individual software applications may choose their own internal representation of these structures.

4.2. Externally Defined Data (EDD)

Externally defined data (EDD) are defined as part of ADMs for various applications and protocols. These represent values that are calculated outside of the context of Agents and Managers, such as those values measured by firmware. As such, their value is defined external to the AMP system.

4.2.1. Definition

The representation of these data is simply their identifying MIDs. The representation of an EDD is illustrated in Figure 16.

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

Figure 16: Externally Defined Data Format

ID
: This is the MID identifying the EDD. Since EDDs are always defined solely in the context of an ADM, this MID MUST NOT have an ISSUER field and MUST NOT have a TAG field.

4.2.2. Processing

Managers

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

Agents

- Store the MID for each known EDD definition.
- Associate a data type to each known EDD definition.

o Calculate the value of an EDD definition when required, such as when generating a Report Entry or evaluating an Expression.

4.3. Variables (VAR)

Variables (VAR) are either statically defined in an ADM or dynamically defined by a particular network. They differ from EDDs in that they are completely described by other known data in the system (either other Variables, or other EDDs). For example, letting E# be a EDD item and V# be a VAR item, the following are examples of VAR definitions.

V1 = E1 * E2

V2 = V1 + E3

4.3.1. Definition

VARs are defined by the triplet (ID, TYPE, EXPR) as illustrated in Figure 17.

```
                  +------------+
                  |  Variable  |
                  |   [VAR]    |
                  +-----++-----+
                        ||
                        ||
        ______________/  \______________
       /                                \
        +-------+--------+-------------+
        |  ID   |  Type  | Initializer |
        | [MID] | [BYTE] |   [EXPR]    |
        +-------+--------+-------------+
```

Figure 17: Variable Format

ID
: This is the MID identifying the VAR. When defined in an ADM this MID MUST NOT have an ISSUER field and MUST NOT have a TAG field. When defined outside of an ADM, the MID MUST have an ISSUER field and MAY have a TAG field. This ID MUST NOT encapsulate a parameterized OID.

Type
: This is the type of the VAR, and acts as a static cast for the result of the initializing Expression. Note, 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. This BYTE is populated with the enumeration of the associated type and MUST be defined as one of the numeric data types outlined in Section 5.

Initializer
The initial value of the VAR is given by an initializing Expression. In the case where the type of the VAR is an EXPR, then the initializer is simply copied as the value of the VAR. In the case where the type of the VAR is anything other than EXPR, then the initializer Expression is evaluated and the resultant value is copied into the VAR as its value. Once the initializer Expression has been used to calculate an initial value for the VAR it may be discarded.

4.3.2. Processing

Managers

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

Agents

- Store the MID for each ADM-defined VAR definition.
- Calculate the value of VARs when required, such as during Rule evaluation, calculating other VAR values, and generating Reports.
- Add, remove, list, and describe custom VAR definitions.

4.4. Report Template (RPTT), Report Entry (RPTE)

A Report is an AMP message whose format is described in Section 7.4. This message is populated with Report Entries that contain data formatted in accordance with Report Templates.

A Report Template is the ordered set of data descriptions that describe how values will be represented in a corresponding Report Entry. Templates can be viewed as a schema that describes how to interpret a Report Entry, since these entries do not embed schema or name information in them. Templates contain no values and are either defined in an ADM or configured between Managers and Agents.

A Report Entry is a set of data values populated using a given Report Template. A Report Entry contains only data values and no template definitions. By removing definition information from a Report Entry, the volume of information sent from the Agent to the Manager is greatly reduced. When a Report Entry is generated as capturing the result of a Control, the Report Template for the Control is assumed to be known to both the generating Agent and all receiving Managers.

4.4.1. Definition

A Report Template is modeled as a MC, as each data definition in the template is identified by a MID.

A Report Entry is a TDC identified by a MID and generated to capture the return value of a Control. Generated Report Entries MUST be collected by an Agent periodically, placed in an AMP Report message, and sent to one or more Managers.

