

# Application Management Model (AMM) and Application Data Model (ADM) Updates

**IETF 117 DTN WG**

Brian Sipos  
JHU/APL

# Background

- Earlier drafts of “Asynchronous Management” for AMP, AMM, and ADM have existed for several years
- Implementation experience has been obtained for agents, managers, and related tools (both operational and diagnostic)
- Issues and enhancements with existing protocols and definitions have accumulated over time
- Goal is for the DTN Management Architecture (DTNMA) suite of protocols and tools to be as straightforward to use as legacy management systems (SNMP and \*CONF)

# Major Issues Being Addressed

- Typing improvements and consistency
  - Separate ARI-based type-value handling from all others in AMP
  - Introduce the use of undefined non-value for error modes
  - Strengthen rules for implicit type conversions
- Establish a semantic typing mechanism for creating type unions and adding metadata to types (e.g. units, range)
  - Allows a named taxonomy of derived types to be defined, which simplifies object definitions and interpretations
- Syntax and processing model for: MAC lists, EXPR lists, RPTT lists
- Separate managed object model from data encodings
  - Object model exists in ADM and ODM
  - All value encodings are now ARI-based
- Separated autonomy capabilities from core agent capability
- Improve ADM and object lifecycle annotations
- Simplified and minified messaging

# Earlier AMP Type Code Points

- Left table is “Primitive Data Types,” right is “AMM Objects”
- ARI literals use ‘offset into the primitive type table’ as separate code point

Basic Data Type	Mnemonic	Enumeration	Numeric	Structure	Mnemonic	Enumeration	Numeric
Boolean	BOOL	16	No	Constant	CONST	0	No
BYTE	BYTE	17	No	Control	CTRL	1	No
Character String	STR	18	No	Externally Defined Data	EDD	2	No
Signed 32-bit Integer	INT	19	Yes	Literal	LIT	3	No
Unsigned 32-bit Integer	UINT	20	Yes	Macro	MAC	4	No
Signed 64-bit Integer	VAST	21	Yes	Operator	OPER	5	No
Unsigned 64-bit Integer	UFAST	22	Yes	Report	RPT	6	No
Single-Precision Floating Point	REAL32	23	Yes	Report Template	RPTT	7	No
Double-Precision Floating Point	REAL64	24	Yes	State-Based Rule	SBR	8	No
Reserved		25-31	No	Table	TBL	9	No
				Table Template	TBLT	10	No
				Time-Based Rule	TBR	11	No
				Variable	VAR	12	No
				Reserved		13-15	No

We propose to add literals for: BYTESTR, Time Values (TP, TD), LABEL text, and opaque CBOR item

MAC, RPT(T), and TBL(T) object types aren't actually part of ADM/ODM, these are now semantic types specializing AC

# Type Hierarchy and Taxonomy

- Hierarchy of ARI types (related to ARI structure):
  - Literal: the value is within an ARI text
    - Simple: the atomic data being managed by an application
    - Complex: things like time or CBOR item
    - Container: ARI collection (AC)
    - Undefined value (discussion in slide)
  - Object Reference, with orthogonal attributes for:
    - Namespace, ADM or ODM
    - Object type
    - Object name
- New typedef objects allow creating semantic types, which exist only in the model and not in an ARI
  - e.g. define NUMERIC as union of  
BYTE / INT / UINT / VAST / UVAST / REAL32 / REAL64
  - Simplifies type signatures for objects (EDD, CTRL, VAR, OPER)
- New ADM “type” syntax makes it easier to create anonymous type unions for object definitions
  - None of these ADM-defined types are reflected in the ARI model, no encoding

# Literal Types vs. Semantic Types

- Literal types affect ARI processing
  - Literal types define syntax and how ARI text-form translates to and from binary-form
  - Literal types define the vocabulary and domain of represent-able data values (e.g. a 32-bit signed integer vs. a 32-bit float)
  - These types are used in agent ADM implementations and APIs (e.g. “get the third parameter as a UVAST”)
  - New literal types correspond to changes in ARI processing entities; agents, managers, tools, and configurations
    - Both agent and manager *must* be updated to handle a new type
- Semantic types affect logical handling
  - Semantic types can combine literal types by a type union
  - Semantic types can restrict literal types by specific enumerated/bit values, numeric range, text pattern
  - These types are used in value-type and parameter signatures and for presentation (e.g. “this is a counter of bytes”)
  - Semantic types can be defined in ADMs and ODMs but don’t have any direct effect on agent processing
    - An agent or manager *could* validate outside of an ADM but will still function even if they do not

