Delay-Tolerant Networking
Internet-Draft
Intended status: Standards Track
Expires: 19 July 2026

E.J. Birrane
B. Sipos
J. Ethier
JHU/APL

15 January 2026

DTNMA Application Data Model (ADM) YANG Syntax
draft-ietf-dtn-adm-yang-06

Abstract

This document defines a concrete syntax for encoding a Delay-Tolerant Networking Management Architecture (DTNMA) Application Data Model (ADM) using the syntax and modular structure, but not the full data model, of YANG. Extensions to YANG are defined to capture the specifics needed to define DTNMA Application Management Model (AMM) objects and to use the Application Resource Identifier (ARI) data-value syntax.

Table of Contents

1. Introduction 3
1.1. Scope 4
1.2. Terminology 4
2. ADM Module Syntax 5
3. ADM Module Contents 6
4. Inherited YANG Module Processing 7
4.1. Direct Reuse 7
4.2. Restrictions and Exclusions 9
5. Built-In Types, TYPEDEFs, and Semantic Types 11
6. ADM Module Extensions 11
6.1. The amm:enum Statement 11
6.2. Semantic Type Statements 12
6.2.1. The amm:type Statement 12
6.2.2. The amm:ulist Statement 15
6.2.3. The amm:dlist Statement 17
6.2.4. The amm:umap Statement 18
6.2.5. The amm:tblt Statement 20
6.2.6. The amm:union Statement 21
6.2.7. The amm:seq Statement 22
6.3. The amm:parameter Statement 24
6.3.1. The amm:default Statement 25
6.4. The amm:typedef Statement 25
6.5. The amm:ident Statement 27
6.5.1. The amm:abstract Statement 29
6.5.2. The amm:base Statement 29
6.6. The amm:const Statement 30
6.7. The amm:ctrl Statement 32
6.7.1. The amm:result Statement 33
6.8. The amm:edd Statement 34
6.9. The amm:oper Statement 35
6.9.1. The amm:operand Statement 36
6.10. The amm:sbr Statement 37
6.10.1. The amm:min-interval Statement 38
6.10.2. The amm:max-count Statement 39
6.10.3. The amm:init-enabled Statement 39
6.11. The amm:tbr Statement 39
6.11.1. The amm:start Statement 40

6.12. The amm:var Statement 40
6.12.1. The amm:init-value Statement 42
7. ADM Author Considerations 42
8. IANA Considerations 42
8.1. DTN Management Architecture 42
9. Security Considerations 44
10. References 44
10.1. Normative References 44
10.2. Informative References 44
Appendix A. ADM Module for AMM Extensions 46
Appendix B. ADM Module for Semantic Type Introspection 53
Appendix C. ADM Module for AMM Semantic Types and Unions 59
Appendix D. ADM Module for Base Networking Objects 68
Appendix E. IANA Assignments for display-hint 73
Appendix F. ADM Module for DTNMA Agents 76
Appendix G. ADM Module for DTNMA Agent Access Control Lists . . 100
Implementation Status 108
Acknowledgments 108
Authors' Addresses 108

1. Introduction

The Delay-Tolerant Networking Management Architecture (DTNMA) of [RFC9675] defines a concept for remote management in a challenged network environment, and the Application Management Model (AMM) of [I-D.ietf-dtn-amm] defines a meta-model for the structure of object types and literal-value types used in DTNMA and separates static definitions of Application Data Models (ADMs) from dynamic Operational Data Models (ODMs).

This document defines a text representation of an ADM using the types and structures of the AMM combined with the syntax and processing semantics of YANG modules [RFC7950], while using AMM-specific extensions for object and data modeling. With this representation, individual applications can capture their static management information in module files and make use of existing YANG processing and module-level logic (_e.g._, file naming, revision history, imports).

Because the YANG syntax and data modeling language are closely coupled, there is no pre-existing example of using YANG for its syntax (and its module infrastructure) while using different domain-specific language (DSL) of extensions for data modeling. Separating the syntax from the data modeling is similar in concept to making use of ASN.1 syntax of [X.680] without using the SMIv2 data modeling language [RFC2578], which is how unrelated things like certificate profile of X.509 [RFC5280] use ASN.1 module syntax.

One way of thinking of this is that the ADM modules use a syntax that has "YANG characteristics" but does not make use of all YANG tooling or infrastructure and do not interoperate with existing YANG modules or their contained data models.

1.1. Scope

This document defines a specific syntax for representing individual revisions of individual ADMs as text files.

It does not define a representation for the runtime objects and literal values modeled by an ADM or ODM. Encodings of values within the AMM are defined in the Application Resource Identifier (ARI) specification [I-D.ietf-dtn-ari]. While the Agent ADM provides an introspection capability to indicate _which_ objects are present in an ODM, encodings of the runtime state of those objects are outside of the scope of this document.

It is not required that the YANG encoding be used for transmission of ADM information over the wire in the context of a network deployment. Since the AMM is designed to allow for multiple encodings, the expression of ADMs in YANG syntax is intended to support translation to other encodings without loss of information.

1.2. Terminology

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in [RFC2119].

The terms "Application Data Model", "Application Resource Identifier", "Operational Data Model", "Externally Defined Data", "Variable", "Constant", "Control", "Literal", "Macro", "Namespace", "Operator", "Report", "Report Template", "Rule", "State-Based Rule", "Table", and "Time-Based Rule" are used without modification from the definitions provided in [I-D.ietf-dtn-amm].

The terms "Comment", "Keyword", and "Module" are used without modification from the definitions provided in [RFC7950].

Additional terms defined in this document are as follows.

ADM Module: The specific use of a YANG module to represent a DTNMA ADM.

2. ADM Module Syntax

Some aspects of this ADM module profile restrict the allowable definitions to conform with Section 4 of [I-D.ietf-dtn-amm] and by doing so make YANG modules defining ADMs incompatible with YANG modules intended for NETCONF, RESTCONF, or other applications. Because of this, YANG modules defining ADMs SHALL be managed separately from the "YANG Module Names" registry of [IANA-YANG]. See the ADM registry defined in Section 9.3 of [I-D.ietf-dtn-ari] for registration of ADM modules. For the remainder of this document, a YANG module defining an ADM will be referred to as an "ADM module" and a YANG module for any other purpose will be referred to as an "Other Module" to differentiate it.

After explanation of extensions in Section 6, the following minimal ADM module will be expanded upon for further examples.

```
module example-adm {
  yang-version 1.1;
  namespace "ari://example/adm/";
  prefix example-adm;

  import ietf-amm {
    prefix amm;
  }

  organization
    "Example Org." {
    amm:enum 65535;
  }
  contact
    "Some User <user@example.org>";
  description
    "Example ADM module with YANG syntax.
     The organization name and enumeration are IANA reserved.";
  reference
    "draft-ietf-dtn-adm-yang";

  revision 2025-01-31 {
    description
      "Updated for latest ADM document.";
    reference
      "draft-ietf-dtn-adm-yang";
  }
  amm:enum 1;
}
```

Figure 1: Minimal Example ADM Module

3. ADM Module Contents

An ADM module is identified, as defined in Section 4.1, by the module "namespace" having an "ari" scheme. Within an ADM module, this profile makes restrictions in Section 4.2 which are formalized in the following table of allowed module substatements. This table is adapted from Section 7.1 of [RFC7950].

The "yang-version" of all ADM modules conforming to this document SHALL be "1.1". Later versions of YANG syntax could introduce incompatible changes and will need to be examined for consistency with the ADM module use.

Substatement	Cardinality	Reference
yang-version	1	Section 7.1.2 of [RFC7950]
namespace	1	Section 7.1.3 of [RFC7950]
prefix	1	Section 7.1.4 of [RFC7950]
include	0..n	Section 7.1.6 of [RFC7950]
import	0..n	Section 7.1.5 of [RFC7950]
status	0..1	Section 7.21.2 of [RFC7950]
reference	0..1	Section 7.21.4 of [RFC7950]
organization	0..1	Section 7.1.7 of [RFC7950]
description	0..1	Section 7.21.3 of [RFC7950]
revision	0..n	Section 7.1.9 of [RFC7950]
extension	0..n	Section 7.19 of [RFC7950]
feature	0..n	Section 7.20.1 of [RFC7950]
deviation	0..n	Section 7.20.3 of [RFC7950]
amm:enum	1	Section 6.1
amm:typedef	0..n	Section 6.4
amm:const	0..n	Section 6.6

amm:ctrl	0..n	Section 6.7
amm:edd	0..n	Section 6.8
amm:oper	0..n	Section 6.9
amm:sbr	0..n	Section 6.10
amm:tbr	0..n	Section 6.11
amm:var	0..n	Section 6.12

Table 1: ADM module Substatements

4. Inherited YANG Module Processing

The benefit of using the pre-existing YANG syntax is to take advantage of both tools that process YANG modules as well as some of the syntax and module-level semantics provided by YANG.

4.1. Direct Reuse

The following statements and behaviors of YANG are usable within an ADM with no modification:

File Layout: The existing file naming defined in Section 5.2 of [RFC7950] is unchanged for ADM modules. Because ADM modules occupy a separate ecosystem than Other Modules, their file stores SHALL be kept separate.

Modules and Submodules: The existing concepts and syntax of modules and submodules defined in Section 5.1 of [RFC7950] is unchanged for ADM modules. Because ADM modules occupy a separate ecosystem than Other Modules, there will not be any cross-imports between the two ecosystems. There is no harm in including an ADM module in an Other Module store, but it will have no data definitions usable with NETCONF, RESTCONF, _etc._

Module Naming: The existing module naming convention and recommendation from Section 4.1 of [RFC8407] is strengthened to a requirement for ADM modules. All ADM modules SHALL be named by concatenating from the ADM namespace reference: the organization ID, a dash "-", and the model ID.

Module Importing: The existing concept and syntax of importing a

module namespace, to reference objects in another module, defined in Section 5.1.1 of [RFC7950] is unchanged for ADM modules. Because the import includes a specific revision, the ARI references to objects in that imported module do not need any ADM revision information.

Module Prefix: The existing concept and syntax of the "prefix" statement defined in Section 7.1.4 of [RFC7950] is unchanged for ADM modules. The only aspect of ADM modules that the prefix applies to, however, are references to feature names from "if-feature" statements.

Module Namespace: Although an ADM module has no use for an XML namespace (as defined in Section 5.3 of [RFC7950]), the "namespace" statement is used to provide an explicit ARI namespace reference (see Section 3.4 of [I-D.ietf-dtn-ari]). For ADM modules, the "namespace" statement argument SHALL be the text-form ARI referencing the ADM itself.

For example, the ADM "example-adm" will have a namespace of ari://example/adm/ which is valid YANG but does not conform to the guidelines of Section 4.9 of [RFC8407].

Module Organization: The existing concept and use of the "organization" statement defined in Section 7.1.7 of [RFC7950] is unchanged for ADM modules. The organization statement is augmented for ADM modules by adding an amm:enum substatement as defined in Section 6.1.

Name Resolution: The existing name resolution logic described in Section 5.4 of [RFC7950] is unchanged for ADM modules. Because ADM modules are a flat object namespace, the complexity of namespaces is significantly simplified compared to Other Modules.

Features: The existing definition and use of features as described in Section 5.6.2 of [RFC7950] is unchanged for ADM modules. This includes the declaration of a "feature" statement within an ADM module and the conditioning of the presence of an object on an "if-feature" statement. This also treats feature identifiers as module-prefix-qualified references.

Documentation Statements: The existing status, description, and reference statements are unchanged from their definitions in Sections 7.21.2, 7.21.3, and Section 7.21.4 of [RFC7950] respectively.

Reusable Groups: The existing grouping and uses statements and logic

described in Section 4.2.6 of [RFC7950] is unchanged for ADM modules generally. Because ADM modules are a flat object namespace, the utility of reuse in an ADM module is limited to the contents of object definitions as shown in Table 2. See Section 7 for author considerations about the distinction between semantic type groupings and TYPEDEF instances.

Substatement	Cardinality
amm:parameter	0..*
amm:operand	0..*
amm:result	0..1
One of the semantic type:	
amm:type	0..1
amm:ulist	0..1
amm:dlist	0..1
amm:tblt	0..1
amm:umap	0..1
amm:union	0..1
amm:seq	0..1

Table 2: Allowed grouping Substatements

4.2. Restrictions and Exclusions

Because of the different interpretation of data definitions for an ADM module, the following restrictions are used to limit pre-existing valid YANG syntax within an ADM module:

Built-In Types: Because ADM modules have data that ultimately follows the AMM value model (see Section 3.1 of [I-D.ietf-dtn-amm]), the built-in types listed in Section 4.2.4 of [RFC7950], the "type" statement of Section 7.4 of [RFC7950], and the "typedef" statement of Section 7.3 of [RFC7950] do not directly apply to ADM modules. As defined in this document, there

is no direct deterministic mapping between Other Module typing and ADM module typing. An ADM module SHALL NOT contain any YANG built-in type or any "type" or "typedef" statements.

Data Node Structure: As defined in this document, there is no direct deterministic mapping between the complex, hierarchical, named data nodes of YANG and unnamed AMM values. An ADM module SHALL NOT contain any YANG data node statements, among them "container", "leaf", "leaf-list", "list", "anydata", "anyxml", "choice".

Configuration Versus State: Because of the different semantics of an ADM module from an Other Module, there is no concept of a labeling of some data as "configuration" and other as "state". An ADM module SHALL NOT contain any "config" statements.

Data Presence Versus Value: Rather than being an extrinsic notion of presence or absence, AMM objects can use type unions with the NULL type to indicate optional values and AMM values can use the null value to represent that state. In the case of formal parameters, the use of a "default" statement is used to indicate behavior when an actual parameter is undefined. In any case, the AMM value is always present when defined in the structure of an AMM object. An ADM module SHALL NOT contain any "mandatory" statements.

Nested Object Definitions: Because of the flat structure of an ADM, the nesting allowed and encouraged by Section 5.5 of [RFC7950], are not allowed to be present in an ADM module. An ADM module SHALL contain all AMM object definitions at the top (module) level.

XPath Expressions: The ADM module makes no use of XML or XPath in its definitions or logical constraints, so the behaviors described in Section 6.4 of [RFC7950] and in statements containing XPath expressions ("must", "when", "path", "augment") do not apply to ADM modules. All references in an ADM module take the form of an ARI.

NETCONF Operations: Because an ADM module will not be used for modeling in NETCONF and related protocols, the statements associated with NETCONF operations (notably "action", "notification", "rpc") SHALL NOT be present in an ADM module.

Extending Models: Because the transport of AMM values takes the form of an ARI, which does not include identity information the way XML elements do, the concept of extending an existing Legacy model by another model as described in Section 4.2.8 of [RFC7950] cannot apply to an ADM. The model extension "augment" statement SHALL NOT be present in an ADM module.

5. Built-In Types, TYPEDEFs, and Semantic Types

As discussed in Section 4.2, the YANG type and node models are fundamentally different from the AMM value model of Section 3.1 of [I-D.ietf-dtn-amm]. While YANG modules have been used to model data for XML representation for NETCONF (or equivalent representations for equivalent protocols), AMM values do not have explicit identifiers and have ARI representation.

Similar to how YANG treats built-in types, the AMM built-in type names are used without any namespace context. Differently to YANG, the AMM type names are encoded in ARI syntax and compared in a case-insensitive manner. One example of naming a built-in type UINT is below.

```
/ARITYPE/UINT
```

A TYPEDEF is a way for an ADM to apply a name and descriptive metadata to a specific semantic type (see Section 3.3 of [I-D.ietf-dtn-amm]). The TYPEDEF is also a top-level ADM object, as explained in Section 3.4.2 of [I-D.ietf-dtn-amm] with a syntax in Section 6.4. Because they are ADM objects, they can be referenced by ARIs when necessary as in the example below.

```
//example/adm-a/TYPEDEF/mytype
```

Except for the case of a simple "type" with no restrictions or annotations, anywhere one of the semantic type statements (Section 6.2) occurs in an object definition can be considered as an "anonymous" unnamed semantic type. The behavior in those cases is identical to a TYPEDEF just without a name associated with the semantic type.