When a Report Entry is generated in accordance with a named Report Template, the entry identifier MUST be the same as the template defining the data in the entry. When a Report Entry is generated absent a defined Report Template, then the entry identifier MUST be the MID of the Control generating the report.

The definition of a Report Entry is illustrated in Figure 18.

```
                  +-------+--------+
                  |  ID   | Values |
                  | [MID] |  [TDC] |
                  +-------+---++---+
                              ||
                              ||
  ___________________________/  \___________________________________
 /                                                                  \
   +----------+------------+---------+---------+     +---------+
   | # Values | Value Types | Value 1 | Value 2 |     | Value N |
   |  [SDNV]  |    [BLOB]   | [BLOB]  |  [BLOB] | ... | [BLOB]  |
   +----------+------------+---------+---------+     +---------+
```

Figure 18: Report Entry Format

ID

This is the MID identifying the source used to build the entry. If this field identifies a template, and the Report

Template is defined in an ADM, this MID MUST NOT have an ISSUER field and MUST NOT have a TAG field. If the Report Template is not defined in an ADM then this MID MUST have an ISSUER field and MAY have a TAG field. If this field identifies a Control then the MID MUST NOT have an ISSUER field and MUST NOT have a TAG field.

A Report Template MID MAY be parameterized. If the Report Template MID is parameterized, the parameters MUST be used (in the same number and order) to customize any parameterized data in the report when generating values for the Report Entry.

Values
: This is the TDC containing all of the data values that comprise the Report Entry. It is important to note that data values may be other Report Entries.

4.4.2. Processing

Managers

- Store the MID for each ADM-defined Report Templates.
- Send requests to Agents to add, list, describe, and remove custom Report Templates.
- Remember custom Report Templates when processing Report Entries received by Agents.
- Encode Report Template MIDs in Controls to Agents, as appropriate.

Agents

- Store the MID for each ADM-defined Report Template.
- Populate Report Entries for transmission to Managers when required by a Control.
- Add, remove, list, and describe custom Report Templates.
- Agents SHOULD collect multiple Report Entries into a single Report AMP message for transmission to a Manager rather than sending multiple, individual Report messages to a Manager with one Report Entry per Report message.

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. 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 EDDs.

4.5.1. Definition

The format of a Control is illustrated in Figure 19.

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

Figure 19: 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 an ISSUER field and MUST NOT have a TAG field.

4.5.2. Processing

Managers

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

Agents

- Store the MID for each ADM-defined Control definition.
- Implement Controls in firmware and run Controls with appropriate parameters when necessary in the context of Manager direction and Rule execution.
- Communicate "return" values from Controls back to Managers as Report Entries where appropriate.

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 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.

4.6.1. Definition

The format of a TRL is illustrated in Figure 20.

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

Figure 20: Time-Based Rule Format

ID
: This is the MID identifying the TRL. When a TRL is defined in an ADM this MID MUST NOT have an ISSUER field and MUST NOT have a TAG field. When the TRL is defined outside of an ADM, the MID MUST have an ISSUER field and MAY have a TAG field. This ID MUST NOT encapsulate a parameterized OID.

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 may be run. The special value of 0 indicates the TRL should continue running the action indefinitely.

ACTION
The collection of Controls and/or Macros to run by the TRL. This is captured as a MC with the constraint that every MID within the MC represent a Control or Macro.

4.6.2. Processing

Managers

- Send requests to Agents to add, list, describe, and remove custom TRL definitions.
- Remember custom TRL definitions when processing Reports received by Agents.
- Send requests to Agents to suspend/resume the evaluation of TRLs.
- Encode TRL MIDs in Controls to Agents, as appropriate.

Agents

- Run the actions associated with TRLs in accordance with their start time and period.
- Add, remove, list, and describe custom TRL definitions.
- Suspend and resume the evaluation of a TRL when directed by a Manager or another Rule.
- Report on the status of TRLs.

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 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.

