

Delay Tolerant Networking
Internet-Draft
Intended status: Experimental
Expires: October 6, 2016

B. Sipos
RKF Engineering
E. Birrane, Ed.
JHU APL
April 4, 2016

A YANG profile for defining Asynchronous Management Protocol Application
Data Models
[draft-bsipos-dtn-amp-yang-01](#)

Abstract

This document specifies a YANG profile for defining Application Data Model (ADM) schema for the Asynchronous Management Protocol (AMP). The AMP has no relation to NETCONF; YANG is used here only for its language syntax, and its module and type systems.

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 October 6, 2016.

Copyright Notice

Copyright (c) 2016 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.	Requirements Language	3
2.	Application Data Models	3
2.1.	Module Restrictions	3
2.2.	OID Assignment	4
3.	YANG Types for AMP	4
3.1.	The Integer Types	4
3.2.	The Floating Point Types	5
3.3.	Other Simple Types	5
3.4.	The Compound Types	5
3.5.	Object Identifier Types	5
3.6.	Applicaition Module Subtyping	5
4.	YANG Extensions for AMP	6
4.1.	Object Identifiers	6
4.1.1.	The fulloid Extension Statement	6
4.1.2.	The suboid Extension Statement	6
4.1.3.	The nickname Extension Statement	7
4.1.4.	The compressoid Extension Statement	7
4.2.	The group Extension Statement	8
4.3.	Model Definition Extensions	8
4.3.1.	The primitive Extension Statement	8
4.3.2.	The control Extension Statement	9
4.3.3.	The parameter Extension Statement	10
4.3.4.	The result Extension Statement	10
4.3.5.	The report Extension Statement	11
4.3.6.	The reportitem Extension Statement	11
4.3.7.	The macro Extension Statement	12
4.3.8.	The operator Extension Statement	12
4.3.9.	The operand Extension Statement	13
4.4.	Data Instance Extensions	13
4.4.1.	The number-instance Extension Statement	14
4.4.2.	The string-instance Extension Statement	15
4.4.3.	The BLOB-instance Extension Statement	15
4.4.4.	The TS-instance Extension Statement	15
4.4.5.	The MID-instance Extension Statement	16
4.4.6.	The DC-instance Extension Statement	17
4.4.7.	The MC-instance Extension Statement	17
4.4.8.	The TDC-instance Extension Statement	17

5.	IANA Considerations	18
6.	Security Considerations	19
7.	References	19
7.1.	Normative References	19
7.2.	Informative References	20

Appendix A.	YANG Definitions	20
A.1.	AMP Module	20
A.2.	AMP Type Submodule	21
A.3.	AMP Extensions Submodule	27
A.4.	AMP Instances Submodule	29
Appendix B.	Example Application Data Model	32
Authors' Addresses	34

[1.](#) Introduction

This profile uses YANG [[RFC6020](#)] as an encoding for the management schema and makes use of the YANG module and type systems. The fact that YANG is also used to specify data models for the NETCONF protocol has no direct influence over this use of YANG to specify data models for AMP.

This specification follows [[RFC6087](#)] in the definition of the "amp-adm" YANG module. The amp-adm module defines only subtypes and extensions; it does not define any actual data model elements.

[1.1.](#) 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.](#) Application Data Models

An AMP application SHALL define its Application Data Model (ADM) by means of a YANG module which imports and uses "amp-adm" module extensions. An official Pyang tool [[pyang](#)] plugin SHOULD be used to validate the contents of an ADM YANG module.

[2.1.](#) Module Restrictions

A YANG module which defines an AMP ADM SHALL NOT be used to also

define a data model for NETCONF or any other non-AMP protocol. It may be syntatically allowable to mix models for multiple protocols but it decreases the intelligability of the module for either purpose.

A YANG module which defines an AMP ADM SHALL NOT contain any "default" statements.

[2.2.](#) OID Assignment

For each ADM-specific YANG statement which requires an OID to be assigned to it, it is possible to use one of the "fulloid", "suboid", or "compressoid" substatements to make that assignment. Not all of the OID assignment substatements are available in all contexts, so following the allowed substatement table is important.

The types of OID assignments are:

- o Full OIDs can be used in any situation where an OID is needed.
- o Compressed OIDs can also be used in any situation where an OID is needed, but require the use of a module-unique OID nickname ID.
- o Sub-OIDs can be used where the OID being assigned is relative to the structural (module-statement-wise) parent of the assignment.

An ADM SHOULD make use of sub-OIDs where possible, both to avoid typos possible with full OIDs and to avoid nickname assignment for every group within the ADM. Using sub-OIDs also guarantees that the tree structure of the ADM module matches one-for-one with the OID tree.

[3.](#) YANG Types for AMP

This section specifies how AMP types interact with native YANG types. All AMP data types are sub-typed from YANG native types solely for the purpose of providing a baseline of behavior for YANG parsers.

Any YANG module which defines an AMP ADM SHALL only use types (or derived types) from the "amp-adm" module.

[3.1.](#) The Integer Types

The AMP types of "BYTE", "INT", "UINT", "VAST", and "UVAST" are derived directly from the built-in YANG type of the same numeric domain. There is no functional difference between these types and the native types, other than the namespace of these types. Using AMP-specific names allows ADM module authors to keep consistent terminology between textual specification and YANG specification.