# New Types and Purposes

- The earlier Agent processing model had limited fail-safe behavior
  - Problems generating a report would inhibit the entire report
- A new on-the-wire value “undefined” uses existing CBOR syntax
  - This allows an Agent to handle cases where reports or tables fail to fully generate, or to signal things like the failure of a control to fully execute
- The undefined value is not present in the primitive data types
  - No definition of a literal value can specify that the undefined value is expected
  - All processors of literal values from Agents must handle an undefined value
  - Literal ARI is handled as a special case, always as an untyped-literal
- A new NULL primitive type and “null” singleton value allow an ADM to express the notions of optional value and identity object
  - This also uses existing CBOR syntax, so no overloaded definitions are used
  - An identity CONST object is similar in concept to MIB and YANG use
- This doesn't impose on existing ADMs
  - It is a design decision of whether or not to allow a parameter, state, or result value to be null

# Time Types and Handling

- Earlier time-related types had issues:
  - The TV definition used one syntax to encode two separate semantics: absolute and relative time
  - Agent implementations were inconsistent in how the TV and TS were handled
  - Time types were limited to one-second precision by definition
- New type model has two distinct types with distinct and cohesive semantics:
  - Time Point (TP) is an instant in absolute time using the DTN Time Epoch
  - Time Difference (TD) is a pure relative time
- New typedef mechanism allows signatures to take advantage of type unions
  - It is simple to use the TIME type, which is a union of TP and TD, where necessary
- Both time types are encoded as a signed number with new syntax to allow sub-second precision
  - Uses the same “decimal float” syntax as base CBOR specification
  - CDDL example on the right:
  - Encoding logic will choose the smallest representation
- Both time types provide friendly text representation based on RFC 3339 / ISO 8601
- Base ADM will include a mechanism to expose Agent’s supported time precision

```
time-val = int / time-fraction
time-fraction = [
    exp: (-9 .. 9) .within int,
    mantissa: int,
]
```

# Clarify Processing of Execution and MAC

- Execution Context:
  - The current ADM document does not give specific requirements on the execution context for EDD, CTRL, or MAC
  - Without a specific guarantee of information available to those contexts,
  - Context includes: source peer name/address (for implied report messages)
- Execution Ordering:
  - Allow message reception to be handled in any order or in parallel
  - Allow report population to be in any order or parallel
  - Allow Execute CTRL/MAC message to be handled in any order or parallel
  - Allow CTRL Result items to be sent in any combination, but recommend batching (below)
  - Refine EXPR processing model for stages and item-processing ordering
  - Require MAC to have in-order execution
- Batching
  - Recommend results/reports/tables generated from the same context to be batched together into a single message
  - New report message structure optimizes items with similar creation time

# Clarify Processing of Expressions

- Earlier AMP/ADM spec defined an expression (EXPR) as a combination of result type and postfix-processed (RPN) AC list
- Expressions are used only as: VAR initializer, SBR condition, and RPTT item
- In all of these cases the result type is either known or unneeded
  - VAR has an intrinsic storage type and SBR requires BOOL type
  - RPTT item type doesn't affect report processing
- Expression is now defined as a typedef of AC with a restricted domain of allowed ARI contents
  - As “AC[\* (PRIMVAL | VALUE-OBJ | OPER)]” with VALUE-OBJ as “CONST | EDD | VAR”
- Clarified processing stages, logically equivalent to two-pass:
  1. Convert all non-OPER object references into literals
  2. Process the AC list from the end as RPN

# Clarify Processing of Report Templates

- Earlier AMP/ADM spec defined a report template (RPTT) and tabular report template (TBLT) as separate AMM object types
- Report Template is now defined as a typedef of AC with a restricted domain of allowed ARI contents
- Report (RPT) is now defined as a typedef of an AC with structure
- Table Template is now handled as a complex type on an EDD
  - Tables are generated from external data just like all other EDD uses
- Tabular data is now defined as a typedef of an AC with structure
- All of these changes allow optimized behavior of an Agent ADM:
  - Report caching mechanism

