

ASYNCHRONOUS MESSAGE SERVICE (AMS)

Concept Paper

August 2004

FOREWORD

As the computers used to conduct space flight mission operations both in flight and on the ground increase in capability, the software operating on those computers tends to increase in functional scope and thus take on greater operational significance. With that increase in scope comes increasing size and complexity, which may be partially mitigated by decomposition into modules whose functionality can be readily defined and tested. However, this modularity in turn entails a growing reliance on effective communication among modules. That is, mere decomposition cannot diminish the functional complexity implied by complex requirements. It can only partition that complexity into manageable portions, while the resulting web of communication relationships among modules introduces new complexity of a different order: rather than a relatively simple system of a few increasingly large and complex modules, a modern mission requires an increasingly large and complex system of relatively simple modules.

Increasing complexity tends to increase the likelihood of failure. The increasing complexity of mission systems based on communication among modules therefore tends to increase the chance of such systems failing even as the success of those systems become increasingly critical to the achievement on mission objectives. Measures that can minimize the chance of failure in complex systems – exhaustive regression testing and configuration management, flight rules constraining the exercise of unproven system capabilities and the introduction of improvements – increase cost if they are taken and increase risk if they are not.

These considerations have led to the present recommendation for a standard system of communication – *messaging* – among mission software modules. The objective of this proposed messaging standard is to reduce mission cost and risk by confining much of the complexity of modern mission systems to relatively static and proven reusable infrastructure.

Introduction

1.1 Purpose and Scope

This document defines a CCSDS Asynchronous Message Service (AMS) for mission data system communications. The service and its protocol implement an architectural concept under which the modules of mission systems may be designed as if they were to operate in isolation, each one producing and consuming mission information without explicit awareness of which other modules are currently operating. Communication relationships among such modules are self-configuring; this tends to minimize complexity in the development and operations of modular data systems.

A system built on this model is a “society” of generally autonomous modules interacting in real time that may fluctuate freely over time in response to changing mission objectives, modules’ functional upgrades, and recovery from individual module failure. The purpose of AMS, then, is to reduce mission cost and risk by providing standard, reusable infrastructure for the exchange of information among data system modules in a manner that is simple to use, highly automated, flexible, robust, scalable, and efficient.

This Recommendation specifies the protocol procedures and data units that accomplish automatic configuration of AMS communication relationships, dynamic reconfiguration of those relationships during operations, and the use of those relationships to accomplish the exchange of mission information among data system modules.

1.2 Applicability

This Recommendation specifies protocols and associated services that enable communication among modules of mission data systems, specifically:

- between modules of a ground data system,
- between modules of a flight data system,
- between modules of different ground data systems,
- between modules of the flight data systems of different spacecraft,
- between modules of flight data systems and modules of ground data systems, over interplanetary distances.

1.3 Conventions and Definitions

1.3.1 Bit Numbering Convention and Nomenclature

In this document, the following convention is used to identify each bit in an N -bit field. The first bit in the field to be transmitted (i.e., the most left justified when drawing a

figure) is defined to be ‘Bit 0’; the following bit is defined to be ‘Bit 1’ and so on up to ‘Bit $N-1$ ’. When the field is used to express a binary value (such as a counter), the Most Significant Bit (MSB) shall be the first transmitted bit of the field, i.e., ‘Bit 0’.

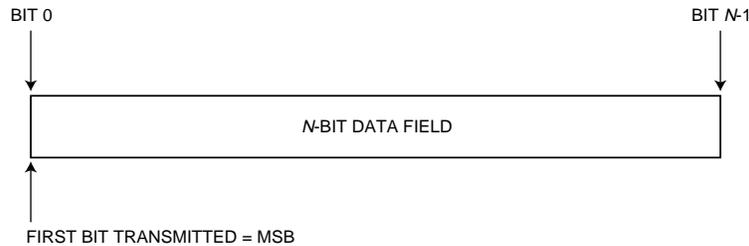


Figure 1: Bit Numbering Convention

In accordance with modern data communications practice, spacecraft data fields are often grouped into 8-bit ‘words’ which conform to the above convention. Throughout this Recommendation, the following nomenclature is used to describe this grouping:

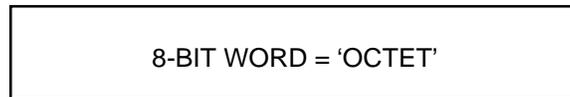


Figure 2: Octet Convention

By CCSDS convention, all ‘spare’ bits shall be permanently set to value ‘zero’.