The "SDNV" type is derived from YANG "binary" type due to its domain being larger than any of the built-in YANG types.

[3.2.](#) The Floating Point Types

The AMP types of "REAL32" and "REAL64" are derived from YANG "binary" type because the built-in floating point type is not a clear superset of the floating point types of [[I-D.birrane-dtn-amp](#)].

[3.3.](#) Other Simple Types

The "SDNV" type is derived from YANG "binary" type due to its domain being larger than any of the built-in YANG numeric types.

The "TS" type is also derived from YANG "binary" type due to the more complex encoding semantics of the TS type.

[3.4.](#) The Compound Types

The "STR" type is derived from the YANG "string" type only because they both are intended to have the semantics of human-readable text. An ADM SHOULD NOT use the amp:STR type for any data other than text encoded with UTF-8 (see [[RFC3629](#)]). The encoding of "STR" type is wholly unrelated to any NETCONF use of the YANG "string" type.

The "BLOB" type is derived from YANG "binary" type. The BLOB is the simplest AMP-specific type which is encoded using internal sub-items (the size separate from the bytes).

The "DC" and "TDC" types are also derived from YANG "binary" type for lack of specific YANG mechanism for type decomposition.

[3.5.](#) Object Identifier Types

The "MID" type is derived from YANG "binary" type due to its combined use of bit patterns (in its header) and BER-encoded data.

The "MC" type is also derived from YANG "binary" type for lack of specific YANG mechanism for type decomposition.

[3.6.](#) Applicaiton Module Subtyping

An ADM SHOULD subtype any numeric type in order to apply additional semantic context to the numerical values (similar to the SMIV2 CounterXX and GaugeXX [[RFC2578](#)]). An ADM SHOULD make use of the YANG "units" substatement when numeric types are used (within either a typedef or a type-use).

An ADM SHOULD subtype the BLOB type in order to identify application-specific encoding formats. Using plain BLOB types within an ADM is discouraged due to the opaqueness of the

Any ADM subtype SHALL have no effect on the value encoding of AMP. Subtypes are purely used to assist applications in managing value semantics. Any ADM subtype SHALL include a description substatement explaining the purpose of the subtype.

[4.](#) YANG Extensions for AMP

This section specifies how AMP extension statements interact with native YANG statements within an application YANG module.

[4.1.](#) Object Identifiers

This section contains extensions for identifying a data node within the YANG model tree by a unique OID value.

[4.1.1.](#) The fulloid Extension Statement

A "fulloid" statement is used to anchor an item in the OID tree. The value of a fulloid is the dotted-numeric notation of the OID value. A fulloid value must not be empty. Any substatements which are sibling to a "fulloid" will be relative to that OID root for the purposes of "suboid" processing.

The format of a fulloid argument is a string of period-separated numeric components.

+-----+-----+	
substatement	cardinality
+-----+-----+	
description	0..1
+-----+-----+	

Table 1: fulloid Substatements

[4.1.2.](#) The suboid Extension Statement

A "suboid" statement is used to define an item's OID relative to a sibling statement's full OID. The value of a suboid is the dotted-numeric notation of the OID parts under the full OID. A suboid value must not be empty.

The format of a suboid argument is the same as a fulloid argument. The interpretation of a suboid depends upon the context of the statement.

+-----+-----+	
substatement	cardinality
+-----+-----+	
description	0..1
+-----+-----+	

Table 2: suboid Substatements

[4.1.3.](#) The nickname Extension Statement

A "nickname" statement is used to define an application-specific numeric identifier for a full OID prefix. The nickname is used by both the AMP agent and manager to shorten OID encoding.

The format of a nickname argument is a single non-negative integer value. Each nickname is defined within the namespace of the ADM module.

+-----+-----+	
substatement	cardinality
+-----+-----+	
description	0..1
fulloid	1
+-----+-----+	

Table 3: nickname Substatements

[4.1.4.](#) The compressoid Extension Statement

A "compressoid" statement is used to define a full OID based on a module-specific nickname (see [Section 4.1.3](#)) as a prefix and a suboid suffix.

The format of a compressoid argument is a nickname value (see [Section 4.1.3](#)) within square brackets followed by a suboid string.

+-----+-----+	
substatement	cardinality
+-----+-----+	
description	0..1
+-----+-----+	

Table 4: compressoid Substatements

[4.2.](#) The group Extension Statement

The "group" statement is used to define a grouping of other items within the ADM. Each group is assigned an OID (see [Section 2.2](#)) and used as an OID anchor for its substatements.