6. ADM Module Extensions

In order to provide syntax necessary for AMM Object instance definitions this document defines, via the DTNMA ADM (Appendix A), the following extensions for ADM modules.

6.1. The amm:enum Statement

This statement is used to apply an integer enumeration to an ADM module, organization, or AMM object. Integer enumeration enables the compressed form of ARI discussed in Section 3.1 of [I-D.ietf-dtn-ari]. The argument to this statement is an integer in the YANG range 0..2^31-1 to fit within the ARI syntax. There are no substatements defined in this profile.

All well-known registered ADMs SHALL contain an amm:enum substatement within the top module statement and its organization statement. The value in the organization enumeration SHALL agree with the registration in the "Namespace Organizations" registry of [IANA-DTNMA].

6.2. Semantic Type Statements

The statements within this section enable the various semantic type classes, as explained in Section 3.3 of [I-D.ietf-dtn-amm]. They are all used as substatements within one of the AMM object definition statements, sometimes directly (_e.g._, for CONST or EDD typing) and sometimes indirectly (_e.g._, for parameter or operand typing), but always as a choice between the need for one of the semantic type statements.

6.2.1. The amm:type Statement

This statement creates a Named Type Use semantic type (see Section 3.3.1 of [I-D.ietf-dtn-amm]) for its parent statement. The argument to this statement is the text form of the ARI for the type being used, which can either be a built-in ARITYPE or a TYPEDEF object reference.

The substatements under this are annotations or constraints on the type being used. The units, range, length, and pattern statements have the same syntax and semantics as defined in [RFC7950] when applied to AMM values. For compatibility reasons, the arguments used for pattern statements SHALL be limited to the constructs compatible with I-Regexp of [RFC9485].

Substatement	Cardinality	Valid for These Types
description	0..1	any
reference	0..1	any
Display annotations		
units	0..1	Any NUMERIC
amm:display-hint	0..1	any
amm:int-labels	0..1	Any INTEGER type
Value constraints		
range	0..1	Any INTEGER or FLOAT, with appropriate quantized values in the argument
length	0..1	For TEXTSTR as number of characters. For BYTESTR or CBOR as number of bytes.
pattern	0..1	For TEXTSTR only
amm:cddl	0..1	For CBOR only
amm:base	0..*	For IDENT only

Table 3: amm:type Substatements

Examples of the "type" statement are below, some as simple use and some as restriction.

```
amm:type "/aritype/uint" {
  description
    "unconstrained";
}
amm:type "/aritype/uint" {
  description
    "constrained number";
  units "meter";
  range "3..10";
}
amm:type "/aritype/textstr" {
  description
    "constrained text size";
  length "10..100";
}
```

6.2.1.1. The amm:display-hint Statement

This statement provides general purpose tailoring of how AMM values are intended to be displayed. The argument to this statement is the text form of the ARI referencing an IDENT object derived from the root ari://ietf-amm/IDENT/display-hint object.

The description of any leaf IDENT derived from this root object SHALL contain specific requirements for how the display hint affects text form of any associated values, including restrictions on which built-in types the hint can be used with.

6.2.1.2. The amm:int-labels Statement

This statement provides human-friendly labels for enumerated values or bit positions within an integer type. There is no argument to this statement.

The substatements under this are either "enum" or "bit" statements from YANG, each of which define a single value or bit name within the type. A single "amm:int-labels" SHALL NOT use a mix of "enum" and "bit" substatements as this would result in ambiguous interpretation. When present, each "enum" statement SHALL contain an explicit and unique "value" substatement. When present, each "bit" statement SHALL contain an explicit and unique "position" substatement.

Substatement	Cardinality
enum	0..n
bit	0..n

Table 4: amm:int-labels Substatements

6.2.1.3. The amm:cddl Statement

This statement provides annotation of allowed content within a CBOR type byte string in the form of a Concise Data Definition Language (CDDL) fragment. The argument to this statement is a CDDL fragment in accordance with [RFC8610], escaped as necessary to conform to the YANG syntax.

There are no substatements within this statement.

6.2.1.4. The amm:base Statement

This statement constrains an IDENT (Section 6.5) object reference to allow only objects with a particular base themselves. The argument to this statement is the text form of the ARI referencing the required base object.

There are no substatements within this statement.

6.2.2. The amm:ulist Statement

This statement creates a Uniform List semantic type (see Section 3.3.2 of [I-D.ietf-dtn-amm]) for its parent statement. There is no argument to this statement.

Substatement	Cardinality
description	0..1
reference	0..1
min-elements	0..1
max-elements	0..1
uses	0..1
One of the semantic type:	
amm:type	0..1
amm:ulist	0..1
amm:dlist	0..1
amm:umap	0..1
amm:union	0..1

Table 5: amm:ulist Substatements

Examples of the "amm:ulist" statement are below, some as simple use and some as restriction.

```
amm:ulist {
  amm:type "/aritype/uint";
}
amm:ulist {
  amm:type "/aritype/uint" {
    range "3..10";
    units "meter";
  }
  min-elements 2;
  max-elements 10;
}
```

6.2.3. The amm:dlist Statement

This statement creates a Diverse List semantic type (see Section 3.3.3 of [I-D.ietf-dtn-amm]) for its parent statement. There is no argument to this statement.

Each of the type use substatements defines a single element within the AC, except for the amm:seq which defines a sequence of type-matching elements within the AC (rather than as a sub-AC).

Substatement	Cardinality
description	0..1
reference	0..1
One or more of the semantic type:	
amm:type	0..n
amm:ulist	0..n
amm:dlist	0..n
amm:umap	0..n
amm:union	0..n
amm:seq	0..n

Table 6: amm:dlist Substatements

Examples of the "amm:dlist" statement are below, one as simple structure and the other as a complex sequence.

```
amm:dlist {
  // each "type" substatement is one element of the AC
  amm:type "/aritype/uint";
  amm:type "/aritype/uint";
}
amm:dlist {
  // first AC element is text
  amm:type "/aritype/textstr";
  // remaining 2-10 element are int
  amm:seq {
    amm:type "/aritype/int";
    min-elements 2;
    max-elements 10;
  }
}
```

6.2.4. The amm:umap Statement

This statement creates a Uniform Map semantic type (see Section 3.3.4 of [I-D.ietf-dtn-amm]) for its parent statement. There is no argument to this statement.

When present, each of the amm:keys and amm:values substatements constrains the associated aspect of all map items to the specific type; when absent the aspect is left unconstrained. Keep in mind that AM keys are always constrained to untyped literal values and the use of amm:keys can only narrow this constraint.

Substatement	Cardinality
description	0..1
reference	0..1
amm:keys	0..1
amm:values	0..1

Table 7: amm:umap Substatements

Substatement	Cardinality
description	0..1
reference	0..1
uses	0..1
One of the semantic type:	
amm:type	0..1
amm:ulist	0..1
amm:dlist	0..1
amm:umap	0..1
amm:union	0..1

Table 8: amm:keys and amm:values Substatements

Examples of the "amm:umap" statement are below, some as simple use and some as restriction.

```
amm:umap {
  description
    "restrict only keys";
  amm:keys {
    amm:type "/aritype/uint";
  }
}
amm:umap {
  description
    "Map uint to text values.";
  amm:keys {
    amm:type "/aritype/uint";
  }
  amm:values {
    amm:type "/aritype/textstr";
  }
}
```

6.2.5. The amm:tblt Statement

This statement creates a Table Template semantic type (see Section 3.3.5 of [I-D.ietf-dtn-amm]) for its parent statement. There is no argument to this statement.

The substatements are a combination of "amm:column" statements defining table structure and "amm:key", "amm:unique", "min-elements", "max-elements" statements defining constraints on the table rows.

Substatement	Cardinality
description	0..1
reference	0..1
min-elements	0..1
max-elements	0..1
amm:key	0..1
amm:unique	0..n
amm:column	0..n

Table 9: amm:tblt Substatements

Examples of the "amm:tblt" statement are below.

```
amm:tblt {
  amm:column first {
    amm:type "/aritype/uint";
  }
  amm:column second {
    amm:type "/aritype/textstr";
  }
  amm:key "first";
}
```

6.2.5.1. The amm:column Statement

This statement defines the name and type of each column of a table. The argument to this statement is an identifier for the column. All columns SHALL have unique names within the same parent table.

The primary substatement under this are one of the semantic type statements (Section 6.2) which defines the type for values in this column. If present, any "if-feature" substatement within a column indicates that an Agent which does not implement the named feature(s) will: always use the undefined value in this column when producing tables associated with the parent template and/or always ignore (but not modify) given values in this column when processing these tables.

Substatement	Cardinality
if-feature	0..1
description	0..1
reference	0..1
uses	0..1
One of the semantic type:	
amm:type	0..1
amm:ulist	0..1
amm:dlist	0..1
amm:umap	0..1
amm:union	0..1

Table 10: amm:column Substatements

6.2.6. The amm:union Statement

This statement creates a Type Union [I-D.ietf-dtn-amm] semantic type for its parent statement. There is no argument to this statement.

Each of the substatements defines one of the possible choices of the union. The order of the list of choices is significant, especially when converting ARI values, because the first choice which matches or converts will be used regardless of any other choices.

Substatement	Cardinality
description	0..1
reference	0..1
Ordered choice of semantic types:	
amm:type	0..n
amm:ulist	0..n
amm:dlist	0..n
amm:tblt	0..n
amm:umap	0..n

Table 11: amm:union Substatements

Examples of the "union" statement are below, some as simple use and some as restriction.

```
amm:union {
  description
    "an optional int";
  amm:type "/aritype/int";
  amm:type "/aritype/null";
}
amm:union {
  amm:type "/aritype/uint" {
    range "3..10";
    units "meter";
  }
  amm:type "/aritype/textstr";
}
```

6.2.7. The amm:seq Statement

This statement creates a Sequence semantic type (see Section 3.3.7 of [I-D.ietf-dtn-amm]) for its parent statement. There is no argument to this statement.

This statement is distinct from the "amm:ulist" statement, which types the AC container itself, while the "amm:seq" types a portion of elements within an AC. The "amm:seq" is not present as a top-level value type except for the special case of a greedy capturing parameter type.

Substatement	Cardinality
description	0..1
reference	0..1
min-elements	0..1
max-elements	0..1
uses	0..1
One of the semantic type:	
amm:type	0..1
amm:ulist	0..1
amm:dlist	0..1
amm:umap	0..1
amm:union	0..1

Table 12: amm:seq Substatements

Examples of the "seq" statement are below.

```
amm:dlist {
  amm:type "/aritype/textstr";
  amm:seq {
    amm:type "/aritype/uint";
    min-elements 1;
  }
}
```

6.3. The amm:parameter Statement

This statement is used to define one formal parameter that apply to the parent object. The argument to this statement is an identifier for the parameter. All parameters SHALL have unique names within the same parent object.

The substatements under this are one of the semantic type statements (Section 6.2) which defines the type of the parameter. When the "amm:seq" statement is used to type a formal parameter, it SHALL be the last parameter in the list and represents a greedy matching of that an all subsequent given parameters.

Substatement	Cardinality
description	0..1
reference	0..1
amm:default	0..1
uses	0..1
One of the semantic type:	
amm:type	0..1
amm:ulist	0..1
amm:dlist	0..1
amm:tblt	0..1
amm:umap	0..1
amm:union	0..1
amm:seq	0..1

Table 13: amm:parameter Substatements

An example of the "parameter" statement is below, where there are three defined parameters two of which have default values.

```
amm:parameter first {
  amm:type "/aritype/uint" {
    range "3..10";
  }
}
amm:parameter second {
  amm:type "/aritype/textstr";
  amm:default "\"value\"";
}
amm:parameter third {
  amm:ulist {
    amm:type "/aritype/uint";
  }
  amm:default "/AC/(3,5,8)"
}
```

6.3.1. The amm:default Statement

This statement is used to define default parameter as an AMM value. The argument to this statement is the text form of the ARI for the default value.

The default value is used in the Parameter Handling procedure (see Section 6.4 of [I-D.ietf-dtn-amm]) to normalize given parameters into actual parameters.

6.4. The amm:typedef Statement

This statement is used to define a Semantic Type (TYPEDEF) object (see Section 3.4.2 of [I-D.ietf-dtn-amm]). The argument to this statement is the name of the object instance.

The substatements under this are a choice of a top-level semantic type (see Section 6.2) being named by the typedef.

Substatement	Cardinality
if-feature	0..1
amm:enum	0..1
status	0..1
description	0..1
reference	0..1
uses (for semantic type)	0..1
One of the semantic type:	
amm:type	0..1
amm:ulist	0..1
amm:dlist	0..1
amm:tblt	0..1
amm:umap	0..1
amm:union	0..1

Table 14: amm:typedef Substatements

Examples of the "typedef" statement are below, some as restriction and some as union.

```
amm:typedef restricted-uint {
  description
    "Example restriction";
  amm:type "/aritype/uint" {
    range "3..10";
  }
}
amm:typedef annotated-uint {
  description
    "Example units annotation";
  amm:type "/aritype/uint" {
    units "meter";
  }
}
amm:typedef structured-ac {
  description
    "Example units on AC items";
  amm:ulist {
    amm:type "/aritype/real32";
    amm:units "meter";
    min-elements 3;
    max-elements 10;
  }
}
```

6.5. The amm:ident Statement

This statement is used to define an Identity (IDENT) object (see Section 3.4.3 of [I-D.ietf-dtn-amm]). The argument to this statement is the name of the object instance.

The substatements under this are the common object metadata, optional formal parameters, and any number of base IDENT objects from which this one is derived.

Substatement	Cardinality
if-feature	0..1
amm:enum	0..1
amm:parameter	0..n
status	0..1
description	0..1
reference	0..1
amm:abstract	0..1
amm:base	0..*
uses (for parameters)	0..1

Table 15: amm:ident Substatements

An example of a multiple-base hierarchy of IDENT objects is below.

```
amm:ident base-a {
  amm:abstract true;
  description
    "A base identity object.";
}
amm:ident base-b {
  amm:abstract true;
  description
    "Another base identity object.";
}
amm:ident use-1 {
  amm:base "./IDENT/base-a";
  description
    "An identity derived from base-a.";
}
amm:ident use-2 {
  amm:base "./IDENT/use-1";
  amm:base "./IDENT/base-b";
  description
    "An identity derived from base-a (via use-1) and base-b.";
}
amm:ident use-3 {
  amm:base "./IDENT/base-b";
  amm:parameter option {
    amm:type "/aritype/int";
  }
  description
    "An identity derived from base-b which includes a parameter.";
}
```

6.5.1. The amm:abstract Statement

This statement causes the parent IDENT (Section 6.5) object to be explicitly marked as abstract or not, as defined in Section 3.4.3 of [I-D.ietf-dtn-amm]. The argument to this statement is a boolean ("true" or "false") value in accordance with Section 9.5 of [RFC7950]. If not present, the default marking of the IDENT object is not-abstract.

There are no substatements within this statement.

6.5.2. The amm:base Statement

This statement causes the parent IDENT (Section 6.5) object to be derived from a referenced base object. The argument to this statement is the text form of the ARI referencing the required base object.

Similar to YANG logic, in the AMM derivation of an IDENT from a base has the following properties.

* It is irreflexive, which means that an identity is not derived from itself.

* It is transitive, which means that if identity B is derived from A and C is derived from B, then C is also derived from A.

There are no substatements within this statement.

6.6. The amm:const Statement

This statement is used to define a Constant (CONST) object (see Section 3.4.5 of [I-D.ietf-dtn-amm]). The argument to this statement is the name of the object instance.

The substatements under this are the common object metadata, optional formal parameters, and one of the semantic type statements (Section 6.2) that represents the produced value of the CONST. The amm:const statement SHALL contain a an amm:init-value, which is the constant represented by this object.