4.7.1. Definition

The format of a SRL is illustrated in Figure 21.

```
+-------+-------+--------+--------+--------+
|  ID   | START |  COND  | COUNT  | ACTION |
| [MID] | [TS]  | [PRED] | [UINT] |  [MC]  |
|       |       |        |        |        |
+-------+-------+--------+--------+--------+
```

Figure 21: 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 an ISSUER field and MUST NOT have a TAG field. When the SRL is defined outside of an ADM, the MID MUST have an ISSUER field and MAY have a TAG field. This ID MUST NOT encapsulate a parameterized OID.

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.

COUNT
: 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 captured as a MC data type with the constraint that every MID within the MC represent a Control or Macro.

4.7.2. Processing

Managers

- Send requests to Agents to add, list, describe, suspend, resume, and remove custom SRL definitions.

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

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

Agents

- o Run the actions associated with SRLs in accordance with their start time and evaluation of their predicate.

- o Add, remove, list, and describe custom SRL definitions.

- o Suspend and resume SRL evaluation when commanded by a Manager or another Rule.

4.8. Macro

Macros in the AMP are ordered collections of MIDs (an MC) that contain Controls or other Macros. When run by an Agent, each MID in the MC is run in order.

Any AMP implementation MUST allow at least 4 levels of Macro nesting. Implementations MUST provide some mechanism to prevent recursive nesting of Macros.

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 22.

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

Figure 22: Macro Format

ID

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

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 Control 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 defined in an ADM. 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.

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 23.

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

Figure 23: Control Format

ID
: This is the MID identifying the Literal. Since Literal definitions are always provided in an ADM, this MID MUST NOT have an ISSUER field and MUST NOT have a TAG field.

4.9.2. Processing

Managers

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

Agents

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

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 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 24.

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

Figure 24: 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 an ISSUER field and MUST NOT have a TAG field.

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 IDs and enumerations for 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 a numeric value. These enumerations MUST be used whenever a data type is represented as a numerical representation.

NOTE: Type enumerations are always represented as a BYTE in the AMP.

IDs and enumerations are grouped by the kind of data they represent, as follows. AMP structure identifiers occupy enumerations 0 - 8 and represent AMP data structures that are formally identified by a MID.

Basic data types occupy enumerations 9-18 and represent primitive data types in the AMP specification. Compound and special types occupy enumerations 19-25 and represent other data types known to the AMP specification.

AMP Structure	ID	Enumeration	Numeric
Externally Defined Data	EDD	0	No
Variable	VAR	1	No
Report	RPT	2	No
Control	CTRL	3	No
State-Based Rule	SRL	4	No
Time-Based Rule	TRL	5	No
Macro	MACRO	6	No
Literal	LIT	7	No
Operator	OP	8	No

Basic Data Type	ID	Enumeration	Numeric
BYTE	BYTE	9	No
Signed 32-bit Integer	INT	10	Yes
Unsigned 32-bit Integer	UINT	11	Yes
Signed 64-bit Integer	VAST	12	Yes
Unsigned 64-bit Integer	UVAST	13	Yes
Single-Precision Floating Point	REAL32	14	Yes
Double-Precision Floating Point	REAL64	15	Yes
Self-Delineating Numerical Value	SDNV	16	No
Timestamp	TS	17	No
Character String	STR	18	No

Compound/Special Data Type	ID	Enumeration	Numeric
Binary Large Object	BLOB	19	No
Managed Identifier	MID	20	No
MID Collection	MC	21	No
Expression	EXPR	22	No
Data Collection	DC	23	No
Typed Data Collection	TDC	24	No
Table	TBL	25	No

5.1. Numeric Promotions

When attempting to evaluate operators of different types, wherever possible, an Agent MAY need to promote operands until they are of the correct type. For example, if an Operator is given both an INT and a REAL32, the INT SHOULD be promoted to a REAL32 before the Operator is applied.

The listing of legal promotions in the AMP are listed in Figure 25. In this Figure, operands are listed across the top row and down the first column. The resultant type of the promotion is listed in the table at their intersection.