The order of substatements within a group is not significant. Only the OID assignment of each item is significant.

substatement	cardinality
description	0..1
reference	0..1
status	0..1
fulloid suboid compressoid	1
group	0..*
primitive	0..*
control	0..*
report	0..*
macro	0..*
operator	0..*

Table 5: group Substatements

[4.3.](#) Model Definition Extensions

[4.3.1.](#) The primitive Extension Statement

The "primitive" statement is used to define an atomic value within an ADM. Each primitive is assigned an OID (see [Section 2.2](#)) and a type. The primitive has different semantics from the YANG "leaf" statement due to the lack of secondary (non-type) attributes (e.g. config/state distinction). All primitive objects are state; any configuration is performed via "control" statements.

substatement	cardinality
description	0..1
reference	0..1
status	0..1
fulloid suboid compressoid	1
type	1
units	0..1

Table 6: primitive Substatements

[4.3.2.](#) The control Extension Statement

The "control" statement is used to define an available control within the ADM. Each control is assigned an OID (see [Section 2.2](#)) and an ordered list of parameter and result items. The control has different semantics from the YANG "rpc" statement due to the difference in protocol encoding and to the asynchronous nature of the AMP.

Each control is parameterized by some number of parameters (see [Section 4.3.3](#)) and some number of results (see [Section 4.3.4](#)). There is no provision in an ADM for specifying alternative parameters or alternative results (i.e. no parameters are optional).

substatement	cardinality
description	0..1
reference	0..1
status	0..1
fulloid suboid compressoid	1
parameter	0..*
result	0..*

Table 7: control Substatements

Sipos & Birrane

Expires October 6, 2016

[Page 9]

Internet-Draft

A YANG profile for AMP ADMs

April 2016

[4.3.3.](#) The parameter Extension Statement

The "parameter" statement is used to define single expected parameter of a "control" statement. In the AMP each control has a fixed number of typed parameters, there is no provision for overloaded controls which take variable numbers of parameters.

A control parameter is an atomic value with a distinct type, but has no distinct OID. The parameter statement's argument is used as the parameter's name. Within a single control, each parameter name SHALL be unique. The parameter name is not related to any AMP encoding, so is useful only for the sake of identifying the parameter within the ADM.

substatement	cardinality
description	0..1
type	1
units	0..1

Table 8: parameter Substatements

[4.3.4.](#) The result Extension Statement

The "result" statement is used to define single expected result of a "control" statement. In the AMP each control has a fixed number of typed results, there is no provision for overloaded controls which yield variable numbers of results.

A control result is an atomic value with a distinct type, but has no distinct OID. The result statement's argument is used as the result's name. Within a single control, each result name SHALL be unique. The result name is not related to any AMP encoding, so is useful only for the sake of identifying the result within the ADM.

substatement	cardinality
description	0..1
type	1
units	0..1

Table 9: result Substatements

[4.3.5.](#) The report Extension Statement

The "report" statement is used to define the contents of an AMP report, but does not define when any instances of the report may be created. Each report is assigned an OID (see [Section 2.2](#)) and an ordered list of content items. Asynchronous reporting is a distinct feature of the AMP from other management protocols.

Each report is parameterized by some number of items which are to be contained in corresponding report instances (see [Section 4.3.6](#)). There is no provision in an ADM for specifying alternative report contents.

substatement	cardinality
description	0..1
reference	0..1

status	0..1	
fulloid suboid compressoid	1	
reportitem	0..*	
+-----+-----+		

Table 10: report Substatements

[4.3.6.](#) The reportitem Extension Statement

The "reportitem" statement is used to define single expected item within a report instance. In the AMP each report has a fixed number of typed items, there is no provision for overloaded reports which yield variable numbers of items.

A reportitem is an atomic value with a distinct OID of the primitive to be included in a report instance. A reportitem has no type of its own. The reportitem statement's argument is used as the item's name. Within a single report, each reportitem name SHALL be unique. The reportitem name is not related to any AMP encoding, so is useful only for the sake of identifying the item within the ADM.

+-----+-----+		
substatement	cardinality	
+-----+-----+		
description	0..1	
fulloid compressoid	1	
+-----+-----+		

Table 11: reportitem Substatements

[4.3.7.](#) The macro Extension Statement

The "macro" statement is used to declare an AMP macro within an ADM.

//FIXME: what value is there in the inline definition?

+-----+-----+		
substatement	cardinality	

description	0..1
reference	0..1
status	0..1
fulloid suboid compressoid	1

Table 12: macro Substatements

4.3.8. The operator Extension Statement

The "operator" statement is used to define the syntax of an ADM operator (for use in expressions). Each operator is assigned an OID (see [Section 2.2](#)) and an ordered list of operands.

Each operator is parameterized by some number of items which are to be used as operands at statement execution time. (see [Section 4.3.9](#)).

substatement	cardinality
description	0..1
reference	0..1
status	0..1
fulloid suboid compressoid	1
operand	0..*

Table 13: operator Substatements

4.3.9. The operand Extension Statement

The "operand" statement is used to define single expected operand within an operator statement. In the AMP each operation has a fixed number of untyped operands. There is no provision for overloaded operators which take variable numbers of operands.

An operand is an atomic value with no associated OID or type. The operand statement's argument is used as the item's name. Within a single operator, each operand name SHALL be unique. The operand name is not related to any AMP encoding, so is useful only for the sake of identifying the item within the ADM.