# Clarify Value Production

- Behavior common to CONST, VAR, and EDD references
  - Together these have a semantic type union VALUE-REF
- Used by execution, evaluation, and reporting procedures in similar ways for similar processing
- Some nuances about parameter handling, which is allowed for CONST and VAR for specific circumstances
  - Parameters are used for parameter-by-label flow down to produced MAC, EXPR, and RPTT values

# Separated Autonomy from Core Behavior

- Some potential DTNMA users are dissuaded by needing to include autonomy features (delayed CTRL, TBR, SBR)
  - This was raised in a NASA review as a major barrier to adoption
- The new core spec does not contain any time-based or state-based processing
  - A core AMP agent can be implemented without any timekeeping to provide just controls and commanded (or agent-triggered) reporting
  - Allows a bare-bones DTNMA Agent to be very simple and deterministic
- An autonomy “rule” feature includes objects to manage TBR and SBR and to perform one-shot delayed execution
  - Uses the feature aspect of ADMs to allow implementation choices of complex behavior

# Simplified Messaging (outside AMM/ADM)

- Similar to ARI Updates, this update defines a CDDL model for AMP messaging
- Keeping message and message group structure
  - Execute Message
  - Report Message
- Combine all agent-to-manager content to a single message type
  - Called “Report” message for now
  - Use the ARI to distinguish the type of item

```
...
$amp-msg /= msg-rpt
msg-rpt = [
  msg-type: 2,
  timestamp: time-val,
  items = [*msg-rpt-item]
]
msg-rpt-item = $msg-rpt-item .within [
  ari-objref,
  created-td: time-val,
  *any
]
$msg-rpt-item /= report
report = [
  rptt: objref-type<RPTT>,
  created: amp-td,
  *lit-ari
]
$msg-rpt-item /= table
table = [
  tblt: objref-type<TBLT>,
  created: amp-td,
  col-count: uint,
  *lit-ari
]
...
```

# Provide an ADM Profile of YANG

- ADM structure is embedded as a profile of YANG 1.1
- Inherits some YANG behavior:
  - module, submodule, and revision
  - prefix and import
  - feature and if-feature
  - status, reference, and description
  - Name resolution and file layout
- Prohibits other YANG behavior:
  - action, notification, and rpc
  - Isolated data nodes
  - Configuration vs state, datastores
  - Cross-imports between ADM and legacy modules
- Extends for new behavior:
  - One keyword per AMM object type
  - Object enumeration
  - Formal parameters
  - CONST value and VAR initializer

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

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

  organization "Example Org.";
  description "Example module.";
  amm:enum "4294967296";