Substatement	Cardinality
if-feature	0..1
amm:enum	0..1
amm:parameter	0..n
status	0..1
description	0..1
reference	0..1
amm:init-value	1
uses (for parameters and semantic type)	0..1
One of the semantic type:	
amm:type	0..1
amm:ulist	0..1
amm:dlist	0..1
amm:tblt	0..1
amm:umap	0..1
amm:union	0..1

Table 16: amm:const Substatements

An example of a simple-typed CONST is below.

```
amm:const pi32 {
  amm:type "/aritype/real32";
  amm:init-value "3.14159";
  description
    "A truncated value of Pi.";
}
```

Another example of a semantic-typed MAC-valued CONST (see Section 4.2.4 of [I-D.ietf-dtn-amm]) is below.

```
amm:const do_thing {
  amm:type "//ietf-amm/typedef/mac";
  amm:init-value "/AC/(../CTRL/first,../CTRL/second(2))";
  description
    "Execute two controls in sequence.";
}
```

6.7. The amm:ctrl Statement

This statement is used to define a Control (CTRL) object (see Section 3.4.6 of [I-D.ietf-dtn-amm]). The argument to this statement is the name of the object instance.

The substatements under this are the common object metadata, optional formal parameters, and an optional execution result. If the amm:result substatement is present it SHALL be used to constrain Result Storage for the execution procedure (see Section 6.6 of [I-D.ietf-dtn-amm]). If the amm:result substatement is omitted the assumed result type SHALL be NULL with a default value of null. Regardless of the presence or type indicated by an amm:result substatement, the failure to execute any control always results in an undefined result value.

Substatement	Cardinality
if-feature	0..1
amm:enum	0..1
amm:parameter	0..n
status	0..1
description	0..1
reference	0..1
amm:result	0..1
uses (for parameters and result)	0..1

Table 17: amm:ctrl Substatements

An example of a single-parameter CTRL is below.

```
    amm:ctrl reset_count {
      amm:parameter src {
        type textstr;
        description
          "The name of the source.";
      }
      amm:result previous {
        type UVAST;
        description
          "The value just before reset.";
      }
      description
        "This control resets counts for the given source.";
    }
```

6.7.1. The amm:result Statement

The result statement contains one of the semantic type statements (Section 6.2) which represents the result expected from executing a CTRL or evaluating an OPER. The argument to this statement is an identifier for the result.

Substatement	Cardinality
description	0..1
reference	0..1
uses (for semantic type)	0..1
One of the semantic type:	
amm:type	0..1
amm:ulist	0..1
amm:dlist	0..1
amm:tblt	0..1
amm:umap	0..1
amm:union	0..1

Table 18: amm:result Substatements

6.8. The amm:edd Statement

This statement is used to define an Externally Defined Data (EDD) object (see Section 3.4.4 of [I-D.ietf-dtn-amm]). The argument to this statement is the name of the object instance.

The substatements under this are the common object metadata, optional formal parameters, and one of the semantic type statements (Section 6.2) that represents the produced value of the EDD.

Substatement	Cardinality
if-feature	0..1
amm:enum	0..1
amm:parameter	0..n
status	0..1
description	0..1
reference	0..1
uses (for parameters and semantic type)	0..1
One of the semantic type:	
amm:type	0..1
amm:ulist	0..1
amm:dlist	0..1
amm:tblt	0..1
amm:umap	0..1
amm:union	0..1

Table 19: amm:edd Substatements

An example of a simple-typed EDD is below.

```
amm:edd tx_count {
  amm:type "//ietf-amm/typedef/counter64" {
    units "frames";
  }
  description
    "The count of the number of frames sent.";
}
```

6.9. The amm:oper Statement

This statement is used to define an Operator (OPER) object (see Section 3.4.7 of [I-D.ietf-dtn-amm]). The argument to this statement is the name of the object instance.

The substatements under this are the common object metadata, optional formal parameters, an operands list, and an evaluation result.

Substatement	Cardinality
if-feature	0..1
amm:enum	0..1
amm:parameter	0..n
status	0..1
description	0..1
reference	0..1
amm:operand	0..*
result	1
uses (for parameters, operands, and/or result)	0..1

Table 20: amm:edd Substatements

An example of an arithmetic OPER is below.

```
amm:oper add {
  amm:operand val-a {
    amm:type "//ietf-amm/typedef/numeric";
  }
  amm:operand val-b {
    amm:type "//ietf-amm/typedef/numeric";
  }
  amm:result sum {
    amm:type "//ietf-amm/typedef/numeric";
  }
  description
    "Sum together the two operands.";
}
```

A more complex "variadic" OPER with parameters is below. Also note the anonymous semantic type used to restrict the parameter.

```
amm:oper sum {
  description
    "Sum together a sequence from the stack.";
  parameter count {
    description
      "The number of operands to pop."
    amm:type "/aritype/uint" {
      range "1..max";
    }
  }
  operand value {
    description
      "This is not within a container, so does not represent an AC;
      it is multiple operands.";
    amm:type "//ietf-amm/typedef/numeric";
  }
  result sum {
    amm:type "//ietf-amm/typedef/numeric";
  }
}
```

6.9.1. The amm:operand Statement

This statement is used to define one operand that applies to the parent object. The argument to this statement is an identifier for the operand. All operands SHALL have unique names within the same parent object.

Substatement	Cardinality
description	0..1
reference	0..1
uses (for semantic type)	0..1
One of the semantic type:	
amm:type	0..1
amm:ulist	0..1
amm:dlist	0..1
amm:tblt	0..1
amm:umap	0..1
amm:union	0..1

Table 21: amm:operand Substatements

6.10. The amm:sbr Statement

This statement is used to define a State Based Rule object (see Section 3.4.8 of [I-D.ietf-dtn-amm]). The argument to this statement is the name of the object instance.

The substatements under this are the common object metadata, the action to execute upon trigger, and rule control fields defined below.

Substatement	Cardinality
if-feature	0..1
amm:enum	0..1
status	0..1
description	0..1
reference	0..1
amm:action	1
amm:condition	1
amm:min-interval	0..1
amm:max-count	0..1
amm:init-enabled	0..1

Table 22: amm:sbr Substatements

An example of an SBR with some default fields is below.

```
amm:sbr enable_safe_mode {
  description
    "Enable safe mode below threshold.";
  amm:action "/AC/(./CTRL/ensure_safe_mode(true))";
  amm:condition "/AC/(./EDD/sensor,./VAR/min_threshold,"
    +"/ietf-amm/OPER/compare_lt)";
}
```

6.10.1. The amm:min-interval Statement

This statement is used to define the Minimum Interval field of SBR (Section 6.10) objects. The argument to this statement is the text form of a TD value. If not present, the default value of zero (meaning no minimum) is used.

6.10.2. The amm:max-count Statement

This statement is used to define the Maximum Count field of SBR (Section 6.10) and TBR (Section 6.11) objects. The argument to this statement is the text form of a UVAST value. If not present, the default value of zero (meaning no limit) is used.

6.10.3. The amm:init-enabled Statement

This statement is used to define the Initial Enabled state of SBR (Section 6.10) and TBR (Section 6.11) objects. The argument to this statement is a boolean ("true" or "false") value in accordance with Section 9.5 of [RFC7950]. If not present, the default value of true is used.

6.11. The amm:tbr Statement

This statement is used to define a Time-Based Rule (TBR) object (see Section 3.4.9 of [I-D.ietf-dtn-amm]). The argument to this statement is the name of the object instance.

The substatements under this are the common object metadata, the action to execute upon trigger, and rule control fields defined below.

Substatement	Cardinality
if-feature	0..1
amm:enum	0..1
status	0..1
description	0..1
reference	0..1
amm:action	1
amm:start	1
amm:period	1
amm:max-count	0..1
amm:init-enabled	0..1

Table 23: amm:tbr Substatements

An example of an TBR with some default fields is below.

```
amm:tbr tlm_rule {
  description
    "Generate telemetry reports.";
  amm:action "/AC/(./CTRL/first,/adm2/CTRL/other)";
  amm:period "/TD/PT30s";
}
```

6.11.1. The amm:start Statement

This statement is used to define the Start field of TBR (Section 6.11) objects. The argument to this statement is the text form of a TIME value (see Appendix A). If not present, the default value of zero (meaning start immediately) is used.

6.12. The amm:var Statement

This statement is used to define a Variable (VAR) object (see Section 3.4.10 of [I-D.ietf-dtn-amm]). The argument to this statement is the name of the object instance.

The substatements under this are the common object metadata, optional formal parameters, one of the semantic type statements (Section 6.2) that represents the stored-and-produced value of the VAR, and an optional initializer expression.

Substatement	Cardinality
if-feature	0..1
amm:enum	0..1
amm:parameter	0..n
status	0..1
description	0..1
reference	0..1
amm:init-value	0..1
uses (for parameters and semantic type)	0..1
One of the semantic type:	
amm:type	0..1
amm:ulist	0..1
amm:dlist	0..1
amm:tblt	0..1
amm:umap	0..1
amm:union	0..1

Table 24: amm:var Substatements

An example of a simple-typed VAR with an initializer value is below.

```
amm:var min_threshold {
  description
    "The lower threshold to enable safe mode.";
  amm:type "/aritype/real32" {
    units "meter";
  }
  amm:init-value "1e4";
}
```

6.12.1. The amm:init-value Statement

This statement is used to define CONST values and VAR initial state as an AMM value. The argument to this statement is the text form of the ARI for the initial value.

The initializer is used as part of the Agent Initialization procedure, (see Section 6.1 of [I-D.ietf-dtn-amm]) and when new VAR objects are defined in an ODM.

7. ADM Author Considerations

All of the logical design considerations from Section 7 of [I-D.ietf-dtn-amm] apply to authors of ADM modules when considering the naming and behaviors of AMM object instances.

All of the lifecycle considerations for YANG modules from [RFC8407] apply to those statements reused for ADM modules as defined in Section 4.1. Specifically the guidance from Section 4.17 of [RFC8407] regarding feature use and scoping.

Although this specification allows semantic type statements to be included in a grouping statement, it is RECOMMENDED that ADM authors make use of named TYPEDEF (Section 6.4) object where possible.

8. IANA Considerations

This section provides guidance to the Internet Assigned Numbers Authority (IANA) regarding registration of schema and namespaces related to core ADMs, in accordance with BCP 26 [RFC8126].

8.1. DTN Management Architecture

This document relies on existing ARI-defined registries defined in [IANA-DTNMA] by Section 9.3 of [I-D.ietf-dtn-ari].

This document defines the following entries within the "DTNMA IETF Data Models" registry of the "DTN Management Architecture" registry group [IANA-DTNMA]. All of the ADMs in this registry are under the "ietf" organization (enumeration 1).

Enumeration	Name	Reference	Notes
0	amm	[This document]	Definition of YANG extensions for the AMM itself.
1	dtnma-agent	[This document]	DTNMA Agent introspection objects.
2	dtnma-agent-acl	[This document]	DTNMA Agent access control.
24	amm-semtype	[This document]	AMM-defined semantic type representation.
25	amm-base	[This document]	AMM-defined type unions and other definitions.
26	network-base	[This document]	General purpose and abstract network-related objects.

Table 25: DTNMA IETF Data Models

This document defines the following entries within the "DTNMA IANA Data Models" registry of the "DTN Management Architecture" registry group [IANA-DTNMA]. All of the ADMs in this registry are under the "iana" organization (enumeration 2).

Enumeration	Name	Reference	Notes
0	display-hints	[This document]	Registry of well-known display hints.

Table 26: DTNMA IANA Data Models

9. Security Considerations

This document defines a syntax for encoding DTNMA ADMs within a text file, and for interpreting extensions to the YANG language to support modeling of AMM objects and ARI values. It does not address what the contents of those ADMs are or any security considerations associated with the data models themselves.

10. References

10.1. Normative References

[IANA-DTNMA]
IANA, "Delay-Tolerant Networking Management Architecture (DTNMA) Parameters", <https://www.iana.org/assignments/TBA/>.

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

[RFC7950] Bjorklund, M., Ed., "The YANG 1.1 Data Modeling Language", RFC 7950, DOI 10.17487/RFC7950, August 2016, <https://www.rfc-editor.org/info/rfc7950>.

[RFC9485] Bormann, C. and T. Bray, "I-Regexp: An Interoperable Regular Expression Format", RFC 9485, DOI 10.17487/RFC9485, October 2023, <https://www.rfc-editor.org/info/rfc9485>.

[I-D.ietf-dtn-amm]
Birrane, E. J., Sipos, B., and J. Ethier, "DTNMA Application Management Model (AMM) and Data Models", Work in Progress, Internet-Draft, draft-ietf-dtn-amm-00, 3 July 2024, <https://datatracker.ietf.org/doc/html/draft-ietf-dtn-amm-00>.

[I-D.ietf-dtn-ari]
Birrane, E. J., Annis, E., and B. Sipos, "DTNMA Application Resource Identifier (ARI)", Work in Progress, Internet-Draft, draft-ietf-dtn-ari-00, 22 February 2024, <https://datatracker.ietf.org/doc/html/draft-ietf-dtn-ari-00>.

10.2. Informative References

[IANA-YANG] IANA, "YANG Parameters", <https://www.iana.org/assignments/yang-parameters/>.

[X.680] ITU-T, "Information technology -- Abstract Syntax Notation One (ASN.1): Specification of basic notation", ITU-T Recommendation X.680, ISO/IEC 8824-1:2015, August 2015, <https://www.itu.int/rec/T-REC-X.680-201508-I/en>.

[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, <https://www.rfc-editor.org/info/rfc2578>.

[RFC5280] Cooper, D., Santesson, S., Farrell, S., Boeyen, S., Housley, R., and W. Polk, "Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL) Profile", RFC 5280, DOI 10.17487/RFC5280, May 2008, <https://www.rfc-editor.org/info/rfc5280>.

[RFC7942] Sheffer, Y. and A. Farrel, "Improving Awareness of Running Code: The Implementation Status Section", BCP 205, RFC 7942, DOI 10.17487/RFC7942, July 2016, <https://www.rfc-editor.org/info/rfc7942>.

[RFC8126] Cotton, M., Leiba, B., and T. Narten, "Guidelines for Writing an IANA Considerations Section in RFCs", BCP 26, RFC 8126, DOI 10.17487/RFC8126, June 2017, <https://www.rfc-editor.org/info/rfc8126>.

[RFC8407] Bierman, A., "Guidelines for Authors and Reviewers of Documents Containing YANG Data Models", BCP 216, RFC 8407, DOI 10.17487/RFC8407, October 2018, <https://www.rfc-editor.org/info/rfc8407>.

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

[RFC9675] Birrane, III, E., Heiner, S., and E. Annis, "Delay-Tolerant Networking Management Architecture (DTNMA)", RFC 9675, DOI 10.17487/RFC9675, November 2024, <https://www.rfc-editor.org/info/rfc9675>.

[github-dtnma-ace]
JHU/APL, "The DTNMA AMM CODEC Engine (ACE)",
<https://github.com/JHUAPL-DTNMA/dtnma-ace>.

[github-dtnma-adms]
JHU/APL, "Application Data Models (ADMs)",
<https://github.com/JHUAPL-DTNMA/dtnma-adms>.

Appendix A. ADM Module for AMM Extensions

The ADM module in this section implements base extensions to YANG in support of the AMM. This includes the following:

* Extensions to support ADM Metadata (see Section 4.1.1 of [I-D.ietf-dtn-amm]) within a YANG module.

* Extensions to support AMM Object Types (see Section 3.4 of [I-D.ietf-dtn-amm]) within a YANG module.

* Extensions to support Semantic Value Types (see Section 3.3 of [I-D.ietf-dtn-amm]) within a YANG module.