+-----+	+-----+
substatement	cardinality
+-----+	+-----+
description	0..1
+-----+	+-----+

Table 14: operand Substatements

[4.4.](#) Data Instance Extensions

Some aspects of an ADM module require in-line instantiations of data which will eventually be encoded in AMP format. Rather than requiring the ADM module author to do the encoding manually, and to allow easier inspection by an ADM module reader, each to-be-encoded data item is represented in the ADM module by one of the "amp:*-instance" statements. Examples of uses for these instances are: a numeric value used in the definition of a "literal" statement, or OID values used in the definition of a "report" statement.

Where possible, an ADM author SHOULD supply all data instances necessary to interpret the constant-data items within an ADM.

[4.4.1.](#) The number-instance Extension Statement

The "number-instance" statement is used to define single value of one of the integer data types (BYTE, INT, UINT, VAST, UVAST, SDNV) or one of the floating point types (REAL32, REAL64). The number-type substatement SHALL be present unless the number-instance is a direct substatement of a typed statement (e.g. a literal statement).

The lexical representation of all AMP integer types SHALL conform to the corresponding integer types [RFC6020]. The lexical representation of AMP floating point types SHALL conform to the "decimal64" type of [RFC6020]. The binary-valued floating point domain of the AMP types SHALL be enforced by any YANG module parser.

substatement	cardinality
description	0..1
number-type	0..1

Table 15: number-instance Substatements

[4.4.1.1.](#) The number-type Extension Statement

The "number-type" statement is used to identify the specific encoding type for a number-instance parent statement. A number-type statement's argument SHALL be one of the numeric type names:

BYTE
 INT
 UINT
 VAST
 UVAST
 SDNV
 REAL32
 REAL64

The number-type statement has no substatemnts.

[4.4.2.](#) The string-instance Extension Statement

The "string-instance" statement is used to define single text value for the STR data type. The string-instance argument SHALL NOT contain the UTF-8 code point zero. Code point zero is used to terminate strings in AMP.

substatement	cardinality
description	0..1

Table 16: string-instance Substatements

4.4.3. The BLOB-instance Extension Statement

The "BLOB-instance" statement is used to define single binary-data value for the BLOB data type. The BLOB-instance argument SHALL be represented by Base-64 encoded text according to [\[RFC3548\]](#). The length of the encoded BLOB is implicit in the BLOB-instance representation.

substatement	cardinality
description	0..1

Table 17: string-instance Substatements

4.4.4. The TS-instance Extension Statement

The "TS-instance" statement is used to define single time-stamp value. Although its encoding is identical to the SDNV number-instance, the TS-instance YANG representation is different. The TS-instance argument SHALL contain an fully qualified absolute time represented according to [\[RFC3339\]](#).

substatement	cardinality
description	0..1

Table 18: TS-instance Substatements

[4.4.5.](#) The MID-instance Extension Statement

The "MID-instance" statement is used to define single MID value, including all of the possible MID variations.

substatement	cardinality
description	0..1
MID-type	1
MID-category	1
MID-issuer	0..1
MID-tag	0..1

Table 19: MID-instance Substatements

[4.4.5.1.](#) The MID-issuer Extension Statement

The "MID-issuer" statement is used to define the optional Issuer payload of the MID value. The presence or absence of a MID-issuer statement determines the header and payload encoding of the MID value. The MID-issuer argument SHALL be a positive integer value.

substatement	cardinality
description	0..1

Table 20: MID-issuer Substatements

[4.4.5.2.](#) The MID-tag Extension Statement

The "MID-tag" statement is used to define the optional Tag payload of the MID value. The presence or absence of a MID-tag statement determines the header and payload encoding of the MID value. The MID-tag argument SHALL be a positive integer value.

Internet-Draft

A YANG profile for AMP ADMs

April 2016

substatement	cardinality
description	0..1

Table 21: MID-tag Substatements

[4.4.6.](#) The DC-instance Extension Statement

The "DC-instance" statement is used to define a list of BLOB values. The count of values present in the DC is implicit in the number of BLOB-instance substatements. The order of BLOB-instance substatements SHALL correpond with the encoded DC value.

substatement	cardinality
description	0..1
BLOB-instance	0..*

Table 22: DC-instance Substatements

[4.4.7.](#) The MC-instance Extension Statement

The "MC-instance" statement is used to define a list of MID values. The count of values present in the MC is implicit in the number of MID-instance substatements. The order of MID-instance substatements SHALL correpond with the encoded MC value.

substatement	cardinality
description	0..1
MID-instance	0..*

Table 23: MC-instance Substatements

4.4.8. The TDC-instance Extension Statement

The "TDC-instance" statement is used to define a list of typed data instance values. The count of values present in the TDC is implicit in the number of "*-instance" substatements. The order of data instance substatements SHALL correspond with the encoded TDC value.

Sipos & Birrane

Expires October 6, 2016

[Page 17]

Internet-Draft

A YANG profile for AMP ADMs

April 2016

substatement	cardinality
description	0..1
number-instance string-instance BLOB-instance TS-instance MID-instance DC-instance TDC- instance	0..*

Table 24: TDC-instance Substatements

5. IANA Considerations

This document registers one URI in the IETF XML registry [[RFC3688](#)].
NOTE TO EDITOR: module registration is pending I-D approval. The following registration has been made:

Field	Value
URI	urn:ietf:params:xml:ns:yang:amp-adm
Registrant Contact	The DTN WG of the IETF.
XML	N/A, the requested URI is an XML namespace.