	INT	UINT	VAST	UVAST	REAL32	REAL64
INT	INT	INT	VAST	UNK	REAL32	REAL64
UINT	INT	UINT	VAST	UVAST	REAL32	REAL64
VAST	VAST	VAST	VAST	VAST	REAL32	REAL64
UVAST	UNK	UVAST	VAST	UVAST	REAL32	REAL64
REAL32	REAL32	REAL32	REAL32	REAL32	REAL32	REAL64
REAL64	REAL64	REAL64	REAL64	REAL64	REAL64	REAL64

Figure 25: AMP Numeric Promotions

AMP does not permit promotions between non-numeric types, and numeric promotions not listed in this section are not allowed in the AMP. Any attempt to perform an illegal promotion in the AMP SHOULD result in an error.

5.2. Numeric Conversions

Variables, Expressions, and Predicates in the AMP are typed values. When attempting to assign a value of a different type, a numeric conversion must be performed. Any numeric type may be converted to any other numeric type in accordance with the C rules for arithmetic type conversions.

6. Application Data Model Template

6.1. Overview

An application data model (ADM) specifies the set of AMP components associated with a particular application or 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.

Any implementation claiming compliance with a given ADM must collect all identified EDDs, compute all identified Variables, perform identified Controls and Macros, generate Report Entries to defined Report Templates, and understand identified Literals and Operators.

6.2. Template

Each ADM specifies the globally unique identifiers and descriptions for all EDDs, Variables, Controls, Literals, Macros, Report Templates, and Operators associated with the application or protocol managed by the ADM.

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 field and MUST NOT contain a TAG field. In cases where the OID is parameterized, the parameter values are not included in the ADM MID definition as parameters are provided at runtime.

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 item.

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. This description MUST note if the parameter is optional.

Parameter N Type
: Each parameter of a parameterized component must be given a type that describes the structure capturing the parameter value.

6.3. The Agent ADM

The full set of EDDs, Variables, Report Templates, 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.

7. Functional Specification

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

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

7.1. 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 26. 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]   | [VARIES]  |     |  [VARIES] |
+--------+----------+----------+     +-----------+
```

Figure 26: 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.

7.2. Message Format

Each message identified in the AMP specification adheres to a common message format, illustrated in Figure 27, consisting of a message header, a message body, and an optional trailer.

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

Figure 27: AMP Message Format

Header

The message header BYTE is shown in Figure 28. 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).

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

Figure 28: AMP Common Message Header

Opcode

The opcode field identifies the opcode of the message.

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 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.

7.3. Register Agent (0x00)

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

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

Figure 29: 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.

7.4. Data Report (0x12)

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.

Reports are an ordered collection of Report Entries collected from a managed device.

```
+------+---------+-----------+---------+   +---------+
| Time | RX Name | # Entries | ENTRY 1 |   | ENTRY N |
| [TS] |  [BLOB] |   [SDNV]  | [RPTE]  |...| [RPTE]  |
+------+---------+-----------+---------+   +---------+
```

Figure 30: Data Report Message Body

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

RX Name
: The identifier of the Manager meant to receive this report.

Entries
The number of Report Entries in the Report.

ENTRY N
The Nth Report Entry.

7.5. Perform Control (0x1A)

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

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

Figure 31: Perform Control Message Body

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

Controls
The collection of MIDs that represent the Controls and/or Macros to be run by the AMP Actor.

8. IANA Considerations

At this time, this protocol has no fields registered by IANA. However, such a registry MUST be established to capture certain data elements provided in ADMs, such as nicknames and root OIDs.

9. 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.

10. References

10.1. 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.

10.2. 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>.

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 and has agreed to document the access control list and error reporting portion of the specification.

Authors' Addresses

Edward J. Birrane
Johns Hopkins Applied Physics Laboratory

Email: Edward.Birrane@jhuapl.edu

Jeremy Pierce-Mayer
INSYEN AG
Muenchner Str. 20
Oberpfaffenhofen, Bavaria DE
Germany

Phone: +49 08153 28 2774
Email: jeremy.mayer@insyen.com