<CODE BEGINS> file "ietf-amm.yang"

```
module ietf-amm {
  yang-version 1.1;
  namespace "ari://ietf/amm/";
  prefix amm;

  organization
    "Internet Engineering Task Force (IETF)" {
    amm:enum 1;
  }
  contact
    "WG Web: <http://tools.ietf.org/wg/dtn/>
     WG List: <mailto:dtn@ietf.org>

     Editor: Brian Sipos
             <mailto:brian.sipos+ietf@gmail.com>";
  description
    "This module defines the DTN Management Architecture (DTNMA)
     Application Management Model (AMM) extensions within YANG.
     It also defines the base IDENT and TYPEDEF objects for the AMM.

     The key words 'MUST', 'MUST NOT', 'REQUIRED', 'SHALL', 'SHALL
     NOT', 'SHOULD', 'SHOULD NOT', 'RECOMMENDED', 'NOT RECOMMENDED',
     'MAY', and 'OPTIONAL' in this document are to be interpreted as
     described in BCP 14 (RFC 2119) (RFC 8174) when, and only when,
     they appear in all capitals, as shown here.
```

```
   Copyright (c) 2025 IETF Trust and the persons identified as
   authors of the code.  All rights reserved.

   Redistribution and use in source and binary forms, with or
   without modification, is permitted pursuant to, and subject to
   the license terms contained in, the Revised BSD License set
   forth in Section 4.c of the IETF Trust's Legal Provisions
   Relating to IETF Documents
   (https://trustee.ietf.org/license-info).

   This version of this YANG module is part of RFC XXXX
   (https://www.rfc-editor.org/info/rfcXXXX); see the RFC itself
   for full legal notices.";
 reference
   "draft-ietf-dtn-adm-yang";

 revision 2025-07-03 {
   description
     "Updated for latest ADM document.";
   reference
     "draft-ietf-dtn-adm-yang";
 }

 /*
  * This section contains extensions for common behavior of AMM
  * objects.
  */

 extension enum {
   argument value;
   description
     "An enumeration identifies an object within a namespace.
      The argument to this statement is the integer value.";
 }

 extension parameter {
   argument name;
   description
     "The schema for a formal parameter of an object.
      Order of parameters is significant within this statement.";
 }

 extension default {
   argument value;
   description
     "The literal value default for a parameter or result.
      The argument is the text form of the ARI";
 }
```

```
  extension init-value {
    argument value;
    description
      "The literal value of a CONST object,
       initial state for a VAR object.
       The argument is the text form of the ARI";
  }

  extension init-expr {
    argument expr;
    description
      "An expression evaluated to initialize a VAR object.
       The argument is the text form of the EXPR AC";
  }

  /*
   * This section contains extensions for AMM object definitions.
   */

  extension typedef {
    argument name;
    description
      "Definition of a TYPEDEF within an ADM.
       The argument to this statement is the object name.
       One of the type use substatements must be present.";
  }

  extension ident {
    argument name;
    description
      "Definition of an IDENT within an ADM.
       The argument to this statement is the object name.
       An 'base' substatement must be present.";
  }

  extension abstract {
    argument is-abstract;
    description
      "The boolean abstract marking of an IDENT object.";
  }

  extension base {
    argument name;
    description
      "The argument is the base of an IDENT object.";
  }

  extension const {
```

```
      argument name;
      description
        "Definition of a CONST within an ADM.
         The argument to this statement is the object name.
         An 'init-value' substatement must be present.";
    }

    extension ctrl {
      argument name;
      description
        "Definition of a CTRL within an ADM.
         The argument to this statement is the object name.";
    }

    extension result {
      argument name;
      description
        "A result value reported as a response to a control.
         The substatement is the result type.
         Each CTRL can have a single optional result.";
    }

    extension edd {
      argument name;
      description
        "Definition of an EDD within an ADM.
         The argument to this statement is the object name.";
    }

    extension oper {
      argument name;
      description
        "Definition of an OPER within an ADM.
         The argument to this statement is the object name.";
    }

    extension operand {
      argument name;
      description
        "An individual operand taken from the expression stack
         during evaluation of the OPER.
         Each substatement is an operand as a leaf (ARI).
         The order of operands is significant within an object
         definition.";
    }

    extension var {
      argument name;
```

```
    description
      "Definition of a VAR within an ADM.
       The argument to this statement is the object name.";
  }

  extension sbr {
    argument name;
    description
      "Definition of a SBR within an ADM.
       The argument to this statement is the object name.";
  }

  extension action {
    argument exec-tgt;
    description
      "Definition of the action executed by a rule object.
       The argument to this statement is the exec-tgt to execute.";
  }

  extension condition {
    argument expr;
    description
      "Condition evaluated by an SBR object to determine whether to
       execute its action.
       The argument to this statement is the expr to evaluate.";
  }

  extension min-interval {
    argument relative-time;
    description
      "The argument to this statement is a time-based ARI specifying
       the minimum amount of time between condition evaluations
       for an SBR object.";
  }

  extension max-count {
    argument count;
    description
      "The argument to this statement is the integer maximum
       execution count for a rule object.";
  }

  extension init-enabled {
    argument state;
    description
      "The argument to this statement is the boolean initial enabled
       state for a rule object.";
  }
```

```
     extension tbr {
       argument name;
       description
         "Definition of a TBR within an ADM.
          The argument to this statement is the object name.";
     }

     extension start {
       argument time;
       description
         "The argument to this statement is the start time ARI for a
          TBR object.";
     }

     extension period {
       argument relative-time;
       description
         "The argument to this statement is an ARI defining the period
          for a TBR object.";
     }

     /*
      * This section contains extensions for defining semantic type
      * instances.
      */

     extension type {
       argument name;
       description
         "A reference to a built-in type or prefix-qualified typedef.";
     }

     extension display-hint {
       argument ident;
       description
         "Reference an IDENT object which indicates how a value should
          be displayed for human consumption.
          The base of the IDENT SHALL be the
          <ari://ietf/amm-base/ident/display-hint> object.";
     }

     extension int-labels {
       description
         "Type narrowing for an integer to label enum values or bit
          positions.";
     }

     extension cddl {
```

```
    argument text;
    description
      "Type narrowing for a CBOR item in the form of CDDL syntax.
       The argument to this statement is the actual CDDL text.";
  }

  extension ulist {
    description
      "Type for an AC containing a list of uniform-typed values.";
  }

  extension dlist {
    description
      "Type for an AC containing a list of diverse-typed sequences.";
  }

  extension seq {
    description
      "A sequence of uniform-typed values within a 'dlist'.";
  }

  extension umap {
    description
      "Type for an AM containing a set of uniform-typed key-value
       pairs.";
  }

  extension keys {
    description
      "The type restriction for 'umap' keys.";
  }

  extension values {
    description
      "The type restriction for 'umap' values.";
  }

  extension tblt {
    description
      "Type for a table of values with a column schema.";
  }

  extension column {
    argument name;
    description
      "An individual column within a 'tblt' definition.";
  }
```

```
    extension key {
      argument columns;
      description
        "Names of columns composing the row key, separated by spaces,
         within a 'tblt' definition.";
    }

    extension unique {
      argument columns;
      description
        "Names of columns composing a uniqueness constraint, separated
         by spaces, within a 'tblt' definition.";
    }

    extension union {
      description
        "Semantic type for a union of other types in substatements.";
    }

    amm:enum 0;
  }
  <CODE ENDS>
```

Appendix B. ADM Module for Semantic Type Introspection

The ADM module in this section implements the entities described in Section 4.2.2 of [I-D.ietf-dtn-amm]. This includes an IDENT object hierarchy which allows introspection of semantic types in ARI syntax.

```
<CODE BEGINS> file "ietf-amm-semtype.yang"
module ietf-amm-semtype {
  yang-version 1.1;
  namespace "ari://ietf/amm-semtype/";
  prefix amm-semtype;

  import ietf-amm {
    prefix amm;
  }

  organization
    "Internet Engineering Task Force (IETF)" {
    amm:enum 1;
  }
  contact
    "WG Web: <http://tools.ietf.org/wg/dtn/>
     WG List: <mailto:dtn@ietf.org>

     Editor: Brian Sipos
```

```
            <mailto:brian.sipos+ietf@gmail.com>";
    description
      "This module defines base and derived IDENT objects which
       allow introspection of AMM semantic types.

       The key words 'MUST', 'MUST NOT', 'REQUIRED', 'SHALL', 'SHALL
       NOT', 'SHOULD', 'SHOULD NOT', 'RECOMMENDED', 'NOT RECOMMENDED',
       'MAY', and 'OPTIONAL' in this document are to be interpreted as
       described in BCP 14 (RFC 2119) (RFC 8174) when, and only when,
       they appear in all capitals, as shown here.

       Copyright (c) 2025 IETF Trust and the persons identified as
       authors of the code.  All rights reserved.

       Redistribution and use in source and binary forms, with or
       without modification, is permitted pursuant to, and subject to
       the license terms contained in, the Revised BSD License set
       forth in Section 4.c of the IETF Trust's Legal Provisions
       Relating to IETF Documents
       (https://trustee.ietf.org/license-info).

       This version of this YANG module is part of RFC XXXX
       (https://www.rfc-editor.org/info/rfcXXXX); see the RFC itself
       for full legal notices.";
    reference
      "draft-ietf-dtn-adm-yang";

    revision 2025-07-03 {
      description
        "Updated for latest ADM document.";
      reference
        "draft-ietf-dtn-adm-yang";
    }
    amm:enum 24;

    // Base IDENT objects
    amm:ident semtype {
      amm:enum 1;
      description
        "The base IDENT for semantic type introspection.
         This object itself provides no type information.";
    }

    // Semantic-type leaf IDENT objects
    amm:ident type-use {
      amm:enum 2;
      description
        "A named type use with possibe annotations and/or
```

```
         constraints.";
      amm:base "./IDENT/semtype";
      amm:parameter name {
        description
          "A built-in ARITYPE or reference to TYPEDEF object being
           used.";
        amm:type "//ietf/amm-base/TYPEDEF/type-ref";
      }
      // FIXME include constraints and annotations also?
    }
    amm:ident ulist {
      amm:enum 3;
      description
        "A uniformly-typed list within an AC.";
      amm:base "./IDENT/semtype";
      amm:parameter item-type {
        description
          "The semantic type for all items.";
        amm:type "./TYPEDEF/semtype";
      }
      amm:parameter min-elements {
        description
          "The smallest valid item count for the AC.";
        amm:default "null";
        amm:union {
          amm:type "/ARITYPE/uvast";
          amm:type "/ARITYPE/null";
        }
      }
      amm:parameter max-elements {
        description
          "The largest valid item count for the AC.";
        amm:default "null";
        amm:union {
          amm:type "/ARITYPE/uvast";
          amm:type "/ARITYPE/null";
        }
      }
    }
    amm:ident dlist {
      amm:enum 4;
      description
        "A diverse-typed list within an AC.";
      amm:base "./IDENT/semtype";
      amm:parameter item-types {
        description
          "The list of semantic type for each item in sequence within
           the list.
```

```
           The list itself is encoded as an AC in this parameter.";
        amm:ulist {
          amm:type "./TYPEDEF/semtype";
        }
      }
    }
    amm:ident umap {
      amm:enum 5;
      description
        "A uniformly-typed map within an AM.";
      amm:base "./IDENT/semtype";
      amm:parameter key-type {
        description
          "The semantic type for each key.";
        amm:type "./TYPEDEF/semtype";
      }
      amm:parameter value-type {
        description
          "The semantic type for each value.";
        amm:type "./TYPEDEF/semtype";
      }
    }
    amm:ident tblt {
      amm:enum 6;
      description
        "A parameterized table template for data within a TBL.";
      amm:base "./IDENT/semtype";
      amm:parameter columns {
        description
          "The list of column definitions as rows in a table.
           The order of these definitions is significant and correspond
           exactly with the order of columns in the TBL values.";
        amm:tblt {
          amm:key "name";
          amm:column name {
            description
              "The name of this column.
               Each name SHALL be unique within a table template.
               Each name SHALL be restricted to valid label text.";
            amm:type "//ietf/amm-base/typedef/id-text";
          }
          amm:column datatype {
            description
              "The semantic type for values in this column.";
            amm:type "./TYPEDEF/semtype";
          }
        }
      }
```

```
      amm:parameter min-elements {
        description
          "The smallest valid row count for the TBL.";
        amm:default "null";
        amm:union {
          amm:type "/ARITYPE/uvast";
          amm:type "/ARITYPE/null";
        }
      }
      amm:parameter max-elements {
        description
          "The largest valid row count for the TBL.";
        amm:default "null";
        amm:union {
          amm:type "/ARITYPE/uvast";
          amm:type "/ARITYPE/null";
        }
      }
      amm:parameter key {
        description
          "The primary key column names tuples in this table,
           space-separated labels.";
        amm:default "null";
        amm:union {
          amm:type "/ARITYPE/textstr";
          amm:type "/ARITYPE/null";
        }
      }
      amm:parameter unique {
        description
          "The set of unique column name tuples in this table, as
           space-separated labels.
           The set itself is encoded as an AC in this parameter.";
        amm:default "/AC/()";
        amm:ulist {
          amm:type "/ARITYPE/textstr";
        }
      }
    }
    amm:ident union {
      amm:enum 8;
      description
        "A list of alternative type choices.";
      amm:base "./IDENT/semtype";
      amm:parameter choices {
        description
          "The semantic type for each choice.
           The list itself is encoded as an AC in this parameter.";
```

```
      amm:ulist {
        amm:type "./TYPEDEF/semtype";
      }
    }
  }
  amm:ident seq {
    amm:enum 9;
    description
      "A sequence of similarly-typed items as a sub-sequence of an
       AC value.
       This is only usable in specific contexts, such as within a
       ./IDENT/dlist parameter.";
    amm:base "./IDENT/semtype";
    amm:parameter item-type {
      description
        "The semantic type for all items in the sequence.";
      amm:type "./TYPEDEF/semtype";
    }
    amm:parameter min-elements {
      description
        "The smallest valid item count for the sequence.";
      amm:default "null";
      amm:union {
        amm:type "/ARITYPE/uvast";
        amm:type "/ARITYPE/null";
      }
    }
    amm:parameter max-elements {
      description
        "The largest valid item count for the sequence.";
      amm:default "null";
      amm:union {
        amm:type "/ARITYPE/uvast";
        amm:type "/ARITYPE/null";
      }
    }
  }

  // Named types
  amm:typedef semtype {
    amm:enum 25;
    description
      "A parameterized reference to a semantic type object.
       This is used for introspection of AMM objects.";
    amm:type "/ARITYPE/ident" {
      amm:base "./IDENT/semtype";
    }
  }
```

```
}
<CODE ENDS>
```

Appendix C. ADM Module for AMM Semantic Types and Unions

The ADM module in this section implements base semantic types defined by the AMM. This includes the following:

* Simple types described in Section 4.2.3 of [I-D.ietf-dtn-amm].

* Container types described in Section 4.2.4 of [I-D.ietf-dtn-amm].