This document registers one module name/namespace in the IETF YANG Module Names Registry [[RFC6020](#)]. NOTE TO EDITOR: module registration is pending I-D approval. The following registration has been made:

Field	Value
Name	amp-adm
Namespace	urn:ietf:params:xml:ns:yang:amp-adm
Prefix	amp
Reference	draft-bsipos-dtn-amp-yang

6. Security Considerations

This memo only defines a mechanism to specify an application schema, it does not impose any limitations or requirements on the contents of that schema. The amp-adm module defines only subtypes and extensions. It does not define any actual data model elements, so there are no direct security implications.

7. References

7.1. 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>>.
- [RFC3339] Klyne, G. and C. Newman, "Date and Time on the Internet: Timestamps", [RFC 3339](#), DOI 10.17487/RFC3339, July 2002, <<http://www.rfc-editor.org/info/rfc3339>>.
- [RFC3548] Josefsson, S., Ed., "The Base16, Base32, and Base64 Data Encodings", [RFC 3548](#), DOI 10.17487/RFC3548, July 2003, <<http://www.rfc-editor.org/info/rfc3548>>.
- [RFC3629] Yergeau, F., "UTF-8, a transformation format of ISO

10646", STD 63, [RFC 3629](#), DOI 10.17487/RFC3629, November 2003, <<http://www.rfc-editor.org/info/rfc3629>>.

- [RFC3688] Mealling, M., "The IETF XML Registry", [BCP 81](#), [RFC 3688](#), DOI 10.17487/RFC3688, January 2004, <<http://www.rfc-editor.org/info/rfc3688>>.
- [RFC6020] Bjorklund, M., Ed., "YANG - A Data Modeling Language for the Network Configuration Protocol (NETCONF)", [RFC 6020](#), DOI 10.17487/RFC6020, October 2010, <<http://www.rfc-editor.org/info/rfc6020>>.
- [RFC6087] Bierman, A., "Guidelines for Authors and Reviewers of YANG Data Model Documents", [RFC 6087](#), DOI 10.17487/RFC6087, January 2011, <<http://www.rfc-editor.org/info/rfc6087>>.
- [I-D.birrane-dtn-amp]
Birrane, E. and J. Pierce-Mayer, "Asynchronous Management Protocol", [draft-birrane-dtn-amp-02](#) (work in progress), March 2016.

Sipos & Birrane

Expires October 6, 2016

[Page 19]

Internet-Draft

A YANG profile for AMP ADMs

April 2016

[7.2.](#) Informative References

- [RFC2578] McCloghrie, K., Ed., Perkins, D., Ed., and J. Schoenwaelder, Ed., "Structure of Management Information Version 2 (SMIv2)", STD 58, [RFC 2578](#), DOI 10.17487/RFC2578, April 1999, <<http://www.rfc-editor.org/info/rfc2578>>.
- [I-D.birrane-dtn-ama]
Birrane, E., "Asynchronous Management Architecture", [draft-birrane-dtn-ama-02](#) (work in progress), March 2016.
- [pyang] Bjorklund, M., "An extensible YANG validator and converter in python", March 2016.
- [CCITT.X690.2002]
International Telephone and Telegraph Consultative Committee, "ASN.1 encoding rules: Specification of basic encoding Rules (BER), Canonical encoding rules (CER) and

Distinguished encoding rules (DER)", CCITT Recommendation X.690, July 2002.

[Appendix A](#). YANG Definitions

The contents of this section is the machine-readable specification of this YANG module.

[A.1](#). AMP Module

The following YANG definition is the top-level "amp" module.

Sipos & Birrane	Expires October 6, 2016	[Page 20]
-----------------	-------------------------	-----------

Internet-Draft	A YANG profile for AMP ADMs	April 2016
----------------	-----------------------------	------------

```
<CODE BEGINS> file "amp-adm.yang"
```

```
module amp-adm {
  namespace "urn:ietf:params:xml:ns:yang:amp-adm";
  prefix "amp";

  include amp-types;
  include amp-extensions;
  include amp-instances;

  organization
    "IETF Delay Tolerant Networking Working Group";
```

contact
"WG Web: <<http://tools.ietf.org/wg/dtn/>>
WG List: <<mailto:dtm@ietf.org>>

WG Chairs: Brian Haberman
<<mailto:brian@innovationslab.net>>
Marc Blanchet
<<mailto:Marc.Blanchet@viagenie.ca>>

Editor: Brian Sipos
<<mailto:BSipos@rkf-eng.com>>;

```
description
    "This module implements the "
    +"Asynchronous Management Protocol (AMP) "
    +"Application Data Model (ADM) profile within YANG";
reference "draft Asynchronous Management Protocol";

revision "2016-04-01" {
    description "Updated to fix typos.";
    reference "I-D draft-bsipos-dtn-amp-yang";
}

revision "2016-03-14" {
    description "Initial draft release.";
    reference "I-D draft-bsipos-dtn-amp-yang";
}

}
```