  ...
  amm:const hello {
    uses amm:RPTT;
    amm:value "(../EDD/amp_version,../EDD/capability)";
    description
      "A report template to indicate the presence
      of an agent on a network.";
  }
  ...
}
```

# Multiple ADM Revisions

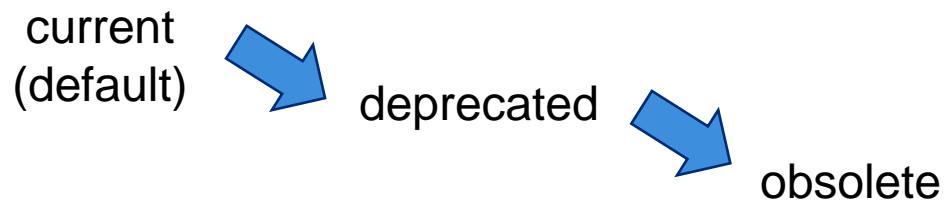
- Use the term “revision” instead of “version” to disambiguate with the ADM or protocol version
- Similar to SNMP and NETCONF (and other \*CONF), a single Agent must be limited to a single revision of each ADM
  - Because of this no ADM revision number appears within an ARI
- A new Agent ADM table will expose ADMs usable on an Agent
  - This is expected to only be used for discovery and troubleshooting
- For an ADM file store, the canonical filename of an ADM will be its name joined to its revision with the “@” symbol
- When an ADM is used by another ADM, it is for a specific revision
  - This allows ADMs to change over time without affecting dependencies
  - ADM management tools can provide hints when multiple ADM revisions are needed transitively, to help avoid “dependency explosion” of multiple revisions

# ADM Conformance and Features

- Based on preexisting YANG behavior, the ADM has a new concept of a “feature” and an Agent implementation can implement (or not) individual module features
- Features allow an Agent implementation to tailor an ADM within limited bounds
- Features, similar to types, affect the interpretation of the ADM but not the behavior of individual objects
  - Objects are either present or absent to conform to a feature
- Constraints on feature consistency are similar to for status
  - Objects within a feature must not be depended upon outside of that feature
  - e.g. a report template outside of the feature cannot reference an object within the feature; the report must also be within the feature
- The conformance of an agent to individual features of an ADM is exposed in the Agent’s table of supported ADMs

# Lifecycle and Annotations

- An ADM needs to reflect the full lifecycle of an object model:
  1. Definition of new objects, preserving enumerations of existing objects
  2. Deprecating objects, along with requirements on transitivity
  3. Obsoleting objects to be no longer supported
- Existing implied enumeration based on ordinal within an ADM object group
  - Only draft-status ADMs should have non-enumerated objects
  - When one object in an ADM group receives an enum value all objects must be given values for stability
- The SMI and YANG models already provide mechanisms to do this, reuse the same concepts and syntax
  - Object “status” passes through the states below
  - Dependencies must not reference objects in a ‘lower’ state



# Open Issues

- What belongs in the Agent ADM?
  - Autonomy rules as a feature
  - Report templates and tables related to objects available to the Agent
  - Numeric operators
- Finalize syntax for in-ADM typing
  - Existing YANG typing follows same two-level structure as XML Schema
  - AMM semantic types should be one-level for simplicity, no difference between 'leaf' value typing and structured value typing
- Refine terminology for reporting: report vs. item vs. entry
- Add requirements for protocol messaging necessary to support ADM operation
  - E.g. execution correlators from managers

# Related Work

- Concurrent and supporting changes in ARI syntax and semantics
  - All of AMP, ADM, and ARI use the same meta-model (and code points)

# Next Steps

- Reconcile document contents with DTNMA and ARI documents
- Get feedback on AMM and ADM changes
- Translate existing ADMs into profiled YANG modules
  - It's the same logical object model so can be automated to some extent
- Trial implementations: agent, manager, and/or tools
  - Stand-alone agent and manager, extracted from NASA ION, is pending open-source release from APL

# **BPSec COSE Context**

**IETF 117 DTN WG**

Brian Sipos  
JHU/APL

# Background

- BPSec and its Default Security Context are usable but intentionally limited in scope:
  - A limited number of symmetric-keyed encryption and MAC algorithms
  - Defines a variable additional authenticated data (AAD) binding to the block/bundle
  - No explicit key identifiers are available
- For internet-facing nodes, possibly as subnetwork gateways, there is a need for PKI-integrated security
  - This was indicated by IETF SECDIR review of BPSec draft and also discussed as a near-future need by NASA DTN planning group
- Don't want to reinvent the wheel, and CBOR Object Signing and Encryption (COSE) already provides syntax and semantics for current and future PKI security
  - Even COSE (with a restricted profile as used here) still provides a lot of variability, in the same sense that TLS or S/MIME does, which must be managed out-of-band (e.g. don't use ECC algorithms if security acceptors can't support it)

# Current State of the Draft

- Earlier edits added an AAD Scope mechanism to allow binding to multiple, arbitrary blocks of the same bundle
  - This expands upon the AAD Scope mechanism of RFC 9173
  - Blocks are identified by explicit Block Number, not relative “the target block”
- Document is pending WG last call at the time of writing
  - Feedback is TBD
- Additional review requested from the COSE mailing list
  - General acceptance, no major issues raised
  - Resulted in minor editorial changes

# Next Steps

- This is not intended to replace or supersede existing BPSec interoperability contexts in RFC 9173
- The point of this security context is to allow BPSec in a PKIX environment in the very near term
- This document doesn't address what kinds of policy are required in a BPA/BPSEC implementation
  - COSE and PKI present a more complex set of possible valid and invalid configurations
  - Security Considerations are given to guide implementations, but are not prescriptive or universal
  - *e.g.* can a policy be written to allow a node to trust *all* signatures by other nodes with certificates under a specific CA?