* Union types described in Section 4.2.5 of [I-D.ietf-dtn-amm].

```
<CODE BEGINS> file "ietf-amm-base.yang"
module ietf-amm-base {
  yang-version 1.1;
  namespace "ari://ietf/amm-base/";
  prefix amm-type;

  import ietf-amm {
    prefix amm;
  }

  organization
    "Internet Engineering Task Force (IETF)" {
    amm:enum 1;
  }
  contact
    "WG Web: <http://tools.ietf.org/wg/dtn/>
     WG List: <mailto:dtn@ietf.org>

     Editor: Brian Sipos
             <mailto:brian.sipos+ietf@gmail.com>";
  description
    "This module defines base IDENT objects and base TYPEDEF
     objects for the AMM.

     The key words 'MUST', 'MUST NOT', 'REQUIRED', 'SHALL', 'SHALL
     NOT', 'SHOULD', 'SHOULD NOT', 'RECOMMENDED', 'NOT RECOMMENDED',
     'MAY', and 'OPTIONAL' in this document are to be interpreted as
     described in BCP 14 (RFC 2119) (RFC 8174) when, and only when,
     they appear in all capitals, as shown here.

     Copyright (c) 2025 IETF Trust and the persons identified as
     authors of the code.  All rights reserved.

     Redistribution and use in source and binary forms, with or
```

```
        without modification, is permitted pursuant to, and subject to
        the license terms contained in, the Revised BSD License set
        forth in Section 4.c of the IETF Trust's Legal Provisions
        Relating to IETF Documents
        (https://trustee.ietf.org/license-info).

        This version of this YANG module is part of RFC XXXX
        (https://www.rfc-editor.org/info/rfcXXXX); see the RFC itself
        for full legal notices.";
    reference
      "draft-ietf-dtn-adm-yang";

    revision 2025-07-03 {
      description
        "Updated for latest ADM document.";
      reference
        "draft-ietf-dtn-adm-yang";
    }
    amm:enum 25;

    // abstract display hint hierarchy
    amm:ident display-hint {
      amm:enum 0;
      description
        "The base IDENT for objects usable as arguments for the
         'amm:display-hint' extension.
         This object itself provides no display hint purpose.";
      amm:abstract true;
    }
    amm:ident display-hint-integer {
      amm:enum 1;
      description
        "Intermediate base IDENT of display hints for integer values.
         Hints derived from this object SHALL only apply to built-in
         types BYTE, INT, UINT, VAST, and UVAST.";
      amm:abstract true;
      amm:base "./IDENT/display-hint";
    }
    amm:ident display-hint-float {
      amm:enum 2;
      description
        "Intermediate base IDENT of display hints for floating point
         values.
         Hints derived from this object SHALL only apply to built-in
         types REAL32 and REAL64.";
      amm:abstract true;
      amm:base "./IDENT/display-hint";
    }
```

```
amm:ident display-hint-bstr {
  amm:enum 3;
  description
    "Intermediate base IDENT of display hints for byte string
     values.
     Hints derived from this object SHALL only apply to built-in
     type BYTESTR.";
  amm:abstract true;
  amm:base "./IDENT/display-hint";
}
amm:ident display-hint-time {
  amm:enum 4;
  description
    "Intermediate base IDENT of display hints for time values.
     Hints derived from this object SHALL only apply to built-in
     types TP and TD.";
  amm:abstract true;
  amm:base "./IDENT/display-hint";
}

// Named type unions
amm:typedef type-ref {
  amm:enum 0;
  description
    "Reference to either a built-in type or a TYPEDEF object.";
  amm:union {
    amm:type "/ARITYPE/aritype";
    amm:type "/ARITYPE/typedef";
  }
}
amm:typedef integer {
  amm:enum 1;
  description
    "Any type which represents a discrete integer
     This union order prefers smaller range and signed types.";
  amm:union {
    amm:type "/ARITYPE/byte";
    amm:type "/ARITYPE/int";
    amm:type "/ARITYPE/uint";
    amm:type "/ARITYPE/vast";
    amm:type "/ARITYPE/uvast";
  }
}
amm:typedef float {
  amm:enum 2;
  description
    "Any type which represents a floating point number.";
  amm:union {
```

```
        amm:type "/ARITYPE/real32";
        amm:type "/ARITYPE/real64";
      }
    }
    amm:typedef numeric {
      amm:enum 3;
      description
        "Any type which can be used with numeric expressions.";
      amm:union {
        amm:type "./TYPEDEF/integer";
        amm:type "./TYPEDEF/float";
      }
    }
    amm:typedef primitive {
      amm:enum 4;
      description
        "Any primitive type.";
      amm:union {
        amm:type "/ARITYPE/null";
        amm:type "/ARITYPE/bool";
        amm:type "./TYPEDEF/numeric";
        amm:type "/ARITYPE/textstr";
        amm:type "/ARITYPE/bytestr";
      }
    }
    amm:typedef time {
      amm:enum 5;
      description
        "Any type which can be used with time expressions.";
      amm:union {
        amm:type "/ARITYPE/TP";
        amm:type "/ARITYPE/TD";
      }
    }
    amm:typedef simple {
      amm:enum 6;
      description
        "Any type which contains a single literal value (not nested).";
      amm:union {
        amm:type "./TYPEDEF/PRIMITIVE";
        amm:type "./TYPEDEF/TIME";
      }
    }
    amm:typedef nested {
      amm:enum 7;
      description
        "A literal type which contains other ARI values.";
      amm:union {
```

```
        amm:type "/ARITYPE/AC";
        amm:type "/ARITYPE/AM";
        amm:type "/ARITYPE/TBL";
      }
    }
    amm:typedef any {
      amm:enum 8;
      description
        "Any value representable by an ARI.";
      // These type names are built-in
      amm:union {
        amm:type "/ARITYPE/literal";
        amm:type "/ARITYPE/object";
      }
    }
    amm:typedef value-obj {
      amm:enum 9;
      description
        "A reference to an object which can produce a value.";
      amm:union {
        amm:type "/ARITYPE/const";
        amm:type "/ARITYPE/edd";
        amm:type "/ARITYPE/var";
      }
    }
    amm:typedef nonce {
      amm:enum 10;
      description
        "This type union is used to correlate Agent-Manager messages.
         The union is defined in Section 4.2.5 of draft-ietf-dtn-amm.";
      reference
        "draft-ietf-dtn-amm";
      amm:union {
        amm:type "/ARITYPE/bytestr";
        amm:type "/ARITYPE/uvast";
        amm:type "/ARITYPE/null";
      }
    }
    amm:typedef id-text {
      amm:enum 25;
      description
        "This type is used to define and match object text names.
         The type is defined in Section 4.2.5 of draft-ietf-dtn-amm.";
      reference
        "draft-ietf-dtn-amm";
      amm:type "/ARITYPE/textstr" {
        pattern '[A-Za-z_][0-9A-Za-z_\-\.]*';
      }
```

```
    }
    amm:typedef id-int {
      amm:enum 26;
      description
        "This type is used to define and match object integer
         enumerations.
         The type is defined in Section 4.2.5 of draft-ietf-dtn-amm.";
      reference
        "draft-ietf-dtn-amm";
      amm:type "/ARITYPE/int";
    }

    // operational semantic types
    amm:typedef counter32 {
      amm:enum 11;
      description
        "A 32-bit counter with an arbitrary initial value which can
         only increment and never decrement.
         When the value reaches the upper limit of the type it wraps
         around to zero.
         At least two samples of this value need to be compared over
         time to be meaningful.
         This type is consistent with the SMIv2 and YANG data types
         of the same name.";
      reference
        "RFC 2578: Structure of Management Information Version 2
         RFC 6021: Common YANG Data Types";
      amm:type "/ARITYPE/uint";
    }
    amm:typedef counter64 {
      amm:enum 12;
      description
        "A 64-bit counter with an arbitrary initial value which can
         only increment and never decrement.
         When the value reaches the upper limit of the type it wraps
         around to zero.
         At least two samples of this value need to be compared over
         time to be meaningful.
         This type is consistent with the SMIv2 and YANG data types
         of the same name.";
      reference
        "RFC 2578: Structure of Management Information Version 2
         RFC 6021: Common YANG Data Types";
      amm:type "/ARITYPE/uvast";
    }
    amm:typedef gauge32 {
      amm:enum 13;
      description
```

```
        "A 32-bit value sampling some quantized measurement.
         The value can increase or decrease arbitrarily over time.
         This type is consistent with the SMIv2 and YANG data types
         of the same name.";
      reference
        "RFC 2578: Structure of Management Information Version 2
         RFC 6021: Common YANG Data Types";
      amm:type "/ARITYPE/int";
    }
    amm:typedef gauge64 {
      amm:enum 14;
      description
        "A 64-bit value sampling some quantized measurement.
         The value can increase or decrease arbitrarily over time.
         This type is consistent with the SMIv2 and YANG data types
         of the same name.";
      reference
        "RFC 2578: Structure of Management Information Version 2
         RFC 6021: Common YANG Data Types";
      amm:type "/ARITYPE/vast";
    }
    amm:typedef timestamp {
      amm:enum 15;
      description
        "A time point representing the system clock at which a specific
         occurrence happened.
         The specific occurrence must be defined in the description
         of any node defined using this type.
         This type is consistent with the SMIv2 and YANG data types
         of the same name.";
      reference
        "RFC 2579: Textual Conventions for SMIv2
         RFC 6021: Common YANG Data Types";
      amm:type "/ARITYPE/tp";
    }

    // Restrictions on AC item types for Evaluation and EXPR
    amm:typedef eval-tgt {
      amm:enum 16;
      description
        "A value which can be the target of an evaluation.";
      amm:union {
        // special case of literal value result
        amm:type "./TYPEDEF/simple";
        // produces an eval-tgt
        amm:type "./TYPEDEF/value-obj";
        amm:type "./TYPEDEF/expr";
      }
```

```
    }
    amm:typedef expr-item {
      amm:enum 17;
      description
        "Each item of an EXPR list.";
      amm:union {
        amm:type "./TYPEDEF/simple";
        // produces an eval-tgt
        amm:type "./TYPEDEF/value-obj";
        // treated as unary operator
        amm:type "./TYPEDEF/type-ref";
        // substitutable label of an external context value
        amm:type "/ARITYPE/label";
        amm:type "/ARITYPE/oper";
      }
    }
    amm:typedef expr {
      amm:enum 18;
      description
        "The contents of an EXPR container.";
      amm:ulist {
        amm:type "./TYPEDEF/expr-item";
      }
    }

    // Restrictions on AC item types for Execution and MAC
    amm:typedef exec-tgt {
      amm:enum 19;
      description
        "A value which can be the target of an execution.";
      amm:union {
        amm:type "./TYPEDEF/exec-item";
        amm:type "./TYPEDEF/mac";
      }
    }
    amm:typedef exec-item {
      amm:enum 20;
      description
        "A reference to an object which can be executed.
         The value-object must be typed to contain an exec-tgt.";
      amm:union {
        amm:type "/ARITYPE/ctrl";
        amm:type "./TYPEDEF/value-obj"; // produces an exec-tgt
      }
    }
    amm:typedef mac {
      amm:enum 21;
      description
```

```
        "The contents of a MAC container are an ordered list of
         executable values.";
      amm:ulist {
        amm:type "./TYPEDEF/exec-item";
      }
    }

    // Restrictions on AC item types for Reporting and RPTT
    amm:typedef rpt-tgt {
      amm:enum 22;
      description
        "A value which can be the target of reporting.";
      amm:union {
        amm:type "./TYPEDEF/value-obj"; // produces an RPTT
        amm:type "./TYPEDEF/rptt";
      }
    }
    amm:typedef rptt-item {
      amm:enum 23;
      description
        "Each item of a RPTT.
         Each item references a value-producing object or contains an
         expression to be evaluated.";
      amm:union {
        // produces the report item
        amm:type "./TYPEDEF/value-obj";
        // evaluates to the report item
        amm:type "./TYPEDEF/expr";
      }
    }
    amm:typedef rptt {
      amm:enum 24;
      description
        "The contents of a report template, encoded as the sequence
         of template items.";
      amm:ulist {
        amm:type "./TYPEDEF/rptt-item";
      }
    }

    // Representation of ARI patterns as values
    amm:ident bstr-ari-pattern {
      amm:enum 5;
      description
        "Represent a binary form of ARI Pattern as human-friendly
         text form, converting identifiers if possible based on ADM
         and ODM contents.";
      reference
```

```
        "draft-ietf-dtn-adm-yang";
      amm:base "./ident/display-hint-bstr";
    }
    amm:typedef ari-pattern {
      amm:enum 27;
      description
        "The value type for an ARI pattern in binary form.";
      reference
        "draft-ietf-dtn-adm-yang";
      amm:type "/aritype/cbor" {
        amm:display-hint "./ident/bstr-ari-pattern";
      }
    }
  }
  <CODE ENDS>
```

Appendix D. ADM Module for Base Networking Objects

The ADM module in this section implements IDENT objects and semantic types associated with abstract networking identifiers and endpoints. Specific networking technologies and protocols can derive from the abstract bases to implement concrete protocol-specific logic.

```
  <CODE BEGINS> file "ietf-network-base.yang"
  module ietf-network-base {
    yang-version 1.1;
    namespace "ari://ietf/network-base/";
    prefix network-base;

    import ietf-amm {
      prefix amm;
    }
    import ietf-amm-base {
      prefix amm-base;
    }

    organization
      "Internet Engineering Task Force (IETF)" {
      amm:enum 1;
    }
    contact
      "WG Web: <http://tools.ietf.org/wg/dtn/>
       WG List: <mailto:dtn@ietf.org>

       Editor: Brian Sipos
               <mailto:brian.sipos+ietf@gmail.com>";
    description
      "This module defines base IDENT objects for network-related
```

```
      entities such as protocol endpoints. The base objects are
      general purpose, and all derived objects are layer- and/or
      protocol-specific with appropriate parameters.

      The key words 'MUST', 'MUST NOT', 'REQUIRED', 'SHALL', 'SHALL
      NOT', 'SHOULD', 'SHOULD NOT', 'RECOMMENDED', 'NOT RECOMMENDED',
      'MAY', and 'OPTIONAL' in this document are to be interpreted as
      described in BCP 14 (RFC 2119) (RFC 8174) when, and only when,
      they appear in all capitals, as shown here.

      Copyright (c) 2025 IETF Trust and the persons identified as
      authors of the code.  All rights reserved.

      Redistribution and use in source and binary forms, with or
      without modification, is permitted pursuant to, and subject to
      the license terms contained in, the Revised BSD License set
      forth in Section 4.c of the IETF Trust's Legal Provisions
      Relating to IETF Documents
      (https://trustee.ietf.org/license-info).

      This version of this YANG module is part of RFC XXXX
      (https://www.rfc-editor.org/info/rfcXXXX); see the RFC itself
      for full legal notices.";
  reference
    "draft-ietf-dtn-adm-yang";

  revision 2025-07-03 {
    description
      "Updated for latest ADM document.";
    reference
      "draft-ietf-dtn-adm-yang";
  }
  amm:enum 26;

  // General purpose network data types and forms
  amm:typedef uri {
    amm:enum 0;
    description
      "A specialization of the textstr type to only contain a single
       Uniform Resource Identifier (URI) as defined by STD 66.

       Objects using the uri type MUST be in US-ASCII encoding,
       and MUST be normalized as described by RFC 3986 Sections
       6.2.1, 6.2.2.1, and 6.2.2.2.  All unnecessary
       percent-encoding is removed, and all case-insensitive
       characters are set to lowercase except for hexadecimal
       digits, which are normalized to uppercase as described in
       Section 6.2.2.1.
```

```
          The purpose of this normalization is to help provide
          unique URIs.  Note that this normalization is not
          sufficient to provide uniqueness.  Two URIs that are
          textually distinct after this normalization may still be
          equivalent.

          Objects using the uri type may restrict the schemes that
          they permit.  For example, 'data:' and 'urn:' schemes
          might not be appropriate.

          A zero-length URI is not a valid URI.  This can be used to
          express 'URI absent' where required.

          In the value set and its semantics, this type is equivalent
          to the Uri SMIv2 textual convention defined in RFC 5017.";
        reference
          "RFC 3986: Uniform Resource Identifier (URI): Generic Syntax
           RFC 3305: Report from the Joint W3C/IETF URI Planning Interest
                     Group: Uniform Resource Identifiers (URIs), URLs,
                     and Uniform Resource Names (URNs): Clarifications
                     and Recommendations
           RFC 5017: MIB Textual Conventions for Uniform Resource
                     Identifiers (URIs)";
        amm:type "/aritype/textstr" {
          // This pattern is on the normalized scheme
          pattern '[a-z][a-z0-9+-.]*:.*';
        }
      }
      amm:ident display-bstr-uuid {
        amm:enum 17;
        description
          "Interpret byte string values as UUID and display as
           human-friendly text according to RFC 9562 conventions.";
        reference
          "RFC 9562: Universally Unique IDentifiers (UUIDs)";
        amm:base "//ietf/amm-base/IDENT/display-hint-bstr";
      }
      amm:typedef uuid {
        amm:enum 5;
        description
          "A specialization of the bytestr type to contain a binary
           encoded UUID value.
           Values of this type SHALL contain a valid UUID value
           as defined in Section 4 of RFC 9562.";
        reference
          "RFC 9562: Universally Unique IDentifiers (UUIDs)";
        amm:type "/aritype/bytestr" {
          length "16";
```

```
      }
    }
    amm:ident display-bstr-oid {
      amm:enum 18;
      description
        "Interpret byte string values as ASN.1 BER-encoded OIDs
         and display as human-friendly dotted-decimal form of
         ITU-T X.660.";
      reference
        "ITU-T X.660-2011: General procedures and top arcs of the
           international object identifier tree";
      amm:base "//ietf/amm-base/IDENT/display-hint-bstr";
    }
    amm:typedef oid {
      amm:enum 4;
      description
        "A specialization of the bytestr type to contain an ASN.1
         BER-encoded object identifier (OID) value.
         This typedef gives no more specific AMM constraints on
         the byte string value but does constrain it by these
         requirements.
         Values of this type SHALL contain valid BER-encoded OID
         contents consistent with the definiton of CBOR tag 111
         but without the actual tag, as defined in Section 2.1 of
         RFC 9090.
         Values of this type always contain an absolute OID.";
      reference
        "RFC 9090: Concise Binary Object Representation (CBOR) Tags
           for Object Identifiers
         ITU-T X.660-2011: General procedures and top arcs of the
           international object identifier tree
         ITU-T X.690-2021: ASN.1 encoding rules: Specification of
           Basic Encoding Rules (BER), Canonical Encoding
           Rules (CER) and Distinguished Encoding Rules (DER)";
      amm:type "/aritype/bytestr" {
        length "2..max";
      }
    }

    // Abstract base IDENTs and types
    amm:ident abstract-endpoint {
      amm:enum 0;
      description
        "This object defines an abstract base for all general purpose
         network/transport endpoint identifiers as seen from outside
         the endpoint node.
         Uses where the endpoint is being bound to from inside the
         node might require additional parameters outside of what is
```

```
          present in this object hierarchy.";
        amm:abstract true;
      }
      amm:typedef endpoint {
        amm:enum 1;
        description
          "A value which references a non-abstract derived object
           for specific types of network or transport endpoints.";
        amm:type "/aritype/ident" {
          amm:base "./IDENT/abstract-endpoint";
        }
      }
      amm:typedef endpoint-or-uri {
        amm:enum 3;
        description
          "A value which is either a specific parameterized endpoint
           IDENT reference or a text URI value identifying an endpoint.";
        amm:union {
          amm:type "./typedef/endpoint";
          amm:type "./typedef/uri";
        }
      }
      amm:ident abstract-endpoint-pattern {
        amm:enum 1;
        description
          "This object defines an abstract base for general purpose
           network/transport endpoint pattern matching.
           Each derived type is expected to match only a single object
           derived from <./IDENT/abstract-endpoint>.
           Each pattern SHALL specify in its descriptions which types
           of endpoints can possibly match the pattern.";
        amm:abstract true;
      }
      amm:typedef endpoint-pattern {
        amm:enum 2;
        description
          "A value which references a non-abstract derived pattern object
           for specific types of network or transport endpoints.";
        amm:type "/aritype/ident" {
          amm:base "./IDENT/abstract-endpoint-pattern";
        }
      }

      // general-purpose leaf IDENTs
      amm:ident uri-regexp-pattern {
        amm:enum 2;
        description
          "This object defines a means to match a full URI using a
```

```
          regular expression limited to I-Regexp compatibility.
          This only allows matching text values, more specifically those that
          already match the <./typedef/uri> semantic type.";
        reference
          "RFC 9485: I-Regexp: An Interoperable Regular Expression Format";
        amm:base "//ietf/network-base/ident/abstract-endpoint-pattern";
        amm:parameter regexp {
          description
            "The I-Regexp text pattern to match.
             In accordance with that definition, this pattern is implicitly
             anchored to the start and end of the text being matched
             (i.e. it is always a full match).";
          amm:type "/aritype/textstr";
        }
      }
    }
    <CODE ENDS>
```