<CODE ENDS>

[A.2.](#) AMP Type Submodule

The following YANG definition includes types specific to AMP.

```
<CODE BEGINS> file "amp-types.yang"
submodule amp-types {
```

```
    belongs-to amp-adm {
        prefix "amp";
    }

organization
```



```
"IETF Delay Tolerant Networking Working Group";
contact
  "WG Web: <http://tools.ietf.org/wg/dtn/>
  WG List: <mailto:dtm@ietf.org>
```

```
WG Chairs: Brian Haberman
            <mailto:brian@innovationslab.net>
            Marc Blanchet
            <mailto:Marc.Blanchet@viagenie.ca>
```

```
Editor: Brian Sipos
        <mailto:BSIpos@rkf-eng.com>;
```

```
description
  "This submodule contains the set of core types necessary to "
  +"define an Asynchronous Management Protocol data model.";
reference "draft Asynchronous Management Protocol";
```

```
revision "2016-04-01" {
  description "Updated to fix typos.";
  reference "I-D draft-bsipos-dtn-amp-yang";
}
revision "2016-03-14" {
  description "Initial draft release.";
  reference "I-D draft-bsipos-dtn-amp-yang";
}
```

```
// These extensions are only used within this submodule for annotation
extension amp-type-id {
  argument "num";
  description "Internal annotation of the AMP ID number of a type";
}
extension amp-type-item {
  argument "name";
  description "Internal annotation of a sub-type item.";
}
extension amp-type-list {
  argument "name";
  description "Internal annotation of a sub-type list-of-items.";
}

typedef "BYTE" {
  type "uint8";
```

```

        amp:amp-type-id 0;
        description "Unsigned 8-bit integer";
        reference "draft Asynchronous Management Protocol";
    }
    typedef "INT" {
        type "int32";
        amp:amp-type-id 1;
        description "Signed 32-bit integer";
        reference "draft Asynchronous Management Protocol";
    }
    typedef "UINT" {
        type "uint32";
        amp:amp-type-id 2;
        description "Unsigned 32-bit integer";
        reference "draft Asynchronous Management Protocol";
    }
    typedef "VAST" {
        type "int64";
        amp:amp-type-id 3;
        description "Signed 64-bit integer";
        reference "draft Asynchronous Management Protocol";
    }
    typedef "UVAST" {
        type "uint64";
        amp:amp-type-id 4;
        description "Unsigned 64-bit integer";
        reference "draft Asynchronous Management Protocol";
    }
    typedef "REAL32" {
        type "binary";
        amp:amp-type-id 5;
        description
            "Binary encoding of IEEE-754 32-bit floating point number";
        reference "draft Asynchronous Management Protocol";
    }
    typedef "REAL64" {
        type "binary";
        amp:amp-type-id 6;
        description
            "Binary encoding of IEEE-754 64-bit floating point number";
        reference "draft Asynchronous Management Protocol";
    }
    typedef "SDNV" {
        type "binary";
        amp:amp-type-id 9;
        description
            "Binary encoding of self-delimited numeric value.";
        reference "draft Asynchronous Management Protocol";
    }

```

Internet-Draft

A YANG profile for AMP ADMs

April 2016

```
}
typedef "STR" {
    type "string";
    amp:amp-type-id 7;
    description
        "Same UTF-8 encoding as YANG base type. "
        +"Must be zero-terminated.";
    reference "draft Asynchronous Management Protocol";
}

typedef "TS" {
    type "binary";
    amp:amp-type-id 10;
    description "A timestamp value.";
    reference "draft Asynchronous Management Protocol";
}

typedef "MID" {
    type "binary";
    amp:amp-type-id 12;
    description "The basic managed-identifier definition.";
    reference "draft Asynchronous Management Protocol";
}

typedef "BLOB" {
    type "binary";
    amp:amp-type-id 8;
    description
        "The BLOB type should be used as a base type for "
        +"applicaiton-specific types used in data models.";
    reference "draft Asynchronous Management Protocol";

    amp:amp-type-item "count" { type "SDNV"; }
    amp:amp-type-list "octets" { type "BYTE"; }
}

typedef "DC" {
    type "binary";
    amp:amp-type-id 11;
    description "Untyped data collection";

    amp:amp-type-item "count" { type "SDNV"; }
```

```

        amp:amp-type-list "items" { type "BLOB"; }
    }
    typedef "TDC" {
        type "binary";
        amp:amp-type-id 18;
        description "Typed data collection";
    }

```

```

        amp:amp-type-item "entry-count" { type "SDNV"; }
        // These really need not be BLOBs with internal sizes
        amp:amp-type-item "entry-types" { type "BLOB"; }
        amp:amp-type-list "entry-values" { type "BLOB"; }
    }
    typedef "MC" {
        type "binary";
        amp:amp-type-id 13;
        description "Ordered list of MID values.";

        amp:amp-type-item "count" { type "SDNV"; }
        amp:amp-type-list "values" { type "MID"; }
    }

    // Should be pure MC with no type-id?
    // The only time [EXPR] type is used in AMP spec is
    // for DEF definition, which is unambiguous on type.
    typedef "EXPR" {
        type "binary";
        amp:amp-type-id 14;
        description
            "Ordered list of MID values representing a "
            +"postfix arithmetic.";

        amp:amp-type-item "expression" { type "MC"; }
    }
    // PRED is not a type
    typedef "DEF" {
        type "binary";
        amp:amp-type-id 15;
        description
            "Ordered list of MID values with a corresponding result
            +"type and overall OID";

        amp:amp-type-item "id" { type "MID"; }
    }

```

```

        amp:amp-type-item "type" { type "BYTE"; }
        amp:amp-type-item "definition" { type "MC"; }
    }
    typedef "TRL" {
        type "binary";
        amp:amp-type-id 16;
        description
            "Identify and define a time-based macro rule.";

        amp:amp-type-item "id" { type "MID"; }
        amp:amp-type-item "start" { type "TS"; }
        amp:amp-type-item "period" { type "SDNV"; units "seconds"; }
        amp:amp-type-item "count" { type "SDNV"; }
    }

```

```

        amp:amp-type-item "action" { type "MC"; }
    }
    typedef "SRL" {
        type "binary";
        amp:amp-type-id 17;
        description
            "Identify and define a state-based macro rule.";

        amp:amp-type-item "id" { type "MID"; }
        amp:amp-type-item "start" { type "TS"; }
        // Mismatch in AMP spec for PRED type
        amp:amp-type-item "condition" { type "PRED"; }
        amp:amp-type-item "condition" { type "EXPR"; }
        amp:amp-type-item "count" { type "SDNV"; }
        amp:amp-type-item "action" { type "MC"; }
    }
    // Should be pure TDC with no type-id?
    // The only time [RPT] is used, the RPT type is unnecessary
    // because there is no alternative but RPT (i.e. TDC) data.
    typedef "RPT" {
        type "binary";
        amp:amp-type-id 19;
        description
            "Identify and define a report template.";

        // how is this different from TDC type + MID?
        amp:amp-type-item "id" { type "MID"; }
        amp:amp-type-item "entry-count" { type "SDNV"; }
    }

```

```

        // These really need not be BLOBs with internal sizes?
        amp:amp-type-item "entry-types" { type "BLOB"; }
        amp:amp-type-list "entry-values" { type "BLOB"; }
    }
    // May be useful to define a protocol-level CONFIGURE type which
    // looks similar to...
    //typedef CFG {
    //    amp:amp-type-item "target-id" { type "MID"; }
    //    amp:amp-type-list "value" { type "BLOB"; }
    //}
    // This would allow a simple macro of CFG values

    // Should be pure DEF with no type-id?
    // The only time MACRO is used is not for encoding, but for
    // typing objects in OID tree.
    typedef "MACRO" {
        type "DEF";
        amp:amp-type-id 20;
        description "Ordered list of control/macro MID values.";
    }

```

```

    typedef "UNK" {
        type "binary";
        amp:amp-type-id 21;
        description "Invalid type";
    }
}

```

<CODE ENDS>

[A.3.](#) AMP Extensions Submodule

The following YANG definition includes extensions specific to AMP.

```

<CODE BEGINS> file "amp-extensions.yang"
submodule amp-extensions {
    belongs-to amp-adm {
        prefix "amp";
    }

    organization
        "IETF Delay Tolerant Networking Working Group";
}

```

contact

"WG Web: <<http://tools.ietf.org/wg/dtn/>>

WG List: <<mailto:dtm@ietf.org>>

WG Chairs: Brian Haberman

<<mailto:brian@innovationslab.net>>

Marc Blanchet

<<mailto:Marc.Blanchet@viagenie.ca>>

Editor: Brian Sipos

<<mailto:BSIpos@rkf-eng.com>>;

description

"This submodule contains the set of core extensions necessary to
+ "define an Asynchronous Management Protocol data model.";

reference "draft Asynchronous Management Protocol";

revision "2016-04-01" {

description "Updated to fix typos.";

reference "I-D [draft-bsipos-dtn-amp-yang](#)";

}

revision "2016-03-14" {

description "Initial draft release.";

reference "I-D [draft-bsipos-dtn-amp-yang](#)";

}

extension fulloid {

Sipos & Birrane

Expires October 6, 2016

[Page 27]

Internet-Draft

A YANG profile for AMP ADMs

April 2016

argument "value";

description

"This extension defines the complete OID for the parent
+ "statement. ";

}

extension suboid {

argument "value";

description

"This extension defines a sub-OID of a statement relative
+ "to a parent-statement OID.";

}

extension nickname {

argument "id";

```

        description
            "A nickname is a single integer ID which correpsonds to
            +"full OID value.";
    }
    extension compressoid {
        argument "value";
        description
            "This extension allows using an ADM nickname within the
            +"ADM itself.";
    }

    extension "group" {
        argument "name";
        description
            "A logical grouping of ADM items under a parent OID.";
    }

    extension "primitive" {
        argument "name";
        description "A single typed value associated with an OID.";
    }

    extension "computed" {
        argument "name";
        description "A single typed value-expression associated with an
    }

    extension "report" {
        argument "name";
        description "Definition of a report within an ADM.";
    }
    extension "reportitem" {
        argument "name";
        description

```

```

        "A reference to a primitive within a report definition.
    }

    extension "control" {
        argument "name";
        description "Definition of a control within an ADM.";
    }