Appendix E. IANA Assignments for display-hint

The ADM module in this section defines display-hint-derived IDENT objects described in Section 3.3.1 of [I-D.ietf-dtn-amm] for specific representations.

```
    <CODE BEGINS> file "iana-display-hints.yang"
    module iana-display-hints {
      yang-version 1.1;
      namespace "ari://iana/display-hints/";
      prefix ianadh;

      import ietf-amm {
        prefix amm;
      }
      import ietf-amm-base {
        prefix amm-base;
      }

      organization
        "Internet Assigned Numbers Authority (IANA)" {
        amm:enum 2;
      }
      contact
        "Internet Assigned Numbers Authority

         Postal: ICANN
                 12025 Waterfront Drive, Suite 300
                 Los Angeles, CA 90094-2536
                 United States of America
```

```
      Tel:    +1 310 301 5800
      <mailto:iana@iana.org>";
    description
      "This module defines leaf IDENT objects usable as 'display-hint'
       semantic type annotations derived from the base
       <ari://ietf/amm-base/ident/display-hint> object.

       The key words 'MUST', 'MUST NOT', 'REQUIRED', 'SHALL', 'SHALL
       NOT', 'SHOULD', 'SHOULD NOT', 'RECOMMENDED', 'NOT RECOMMENDED',
       'MAY', and 'OPTIONAL' in this document are to be interpreted as
       described in BCP 14 (RFC 2119) (RFC 8174) when, and only when,
       they appear in all capitals, as shown here.

       Copyright (c) 2025 IETF Trust and the persons identified as
       authors of the code.  All rights reserved.

       Redistribution and use in source and binary forms, with or
       without modification, is permitted pursuant to, and subject to
       the license terms contained in, the Revised BSD License set
       forth in Section 4.c of the IETF Trust's Legal Provisions
       Relating to IETF Documents
       (https://trustee.ietf.org/license-info).

       This version of this YANG module is part of RFC XXXX
       (https://www.rfc-editor.org/info/rfcXXXX); see the RFC itself
       for full legal notices.";
    reference
      "https://www.iana.org/assignments/DTNMA-TBA";

    revision 2025-07-03 {
      description
        "Updated for latest ADM document.";
      reference
        "draft-ietf-dtn-adm-yang";
    }
    amm:enum 0;

    // Integer number hints
    amm:ident display-int-dec {
      amm:enum 2;
      description
        "Display integers as decimal (base 10).";
      amm:base "//ietf/amm-base/ident/display-hint-integer";
    }
    amm:ident display-int-bin {
      amm:enum 3;
      description
        "Display integers as binary (base 2).";
```

```
      amm:base "//ietf/amm-base/ident/display-hint-integer";
    }
    amm:ident display-int-hex {
      amm:enum 4;
      description
        "Display integers as hexadecimal (base 16).";
      amm:base "//ietf/amm-base/ident/display-hint-integer";
    }

    // Floating-point number hints
    amm:ident display-float-dec {
      amm:enum 6;
      description
        "Display floating point values as decimal fraction.";
      amm:base "//ietf/amm-base/ident/display-hint-float";
    }
    amm:ident display-float-exp {
      amm:enum 7;
      description
        "Display floating point values as decimal exponential form.";
      amm:base "//ietf/amm-base/ident/display-hint-float";
    }
    amm:ident display-float-hex {
      amm:enum 8;
      description
        "Display floating point values as hexadecimal exponential
         form.";
      amm:base "//ietf/amm-base/ident/display-hint-float";
    }

    // Byte-string hints
    amm:ident display-bstr-text {
      amm:enum 10;
      description
        "Display byte string values as UTF-8 text where possible.
         The base16 encoding is used otherwise.";
      reference
        "draft-ietf-dtn-ari";
      amm:base "//ietf/amm-base/ident/display-hint-bstr";
    }
    amm:ident display-bstr-base16 {
      amm:enum 11;
      description
        "Display byte string values as base16-encoded.";
      reference
        "draft-ietf-dtn-ari";
      amm:base "//ietf/amm-base/ident/display-hint-bstr";
    }
```

```
    amm:ident display-bstr-base64 {
      amm:enum 12;
      description
        "Display byte string values as base64url-encoded.";
      reference
        "draft-ietf-dtn-ari";
      amm:base "//ietf/amm-base/ident/display-hint-bstr";
    }

    // TIME type (TP or TD built-ins) hints
    amm:ident display-time-text {
      amm:enum 14;
      description
        "Display TP and TD values as text in accordance with
         RFC 3339 not using separator characters.";
      reference
        "draft-ietf-dtn-ari";
      amm:base "//ietf/amm-base/ident/display-hint-time";
    }
    amm:ident display-time-dec {
      amm:enum 15;
      description
        "Display TP and TD values as decimal fraction with units of
         seconds, which may or may not include a fractional part.";
      reference
        "draft-ietf-dtn-ari";
      amm:base "//ietf/amm-base/ident/display-hint-time";
    }
  }
  <CODE ENDS>
```

Appendix F. ADM Module for DTNMA Agents

The ADM module in this section implements the entities described in Section 4.3 of [I-D.ietf-dtn-amm]. This includes the following:

* Introspective access to the ADMs supported by the Agent.

* Introspective access to the VARs in ODMs.

* A feature and introspective access to rules in ODMs.

* A report template used to announce presence of an Agent.