```



```

    extension "parameter" {
        argument "name";
        description
            "An individual parameter to a \"control\" statement. "
            +"Order of parameters is signifigant within a control."
    }
    extension "result" {
        argument "name";
        description
            "An individual result value reported as a response to "
            +"a \"control\" statement. "
            +"Order of results is signifigant within a control.";
    }
}

```

<CODE ENDS>

[A.4.](#) AMP Instances Submodule

The following YANG definition includes extensions to define AMP instance values.

```

<CODE BEGINS> file "amp-instances.yang"
submodule amp-instances {
    belongs-to amp-adm {
        prefix "amp";
    }

    organization
        "IETF Delay Tolerant Networking Working Group";
    contact
        "WG Web: <http://tools.ietf.org/wg/dtn/>
        WG List: <mailto:dtm@ietf.org>

        WG Chairs: Brian Haberman
                   <mailto:brian@innovationslab.net>
                   Marc Blanchet
                   <mailto:Marc.Blanchet@viagenie.ca>

        Editor: Brian Sipos
               <mailto:BSIpos@rkf-eng.com>";

```

```

description
    "This submodule contains the extensions necessary to "
    +"define AMP data instances directly in an ADM.";
reference "draft Asynchronous Management Protocol";

revision "2016-04-01" {
    description "Updated to fix typos.";
    reference "I-D draft-bsipos-dtn-amp-yang";
}
revision "2016-03-14" {
    description "Initial draft release.";
    reference "I-D draft-bsipos-dtn-amp-yang";
}

extension number-instance {
    argument "value";
    description
        "Instantiate a value of BYTE, INT, UINT, VAST, UVAST, "
        +"SDNV, REAL32, or REAL64 within an ADM.";
}
extension number-type {
    argument "name";
    description
        "The type name of a number-instance.";
}

extension string-instance {
    argument "value";
    description
        "Instantiate a value of STR from a text value.";
}

extension TS-instance {
    argument "value";
    description
        "Instantiate a value of TS from a text value. "
        +"The value is encoded according to RFC3339.";
}

extension MID-instance {
    description
        "Instantiate a value of MID from substatements "
        +"specializing the MID.";
    /// Must contain instance-identifer, amp:fulloid, or amp:compre
    /// May also contain DC-instance for parameterized OID
}
/// are type and cat necessary?
extension MID-type {

```

Internet-Draft

A YANG profile for AMP ADMs

April 2016

```
        argument "value";
        description "One of data, control, literal, or operator. "
            +"Default is data.";
    }
    extension MID-category {
        argument "value";
        description "One of atomic, computed, or collection. "
            +"Default is atomic.";
    }
    extension MID-issuer {
        argument "value";
        description "A numeric value identifying an issuer.";
    }
    extension MID-tag {
        argument "value";
        description "A numeric value identifying a tag.";
    }

    extension BLOB-instance {
        argument "value";
        description
            "Instantiate a value of BLOB from a text value. "
            +"The value is encoded in Base-64 per RFC3548. ";
    }

    extension DC-instance {
        description
            "Instantiate a value of DC from BLOB-instance substatement";
    }
    extension TDC-instance {
        description
            "Instantiate a value of TDC from *-instance substatement";
    }
    extension MC-instance {
        description
            "Instantiate a value of MC from MID-instance substatement";
    }

    /// Really is just MC-instance
    extension MACRO-instance {
        description
            "Instantiate a value of MACRO from MID-instance substatement";
    }
```

```
}
```

<CODE ENDS>

[Appendix B](#). Example Application Data Model

The following YANG definition includes extensions specific to AMP.

```
module example-adm {
  namespace "urn:example-adm";
  prefix "example-adm";

  import amp-adm {
    prefix "amp";
  }

  organization "Example Org.";
  description "Example module.";

  amp:fulloid "1.3.6.1.2.3.3";

  amp:nickname "3" {
    amp:fulloid "1.3.6.1.2";
  }

  amp:group "primitives" {
    amp:suboid "1";
    description "Primitive data available for getting or setting";

    amp:primitive "example" {
      amp:suboid "1";
      type "amp:UFAST";
      description "Example value.";
    }
  }

  amp:group "reports" {
    amp:suboid 3;

    amp:report showall {
```

```

        amp:suboid 8;

        amp:MC-instance {
            amp:MID-instance {
//                instance-identifier "/primitives/example
            }
        }
    }

    amp:group "controls" {
        amp:suboid "4";

```

```

description "Container for all commands in this ADM.";

amp:control "get" {
    amp:suboid "2";
    description "Get a single MIB value from the agent.";

    amp:parameter "corrid" {
        type "amp:SDNV";
        description "The correlation identifier for the
    }
    amp:parameter "object" {
        type "amp:MID";
        description "Identity of the object to retrieve
    }

    amp:result "corrid" {
        type "amp:SDNV";
        description "The correlation identifier for the
    }
    amp:result "errorcode" {
        type "amp:BYTE";
        description "If non-zero, an indicator of an error
    }
    amp:result "data" {
        type "amp:BLOB";
        description "Encoded value of the object.";
    }
}

```


Edward Birrane (editor)
Johns Hopkins University Applied Physics Laboratory
Email: Edward.Birrane@jhuapl.edu