* Base helper controls and arithmetic operators

```
<CODE BEGINS> file "ietf-dtnma-agent.yang"
module ietf-dtnma-agent {
  yang-version 1.1;
  namespace "ari://ietf/dtnma-agent/";
  prefix da;

  import ietf-amm {
    prefix amm;
  }
  import ietf-amm-semtype {
    prefix amm-semtype;
  }
  import ietf-amm-base {
    prefix amm-base;
  }
  import ietf-network-base {
    prefix network-base;
  }

  organization
    "Internet Engineering Task Force (IETF)" {
    amm:enum 1;
  }
  contact
    "WG Web: <http://tools.ietf.org/wg/dtn/>
     WG List: <mailto:dtn@ietf.org>

     Editor: Brian Sipos
             <mailto:brian.sipos+ietf@gmail.com>";
  description
    "This module implements the DTN Management Architecture (DTNMA)
     Agent core functionality.

     The key words 'MUST', 'MUST NOT', 'REQUIRED', 'SHALL', 'SHALL
     NOT', 'SHOULD', 'SHOULD NOT', 'RECOMMENDED', 'NOT RECOMMENDED',
     'MAY', and 'OPTIONAL' in this document are to be interpreted as
     described in BCP 14 (RFC 2119) (RFC 8174) when, and only when,
     they appear in all capitals, as shown here.

     Copyright (c) 2025 IETF Trust and the persons identified as
     authors of the code.  All rights reserved.

     Redistribution and use in source and binary forms, with or
     without modification, is permitted pursuant to, and subject to
     the license terms contained in, the Revised BSD License set
     forth in Section 4.c of the IETF Trust's Legal Provisions
     Relating to IETF Documents
     (https://trustee.ietf.org/license-info).
```

```
       This version of this YANG module is part of RFC XXXX
       (https://www.rfc-editor.org/info/rfcXXXX); see the RFC itself
       for full legal notices.";
    reference
      "draft-ietf-dtn-adm-yang";

    revision 2025-07-03 {
      description
        "Updated for latest ADM document.";
      reference
        "draft-ietf-dtn-adm-yang";
    }

    feature rules {
      description
        "Conforming to this feature enables time-based and
         state-based autonomy rules.";
    }

    grouping obj-list-params {
      description
        "Common parameters for object listing";
      amm:parameter include-adm {
        description
          "If true, listings will include objects from ADMs";
        amm:default "false";
        amm:type "/aritype/BOOL";
      }
    }

    grouping ensure-object-id {
      description
        "Common parameters for providing object reference to
         ensure an object exists in an ODM.

         When this grouping is used, the logic operates according
         to the following in order to succeed.
         The parent 'namespace' SHALL refer to an existing ODM.
         Both the text 'obj-name' and integer 'obj-enum' SHALL
         either both not exist, in which case the object will be
         created, or both are allocated to the same object, in
         which case the object will be modified as needed.";
      amm:parameter namespace {
        description
          "The parent ODM namespace.";
        amm:type "/aritype/namespace";
      }
      amm:parameter obj-name {
```

```
        description
          "Text name for the desired object.";
        amm:type "//ietf/amm-base/typedef/id-text";
      }
      amm:parameter obj-enum {
        description
          "Integer enumeration for the desired object.";
        amm:type "//ietf/amm-base/typedef/id-int";
      }
    }

    grouping any-binary-operands {
      description
        "Any-value bianry operands";
      amm:operand left {
        description
          "The left-side operand.";
        amm:type "//ietf/amm-base/typedef/any";
      }
      amm:operand right {
        description
          "The left-side operand.";
        amm:type "//ietf/amm-base/typedef/any";
      }
    }

    grouping any-unary-result {
      description
        "Any-value result";
      amm:result result {
        description
          "The single value.";
        amm:type "//ietf/amm-base/typedef/any";
      }
    }

    grouping numeric-unary-operands {
      description
        "Numeric unary operand";
      amm:operand val {
        description
          "The single value.";
        amm:type "//ietf/amm-base/typedef/NUMERIC";
      }
    }

    grouping numeric-binary-operands {
      description
```

```
        "Numeric bianry operands";
      amm:operand left {
        description
          "The left-side operand.";
        amm:type "//ietf/amm-base/typedef/NUMERIC";
      }
      amm:operand right {
        description
          "The left-side operand.";
        amm:type "//ietf/amm-base/typedef/NUMERIC";
      }
    }

    grouping numeric-unary-result {
      description
        "Numeric result";
      amm:result result {
        description
          "The single value.";
        amm:type "//ietf/amm-base/typedef/NUMERIC";
      }
    }

    grouping integer-unary-operands {
      description
        "Integer unary operand";
      amm:operand val {
        description
          "The single value.";
        amm:type "//ietf/amm-base/typedef/INTEGER";
      }
    }

    grouping integer-binary-operands {
      description
        "Integer bianry operands";
      amm:operand left {
        description
          "The left-side operand.";
        amm:type "//ietf/amm-base/typedef/INTEGER";
      }
      amm:operand right {
        description
          "The left-side operand.";
        amm:type "//ietf/amm-base/typedef/INTEGER";
      }
    }
```

```
    grouping integer-unary-result {
      description
        "Integer result";
      amm:result result {
        description
          "The single value.";
        amm:type "//ietf/amm-base/typedef/INTEGER";
      }
    }

    grouping bool-unary-operands {
      description
        "Boolean unary operand";
      amm:operand val {
        description
          "The single value.";
        amm:type "/aritype/BOOL";
      }
    }

    grouping bool-binary-operands {
      description
        "Boolean bianry operands";
      amm:operand left {
        description
          "The left-side operand.";
        amm:type "/aritype/BOOL";
      }
      amm:operand right {
        description
          "The left-side operand.";
        amm:type "/aritype/BOOL";
      }
    }

    grouping bool-unary-result {
      description
        "Boolean result";
      amm:result result {
        description
          "The single value.";
        amm:type "/aritype/BOOL";
      }
    }

    amm:enum 1;
    amm:edd sw-vendor {
      amm:enum 0;
```

```
      description
        "The vendor for this Agent implementation.";
      amm:type "/aritype/TEXTSTR";
    }
    amm:edd sw-version {
      amm:enum 1;
      description
        "The version for this Agent implementation.";
      amm:type "/aritype/TEXTSTR";
    }
    amm:edd capability {
      amm:enum 2;
      description
        "A table to indicate the ADM capability of the sending agent.";
      amm:tblt {
        amm:key "org-name adm-name";
        amm:column org-name {
          description
            "The organization name containing the ADM";
          amm:type "/aritype/TEXTSTR";
        }
        amm:column org-enum {
          description
            "The organization enumeration containing the ADM";
          amm:type "/aritype/VAST";
        }
        amm:column model-name {
          description
            "The model name of the ADM";
          amm:type "/aritype/TEXTSTR";
        }
        amm:column model-enum {
          description
            "The model enumeration of the ADM";
          amm:type "/aritype/VAST";
        }
        amm:column revision {
          description
            "The specific revision the agent supports.";
          amm:type "/aritype/TEXTSTR";
        }
        amm:column features {
          description
            "The features of the ADM which the agent supports.";
          amm:ulist {
            amm:type "/aritype/TEXTSTR";
          }
        }
```

```
      }
    }
    amm:const hello {
      amm:enum 0;
      description
        "A report template to indicate the presence of an agent
         on a network.";
      amm:init-value "/AC/(./EDD/sw-vendor,./EDD/sw-version,"
                   + "./EDD/capability)";
      amm:type "//ietf/amm-base/typedef/RPTT";
    }

    // Messaging counters
    amm:edd num-msg-rx {
      amm:enum 3;
      amm:type "//ietf/amm-base/typedef/counter64";
    }
    amm:edd num-msg-rx-failed {
      amm:enum 4;
      amm:type "//ietf/amm-base/typedef/counter64";
    }
    amm:edd num-msg-tx {
      amm:enum 5;
      amm:type "//ietf/amm-base/typedef/counter64";
    }
    amm:edd num-msg-tx-failed {
      amm:enum 15;
      amm:type "//ietf/amm-base/typedef/counter64";
    }
    amm:edd last-msg-rx-time {
      amm:enum 17;
      amm:type "/aritype/TP";
    }

    // Execution counters and states
    amm:edd num-exec-started {
      amm:enum 6;
      amm:type "//ietf/amm-base/typedef/counter64";
    }
    amm:edd num-exec-succeeded {
      amm:enum 7;
      amm:type "//ietf/amm-base/typedef/counter64";
    }
    amm:edd num-exec-failed {
      amm:enum 8;
      amm:type "//ietf/amm-base/typedef/counter64";
    }
    amm:edd exec-running {
```

```
      amm:enum 9;
      amm:tblt {
        amm:key "pid";
        amm:column pid {
          amm:type "/aritype/UVAST";
        }
        amm:column target {
          amm:type "//ietf/amm-base/typedef/any";
        }
        amm:column state {
          amm:type "/aritype/BYTE" {
            amm:int-labels {
              enum waiting {
                value 0;
                description
                  "The execution is waiting on a condition or timer
                   to continue.";
              }
              enum running {
                value 1;
                description
                  "The execution is currently running.";
              }
            }
          }
        }
      }
    }

    // MAC helper controls
    amm:ctrl if-then-else {
      amm:enum 0;
      description
        "Evaluate an expression and follow one of two branches of
         further evaluation.";
      amm:parameter condition {
        description
          "The condition to evaluate.";
        amm:type "//ietf/amm-base/typedef/eval-tgt";
      }
      amm:parameter on-truthy {
        description
          "The object to execute when the condition is truthy.";
        amm:default "null";
        amm:union {
          amm:type "//ietf/amm-base/typedef/exec-tgt";
          amm:type "/aritype/NULL";
        }
```

```
      }
      amm:parameter on-falsy {
        description
          "An optional execution when the condition is falsey.";
        amm:default "null";
        amm:union {
          amm:type "//ietf/amm-base/typedef/exec-tgt";
          amm:type "/aritype/NULL";
        }
      }
      amm:result branch {
        description
          "Indication of which branch was executed.";
        amm:type "/aritype/BOOL";
      }
    }
    amm:ctrl catch {
      amm:enum 1;
      description
        "Attempt to execute a target, and if there is some failure
         catch it and execute an alternative target.";
      amm:parameter try {
        description
          "The object to execute.";
        amm:type "//ietf/amm-base/typedef/exec-tgt";
      }
      amm:parameter on-failure {
        description
          "An optional execution after failure.";
        amm:default "null";
        amm:union {
          amm:type "//ietf/amm-base/typedef/exec-tgt";
          amm:type "/aritype/NULL";
        }
      }
      amm:result try-success {
        description
          "True if the try target succeeded without exception.";
        amm:type "/aritype/BOOL";
      }
    }
    amm:ctrl wait-for {
      amm:enum 2;
      description
        "This control causes the execution to pause for a given
         amount of time.
         This is intended to be used within a macro to separate
         controls in time.";
```

```
      amm:parameter duration {
        amm:type "/aritype/TD";
      }
    }
    amm:ctrl wait-until {
      amm:enum 3;
      description
        "This control causes the execution to pause until a specific
         absolute time point.
         This is intended to be used within a macro to separate
         controls in time or as a first macro item to delay execution
         after the time of reception.";
      amm:parameter time {
        amm:type "/aritype/TP";
      }
    }
    amm:ctrl wait-cond {
      amm:enum 4;
      description
        "This control causes the execution to pause until a condition
         expression evaluates to truthy.
         This is intended to be used within a macro to separate
         controls in time or as a first macro item to delay execution
         until the condition is met.";
      amm:parameter condition {
        amm:type "//ietf/amm-base/typedef/eval-tgt";
      }
    }

    // Value production and reporting
    amm:ctrl inspect {
      amm:enum 5;
      description
        "Produce a result value to inspect the agent state.
         This does not perform any EXPR evaluation or RPTT handling.";
      amm:parameter ref {
        description
          "An object to produce a value from.";
        amm:type "//ietf/amm-base/typedef/VALUE-OBJ";
      }
      amm:result val {
        description
          "The produced value.";
        amm:type "//ietf/amm-base/typedef/any";
      }
    }
    amm:ctrl report-on {
      amm:enum 6;
```

```
      description
        "Agent-wide control to generate a report from a report
         template (RPTT) value.
         The parameter is a single RPTT list that would be produced
         by an object.
         If used for more than one-shot diagnostics, defining a RPTT
         (e.g. in a CONST within an ODM) is more efficient because
         the RPTT value would not need be present in the EXECSET or
         corresponding RPTSET.";
      amm:parameter template {
        description
          "The reporting template, either as a RPTT value directly or
           as a reference to a value-producing object (possibly
           parameterized) which itself produces an RPTT value.";
        amm:type "//ietf/amm-base/typedef/rpt-tgt";
      }
      amm:parameter destinations {
        description
          "Destinations for the resulting report.
           If given an empty list (the default) reports will be sent
           to the manager executing this CTRL.
           When executed as a rule action, the destinations list is
           mandatory.

           It is possible that each destination has different access
           limitations so each destination might see different report
           contents.";
        amm:default "/ac/()";
        amm:ulist {
          amm:type "//ietf/network-base/typedef/endpoint-or-uri";
        }
      }
    }

    // Objects related to ODM handling
    amm:edd odm-list {
      amm:enum 16;
      description
        "A table of ODM within the agent.";
      amm:tblt {
        amm:key "org-name odm-name";
        amm:column org-name {
          description
            "The organization name containing the ODM";
          amm:type "//ietf/amm-base/typedef/id-text";
        }
        amm:column org-enum {
          description
```

```
            "The organization enumeration containing the ODM";
          amm:type "//ietf/amm-base/typedef/id-int";
        }
        amm:column model-name {
          description
            "The model name of the ODM";
          amm:type "//ietf/amm-base/typedef/id-text";
        }
        amm:column model-enum {
          description
            "The model enumeration of the ODM";
          amm:type "//ietf/amm-base/typedef/id-int";
        }
        amm:column revision {
          description
            "The specific revision the agent supports.";
          amm:type "/aritype/TEXTSTR";
        }
      }
    }
    amm:ctrl ensure-odm {
      amm:enum 18;
      description
        "Ensure a specific ODM is present.
         The referenced organization and model identifiers SHALL either
         not already exist or exist with the exact combination given by
         these parameters";
      amm:parameter org-name {
        description
          "The organization name containing the ODM";
        amm:type "//ietf/amm-base/typedef/id-text";
      }
      amm:parameter org-id {
        description
          "The organization enumeration ID containing the ODM";
        amm:type "//ietf/amm-base/typedef/id-int";
      }
      amm:parameter model-name {
        description
          "The model name containing the ODM.";
        amm:type "//ietf/amm-base/typedef/id-text" {
          pattern '!.+';
        }
      }
      amm:parameter model-id {
        description
          "The model enumeration ID containing the ODM";
        amm:type "//ietf/amm-base/typedef/id-int" {
```

```
          range "min..-1";
        }
      }
    }
    amm:ctrl obsolete-odm {
      amm:enum 19;
      description
        "Mark a specific ODM as obsolete if it is present.";
      amm:parameter odm-ns {
        description
          "Namespace of the ODM";
        amm:type "/aritype/namespace";
      }
    }

    // Helpers for VAR
    amm:ctrl var-reset {
      amm:enum 7;
      description
        "Modify a VAR state to its default value.";
      amm:parameter target {
        description
          "The VAR object to affect.";
        amm:type "/aritype/VAR";
      }
    }
    amm:ctrl var-store {
      amm:enum 8;
      description
        "Modify a VAR state to a specific value.";
      amm:parameter target {
        description
          "The VAR object to affect.";
        amm:type "/aritype/VAR";
      }
      amm:parameter value {
        description
          "The exact value to store in the VAR.";
        amm:type "//ietf/amm-base/typedef/any";
      }
    }

    // Objects related to TYPEDEF handling
    amm:edd typedef-list {
      amm:enum 10;
      description
        "A table of TYPEDEF within the agent.";
      uses obj-list-params;
```

```
      amm:tblt {
        amm:key "obj";
        amm:column obj {
          amm:type "/aritype/typedef";
        }
      }
    }

    // Objects related to CONST handling
    amm:edd const-list {
      amm:enum 14;
      description
        "A table of CONST within the agent.";
      uses obj-list-params;
      amm:tblt {
        amm:key "obj";
        amm:column obj {
          amm:type "/aritype/CONST";
        }
        amm:column type {
          amm:type "//ietf/amm-semtype/typedef/semtype";
        }
      }
    }
    amm:ctrl ensure-const {
      amm:enum 11;
      description
        "Ensure a specific CONST is present in an ODM.
         If an object already exists with the same identifiers, type,
         and value this control will do nothing and succeed.
         If an object already exists in the same namespace with either
         of the object identifiers this control will fail.";
      uses ensure-object-id;
      amm:parameter type {
        description
          "The type for the CONST object.";
        amm:type "//ietf/amm-semtype/typedef/semtype";
      }
      amm:parameter value {
        description
          "A required constant value.
           This value SHALL match the type given for this object.";
        amm:type "//ietf/amm-base/typedef/any";
      }
    }
    amm:ctrl obsolete-const {
      amm:enum 12;
      description
```

```
        "Mark a specific CONST as obsolete if it is present in an
         ODM.";
      amm:parameter obj {
        description
          "A reference to a CONST within an ODM only.";
        amm:type "/aritype/CONST";
      }
    }

    // Objects related to VAR handling
    amm:edd var-list {
      amm:enum 11;
      description
        "A table of VAR within the agent.";
      uses obj-list-params;
      amm:tblt {
        amm:key "obj";
        amm:column obj {
          amm:type "/aritype/VAR";
        }
        amm:column type {
          amm:type "//ietf/amm-semtype/typedef/semtype";
        }
      }
    }
    amm:ctrl ensure-var {
      amm:enum 9;
      description
        "Ensure a specific VAR is present in an ODM.
         If an object already exists with the same identifiers, type,
         and init-value this control will do nothing and succeed.
         If an object already exists in the same namespace with either
         of the object identifiers (but not both), or the type or
         is different, this control will fail.
         That means that this control can change only the initial value
         of a pre-existing VAR object.";
      uses ensure-object-id;
      amm:parameter type {
        description
          "The type for the VAR object.";
        amm:type "//ietf/amm-semtype/typedef/semtype";
      }
      amm:parameter init-value {
        description
          "The exact value to use as the initial value of
           the new object and to store as the initial state
           of the object if created.
           This value SHALL match the type given for this object.";
```

```
        amm:type "//ietf/amm-base/typedef/any";
      }
    }
    amm:ctrl obsolete-var {
      amm:enum 10;
      description
        "Mark a specific VAR as obsolete if it is present in an ODM.";
      amm:parameter obj {
        description
          "A reference to a VAR within an ODM only.";
        amm:type "/aritype/VAR";
      }
    }

    // Objects related to SBR handling
    amm:edd sbr-list {
      if-feature "rules";
      amm:enum 12;
      description
        "A table of SBR within the agent.";
      uses obj-list-params;
      amm:tblt {
        amm:key "obj";
        amm:column obj {
          amm:type "/aritype/SBR";
        }
        amm:column action {
          description
            "The execution when this rule triggers.";
          amm:type "//ietf/amm-base/typedef/MAC";
        }
        amm:column condition {
          amm:type "//ietf/amm-base/typedef/EXPR";
        }
        amm:column min-interval {
          amm:type "/aritype/TD";
        }
        amm:column max-count {
          amm:type "/aritype/UVAST";
        }
        amm:column init-enabled {
          amm:type "/ARITYPE/BOOL";
        }
        amm:column enabled {
          amm:type "/ARITYPE/BOOL";
        }
      }
    }
```

```
    amm:edd tbr-list {
      if-feature "rules";
      amm:enum 13;
      description
        "A table of TBR within the agent.";
      uses obj-list-params;
      amm:tblt {
        amm:key "obj";
        amm:column obj {
          amm:type "/aritype/TBR";
        }
        amm:column action {
          description
            "The execution when this rule triggers.";
          amm:type "//ietf/amm-base/typedef/MAC";
        }
        amm:column start-time {
          amm:type "//ietf/amm-base/typedef/TIME";
        }
        amm:column period {
          amm:type "/aritype/TD";
        }
        amm:column max-count {
          amm:type "/aritype/UVAST";
        }
        amm:column init-enabled {
          amm:type "/ARITYPE/BOOL";
        }
        amm:column enabled {
          amm:type "/ARITYPE/BOOL";
        }
      }
    }

    // CTRLs related to rules
    amm:ctrl ensure-sbr {
      if-feature "rules";
      amm:enum 13;
      description
        "Ensure that a state-based rule with the given identifiers
         exists within an ODM.";
      uses ensure-object-id;
      amm:parameter action {
        description
          "The execution when this rule triggers.";
        amm:type "//ietf/amm-base/typedef/MAC";
      }
      amm:parameter condition {
```

```
        amm:type "//ietf/amm-base/typedef/EXPR";
      }
      amm:parameter min-interval {
        amm:type "/ARITYPE/TD";
      }
      amm:parameter max-count {
        amm:type "/ARITYPE/UVAST";
      }
      amm:parameter init-enabled {
        amm:type "/ARITYPE/BOOL";
      }
      amm:result res {
        amm:type "/ARITYPE/UINT";
      }
    }
    amm:ctrl ensure-tbr {
      if-feature "rules";
      amm:enum 14;
      description
        "Ensure that a time-based rule with the given identifiers
         exists within an ODM.";
      uses ensure-object-id;
      amm:parameter action {
        description
          "The execution when this rule triggers.";
        amm:type "//ietf/amm-base/typedef/MAC";
      }
      amm:parameter start-time {
        amm:type "//ietf/amm-base/typedef/TIME";
      }
      amm:parameter period {
        amm:type "/ARITYPE/TD";
      }
      amm:parameter max-count {
        amm:type "/ARITYPE/UVAST";
      }
      amm:parameter init-enabled {
        amm:type "/ARITYPE/BOOL";
      }
      amm:result res {
        amm:type "/ARITYPE/UINT";
      }
    }
    amm:ctrl ensure-rule-enabled {
      if-feature "rules";
      amm:enum 15;
      description
        "Ensure rule is enabled or disabled.";
```

```
      amm:parameter obj-id {
        description
          "The rule to ensure enabled setting for.";
        amm:union {
          amm:type "/ARITYPE/SBR";
          amm:type "/ARITYPE/TBR";
        }
      }
      amm:parameter enabled {
        amm:type "/ARITYPE/BOOL";
      }
      amm:result res {
        amm:type "/ARITYPE/uint";
      }
    }
    amm:ctrl reset-rule-enabled {
      if-feature "rules";
      amm:enum 16;
      description
        "Reset rule enabled to initial setting.";
      amm:parameter obj-id {
        description
          "The rule to reset enabled seetting for.";
        amm:union {
          amm:type "/ARITYPE/SBR";
          amm:type "/ARITYPE/TBR";
        }
      }
      amm:result res {
        amm:type "/ARITYPE/UINT";
      }
    }
    amm:ctrl obsolete-rule {
      if-feature "rules";
      amm:enum 17;
      description
        "Mark a specific SBR or TBR as obsolete if it is present in
         an ODM.";
      amm:parameter obj-id {
        description
          "A reference to a rule within an ODM only.";
        amm:union {
          amm:type "/ARITYPE/SBR";
          amm:type "/ARITYPE/TBR";
        }
      }
      amm:result success {
        amm:type "/ARITYPE/BOOL";
```

```
    }
  }

  // Numeric operators
  amm:oper negate {
    amm:enum 0;
    description
      "Negate a value.
       This is equivalent to multiplying by -1 but a shorter
       expression.";
    uses numeric-unary-operands;
    uses numeric-unary-result;
  }
  amm:oper add {
    amm:enum 1;
    description
      "Add two numeric values.
       The operands are cast to the least compatible numeric type
       before the arithmetic.";
    uses any-binary-operands;
    uses any-unary-result;
  }
  amm:oper sub {
    amm:enum 2;
    uses any-binary-operands;
    uses any-unary-result;
  }
  amm:oper multiply {
    amm:enum 3;
    uses any-binary-operands;
    uses any-unary-result;
  }
  amm:oper divide {
    amm:enum 4;
    uses any-binary-operands;
    uses any-unary-result;
  }
  amm:oper remainder {
    amm:enum 5;
    uses numeric-binary-operands;
    uses numeric-unary-result;
  }

  // Bitwise operators
  amm:oper bit-not {
    amm:enum 6;
    uses integer-unary-operands;
    uses integer-unary-result;
```

```
    }
    amm:oper bit-and {
      amm:enum 7;
      uses integer-binary-operands;
      uses integer-unary-result;
    }
    amm:oper bit-or {
      amm:enum 8;
      uses integer-binary-operands;
      uses integer-unary-result;
    }
    amm:oper bit-xor {
      amm:enum 9;
      uses integer-binary-operands;
      uses integer-unary-result;
    }

    // Boolean operators
    amm:oper bool-not {
      amm:enum 10;
      uses bool-unary-operands;
      uses bool-unary-result;
    }
    amm:oper bool-and {
      amm:enum 11;
      uses bool-binary-operands;
      uses bool-unary-result;
    }
    amm:oper bool-or {
      amm:enum 12;
      uses bool-binary-operands;
      uses bool-unary-result;
    }
    amm:oper bool-xor {
      amm:enum 13;
      uses bool-binary-operands;
      uses bool-unary-result;
    }

    // Value comparison
    amm:oper compare-eq {
      amm:enum 14;
      uses any-binary-operands;
      uses bool-unary-result;
    }
    amm:oper compare-ne {
      amm:enum 15;
      uses any-binary-operands;
```

```
      uses bool-unary-result;
    }

    // Numeric comparison
    amm:oper compare-gt {
      amm:enum 16;
      description
        "Compare two numbers by value.
         The result is true if the left value is greater than the
         right value.
         The operands are cast to the least compatible numeric type
         before the comparison.";
      uses numeric-binary-operands;
      uses bool-unary-result;
    }
    amm:oper compare-ge {
      amm:enum 17;
      description
        "Compare two numbers by value.
         The result is true if the left value is greater than or equal
         to the right.
         The operands are cast to the least compatible numeric type
         before the comparison.";
      uses numeric-binary-operands;
      uses bool-unary-result;
    }
    amm:oper compare-lt {
      amm:enum 18;
      description
        "Compare two operands by value.
         The result is true if the left value is less than the right.
         The operands are cast to the least compatible numeric type
         before the comparison.";
      uses numeric-binary-operands;
      uses bool-unary-result;
    }
    amm:oper compare-le {
      amm:enum 19;
      description
        "Compare two operands by value.
         The result is true if the left value is less than or
         equal to the right.
         The operands are cast to the least compatible numeric type
         before the comparison.";
      uses numeric-binary-operands;
      uses bool-unary-result;
    }
```

```
// Table filtering
amm:oper tbl-filter {
  amm:enum 20;
  description
    "Filter a table first by rows and then by columns.";
  amm:parameter row-match {
    description
      "A filter applied using two phases for each row:
         1. Substitute row values for LABEL items within EXPR
         2. Evaluate the expression and consider the row matched
            if the result is a truthy value
       Each LABEL item refers to the value at a column index
       within the context of the input row. Label substitution
       is performed as described in the value production section
       of the DTNMA AMM document.";
    amm:type "//ietf/amm-base/typedef/EXPR";
  }
  amm:parameter columns {
    description
      "A filter to restrict output to only the listed columns.
       The order of columns in the filtered table shall
       correspond with the order of columns in this list.";
    amm:ulist {
      amm:type "/aritype/UVAST";
    }
  }
  amm:operand in {
    description
      "Table to filter.";
    amm:type "/aritype/TBL";
  }
  amm:result out {
    description
      "The filtered table.";
    amm:type "/aritype/TBL";
  }
}

// List filtering
amm:oper list-get {
  amm:enum 21;
  description
    "Retrieve an item from the given list.";
  amm:parameter index {
    description
      "Index of the value to retrieve from the list.";
    amm:type "//ietf/amm-base/typedef/INTEGER";
  }
```

```
        amm:operand in {
          description
            "List to retrieve a value from.";
          amm:type "/aritype/AC";
        }
        amm:result out {
          description
            "The retrieved value or undefined if the index is not valid.";
          amm:type "//ietf/amm-base/typedef/any";
        }
      }

      // Map filtering
      amm:oper map-get {
        amm:enum 22;
        description
          "Retrieve an item from the given map.";
        amm:parameter key {
          description
            "Key of the value to retrieve from the map.";
          amm:type "//ietf/amm-base/typedef/primitive";
        }
        amm:operand in {
          description
            "Map to retrieve a value from.";
          amm:type "/aritype/AM";
        }
        amm:result out {
          description
            "The retrieved value or undefined if the key is not valid.";
          amm:type "//ietf/amm-base/typedef/any";
        }
      }
    }
    <CODE ENDS>
```

Appendix G. ADM Module for DTNMA Agent Access Control Lists

The ADM module in this section implements the entities described in Appendix A of [I-D.ietf-dtn-amm].

```
    <CODE BEGINS> file "ietf-dtnma-agent-acl.yang"
    module ietf-dtnma-agent-acl {
      yang-version 1.1;
      namespace "ari://ietf/dtnma-agent-acl/";
      prefix acl;

      import ietf-amm {
```

```
      prefix amm;
    }
    import ietf-dtnma-agent {
      prefix da;
    }
    import ietf-network-base {
      prefix network-base;
    }

    organization
      "Internet Engineering Task Force (IETF)" {
      amm:enum 1;
    }
    contact
      "WG Web: <http://tools.ietf.org/wg/dtn/>
       WG List: <mailto:dtn@ietf.org>

       Editor: Brian Sipos
               <mailto:brian.sipos+ietf@gmail.com>";
    description
      "This module implements the DTN Management Architecture (DTNMA)
       Agent Access Control List (ACL) functionality.

       The key words 'MUST', 'MUST NOT', 'REQUIRED', 'SHALL', 'SHALL
       NOT', 'SHOULD', 'SHOULD NOT', 'RECOMMENDED', 'NOT RECOMMENDED',
       'MAY', and 'OPTIONAL' in this document are to be interpreted as
       described in BCP 14 (RFC 2119) (RFC 8174) when, and only when,
       they appear in all capitals, as shown here.

       Copyright (c) 2025 IETF Trust and the persons identified as
       authors of the code.  All rights reserved.

       Redistribution and use in source and binary forms, with or
       without modification, is permitted pursuant to, and subject to
       the license terms contained in, the Revised BSD License set
       forth in Section 4.c of the IETF Trust's Legal Provisions
       Relating to IETF Documents
       (https://trustee.ietf.org/license-info).

       This version of this YANG module is part of RFC XXXX
       (https://www.rfc-editor.org/info/rfcXXXX); see the RFC itself
       for full legal notices.";
    reference
      "draft-ietf-dtn-adm-yang";

    revision 2025-07-03 {
      description
        "Updated for latest ADM document.";
```

```
    reference
      "draft-ietf-dtn-adm-yang";
  }
  amm:enum 2;

  // Common types in this model
  amm:typedef entry-id {
    amm:enum 2;
    description
      "The identifier for an entry in any access control table.";
    amm:type "/ARITYPE/UINT";
  }
  amm:typedef optional-entry-id {
    amm:enum 3;
    description
      "An optional entry identifier (or a null value).";
    amm:union {
      amm:type "./TYPEDEF/entry-id";
      amm:type "/ARITYPE/null";
    }
  }

  // Atomic permission objects
  amm:ident permission {
    amm:enum 0;
    description
      "The base IDENT for objects usable as permissions in the
       access control table.
       Derived objects MUST define the specific conditions in which
       the permission is exercised.
       Derived objects MAY be parameterized as needed to control
       specific access granted by each permission.
       This object itself is abstract and provides not specific
       permission.";
    amm:abstract true;
  }
  amm:typedef permission-list {
    amm:enum 0;
    description
      "A list of permission object (IDENT) references.";
    amm:ulist {
      amm:type "/aritype/ident" {
        amm:base "./ident/permission";
      }
    }
  }

  // built-in permissions
```

```
    amm:ident execute {
      amm:enum 10;
      description
        "Permission to allow executing specific CTRL objects.
         This includes objects used directly as EXECSET targets,
         indirectly within macros, or as actions of TBR or SBR
         objects.";
      amm:base "./ident/permission";
    }
    amm:ident produce {
      amm:enum 11;
      description
        "Permission to allow producing values from CONST, VAR, or
         EDD objects.
         This includes production for use by reporting or evaluating
         procedures of the Agent.";
      amm:base "./ident/permission";
    }
    amm:ident modify-var {
      amm:enum 12;
      description
        "Permission to allow modifying state on VAR objects.";
      amm:base "./ident/permission";
    }
    amm:ident create-odm {
      amm:enum 13;
      description
        "Permission to allow creating a new ODM within an Agent.";
      amm:base "./ident/permission";
    }
    amm:ident delete-odm {
      amm:enum 14;
      description
        "Permission to allow deleting an ODM within an Agent.";
      amm:base "./ident/permission";
    }
    amm:ident create-object {
      amm:enum 15;
      description
        "Permission to allow creating a new object within an ODM.
         The permission itself does not restrict the type of object,
         but the access list can limit the item to specific object
         types and names.";
      amm:base "./ident/permission";
    }
    amm:ident delete-object {
      amm:enum 16;
      description
```

```
        "Permission to allow deleting an object within an ODM.
         The permission itself does not restrict the type of object,
         but the access list can limit the item to specific object
         types and names.";
      amm:base "./ident/permission";
    }

    // Default access permission
    amm:var default-access {
      amm:enum 1;
      description
        "The default permission for any object if no specific access
         are present in the <./EDD/access-list> table.";
      amm:type "./TYPEDEF/permission-list";
    }

    // Access control table and modifier controls
    amm:edd access-list {
      amm:enum 2;
      description
        "List the current group--object access.";
      amm:tblt {
        amm:key "access-id";
        amm:column access-id {
          description
            "A unique identifier for this access list item.";
          amm:type "./TYPEDEF/entry-id";
        }
        amm:column group-ids {
          description
            "A cross-reference to the groups getting access, or the
             empty list to match all groups.";
          amm:ulist {
            amm:type "./TYPEDEF/entry-id";
          }
        }
        amm:column objects {
          description
            "A pattern for objects being granted access, which may
             match all objects.";
          amm:type "//ietf/amm-base/typedef/ari-pattern";
        }
        amm:column permissions {
          description
            "Refernces to specific permissions being granted.
             Individual permission references MAY be parameterized
             as necessary.
             The absence of permissions does not grant any access.";
```

```
          amm:type "./TYPEDEF/permission-list";
        }
        amm:column added-at {
          description
            "Agent timestamp when this group was added.";
          amm:type "//ietf/amm-base/typedef/timestamp";
        }
        amm:column updated-at {
          description
            "Agent timestamp when this group was added or updated.";
          amm:type "//ietf/amm-base/typedef/timestamp";
        }
      }
    }
    amm:ctrl ensure-access {
      amm:enum 3;
      description
        "Ensures that an access control permission is set for the
         given group and ARI pattern.";
      amm:parameter access-id {
        description
          "A unique identifier for this access list item.";
        amm:type "./TYPEDEF/entry-id";
      }
      amm:parameter group-ids {
        description
          "Identify the groups being assigned the permission,
           or the empty list for all-groups.";
        amm:ulist {
          amm:type "./TYPEDEF/entry-id";
        }
      }
      amm:parameter objects {
        description
          "ARI pattern to determine which objects will be assigned
           the permissions.";
        amm:type "//ietf/amm-base/typedef/ari-pattern";
      }
      amm:parameter permissions {
        description
          "Permissions to grant to the associated group and objects";
        amm:type "./TYPEDEF/permission-list";
      }
    }
    amm:ctrl discard-access {
      amm:enum 4;
      description
        "Discard any access item with the given ID.";
```

```
      amm:parameter access-id {
        description
          "Uniquely identify the access";
        amm:type "./TYPEDEF/entry-id";
      }
    }

    //
    // Group management
    //
    amm:edd current-groups {
      amm:enum 3;
      description
        "Get the group IDs for the current execution context, which may
         incude the implicit Agent group ID zero.";
      amm:ulist {
        amm:type "./TYPEDEF/entry-id";
      }
    }
    amm:edd group-list {
      amm:enum 4;
      description
        "Get the defined access control groups.";
      amm:tblt {
        amm:key "group-id";
        amm:column group-id {
          description
            "The unique identifier for a group.
             The agent itself has implicit group ID zero which cannot
             be modified.";
          amm:type "./TYPEDEF/entry-id";
        }
        amm:column name {
          description
            "The human-friendly name of the group.";
          amm:type "//ietf/amm-base/typedef/id-text";
        }
        amm:column members {
          description
            "The list of members of the group, identified by matching
             transport endpoint.";
          amm:ulist {
            amm:type "//ietf/network-base/TYPEDEF/endpoint-pattern";
          }
        }
        amm:column added-at {
          description
            "Agent timestamp when this group was added.";
```

```
          amm:type "//ietf/amm-base/typedef/timestamp";
        }
        amm:column updated-at {
          description
            "Agent timestamp when this group was added or updated.";
          amm:type "//ietf/amm-base/typedef/timestamp";
        }
      }
    }
    amm:ctrl ensure-group {
      amm:enum 5;
      description
        "Ensure that after the control completes a group exists with
         the given information";
      amm:parameter group-id {
        description
          "Uniquely identify the group";
        amm:type "./TYPEDEF/entry-id";
      }
      amm:parameter name {
        description
          "Name of the group";
        amm:type "//ietf/amm-base/typedef/id-text";
      }
    }
    amm:ctrl ensure-group-members {
      amm:enum 7;
      description
        "Ensure that the membership of a group has a specific set of
         endpoint patterns.";
      amm:parameter group-id {
        description
          "Uniquely identify the group";
        amm:type "./TYPEDEF/entry-id";
      }
      amm:parameter members {
        amm:ulist {
          amm:type "//ietf/network-base/TYPEDEF/endpoint-pattern";
        }
      }
    }
    amm:ctrl discard-group {
      amm:enum 6;
      description
        "Discard any group with the given ID.";
      amm:parameter group-id {
        description
          "Uniquely identify the group";
```

```
        amm:type "./TYPEDEF/entry-id";
      }
    }
  }
  <CODE ENDS>
```

Implementation Status

This section is to be removed before publishing as an RFC.

[NOTE to the RFC Editor: please remove this section before publication, as well as the reference to [RFC7942], [github-dtnma-ace], and [github-dtnma-adms].]

This section records the status of known implementations of the protocol defined by this specification at the time of posting of this Internet-Draft, and is based on a proposal described in [RFC7942]. The description of implementations in this section is intended to assist the IETF in its decision processes in progressing drafts to RFCs. Please note that the listing of any individual implementation here does not imply endorsement by the IETF. Furthermore, no effort has been spent to verify the information presented here that was supplied by IETF contributors. This is not intended as, and must not be construed to be, a catalog of available implementations or their features. Readers are advised to note that other implementations can exist.

A full implementation in Python language of the ADM module encoding and decoding, and for translating between ARI text name and integer enumeration (based on ADM contents) is present in the apl-fy24 development branch of [github-dtnma-ace]. This repository includes unit test vectors for verifying ADM module encoding and decoding and is used as the basis of ADM syntax checking and formatting tools in the ADM modules in this document.

The ADM modules present in Appendix A through D are version controlled in their original form in the apl-fy24 development branch of [github-dtnma-adms].

Acknowledgments

The following participants contributed technical material, use cases, and useful thoughts on the overall approach captured in this document: David Linko, Sarah Heiner, and Jenny Cao of the Johns Hopkins University Applied Physics Laboratory.

Authors' Addresses

Edward J. Birrane, III
The Johns Hopkins University Applied Physics Laboratory
11100 Johns Hopkins Rd.
Laurel, MD 20723
United States of America
Phone: +1 443 778 7423
Email: Edward.Birrane@jhuapl.edu

Brian Sipos
The Johns Hopkins University Applied Physics Laboratory
Email: brian.sipos+ietf@gmail.com

Justin Ethier
The Johns Hopkins University Applied Physics Laboratory
Email: Justin.Ethier@jhuapl.edu