1.3.2 Organization of the Recommendation

This Recommendation is organized as follows:

- Section 2 provides an overview of AMS, its intended use, and a description of the main interactions involved in message exchange.
- Section 3 defines the services provided by AMS along with the associated primitives and parameters.
- Section 4 defines the AMS protocol procedures.
- Section 5 defines the AMS protocol data units.
- Section 6 defines the procedures and PDUs for the auxiliary Remote AMS (RAMS) protocol.
- Section 7 describes the AMS Management Information Base (MIB).
- Annex A. provides a list of informative references.
- Annex B provides a list of acronyms and definitions.

- Annex C discusses the currently recognized underlying transport services for AMS.

1.3.3 Definitions

1.3.3.1 Definitions from OSI Basic Reference Model

This Recommendation makes use of a number of terms defined in reference [5]. The use of those terms in this Recommendation shall be understood in a generic sense, i.e., in the sense that those terms are generally applicable to any of a variety of technologies that provide for the exchange of information between real systems. Those terms are:

- entity;
- Protocol Data Unit (PDU);
- service;
- Service Access Point (SAP);
- Service Data Unit (SDU).

1.3.3.2 Definitions from Open Systems Interconnection (OSI) Service Definition Conventions

This Recommendation makes use of a number of terms defined in reference [6]. The use of those terms in this Recommendation shall be understood in a generic sense, i.e., in the sense that those terms are generally applicable to any of a variety of technologies that provide for the exchange of information between real systems. Those terms are:

- Indication;
- Primitive;
- Request;
- Response.

1.3.3.3 Terms Defined in This Recommendation

Within the context of this document the following definitions apply:

A *continuum* is a closed set of entities that utilize AMS for purposes of communication among themselves.

An *application* is a data system implementation, typically taking the form of a set of source code text files, that uses AMS procedures to accomplish its purposes. Each application is identified by an *application name*.

An *instance* of an application is a functioning projection of the application onto a set of one or more running computers.

An *authority* is a unit of mission organization that is responsible for the configuration and operation of some instance of an application. Each authority is identified by an *authority name*.

A *message* is an octet array of known size which, when copied from the memory of one module of an application instance to that of another (*exchanged*), conveys information that can further the purposes of those application instances.

The *subject number* (or *subject*) of a message is an integer embedded in the message that indicates the general nature of the information the message conveys.

The *content* of a message is the array of zero or more octets embedded in the message containing the specific information that the message conveys.

A *node* – a mission data system module – is a communicating entity that implements some part of the functionality of some AMS application instance by, among other activities, exchanging messages with other nodes.

A *message space* is the set of all of the nodes of one AMS application instance that are members of a single AMS continuum. Each message space is uniquely identified within that continuum by the combination of the name of the application and the name of the authority that is responsible for the application instance.

A *zone* is an administrative subset of a message space, declared during application instance configuration as specified by the message space's controlling authority. Each zone is uniquely identified within the message space by *zone name* and therefore is uniquely identified within its continuum by the combination of message space identifier (application name and authority name) and zone name.

A *subject name* is a text string that serves as a symbolic representation of some subject number within some message space.

1.4 References

The following documents contain provisions which, through reference in this text, constitute provisions of this Recommendation. At the time of publication, the editions indicated were valid. All documents are subject to revision, and users of this Recommendation are encouraged to investigate the possibility of applying the most recent editions of the documents indicated below. The CCSDS Secretariat maintains a register of currently valid CCSDS Recommendations.

- [1] *Advanced Orbiting Systems, Networks and Data Links: Architectural Specification*. Recommendation for Space Data System Standards, CCSDS 701.0-B-3. Blue Book. Issue 3. Washington, D.C.: CCSDS, June 2001.
- [2] *Packet Telemetry*. Recommendation for Space Data System Standards, CCSDS 102.0-B-5. Blue Book. Issue 5. Washington, D.C.: CCSDS, November 2000.

- [3] *Packet Telemetry Service Specification*. Recommendation for Space Data System Standards, CCSDS 103.0-B-2. Blue Book. Issue 2. Washington, D.C.: CCSDS, June 2001.
- [4] *Telecommand Part 2—Data Routing Service*. Recommendation for Space Data System Standards, CCSDS 202.0-B-3. Blue Book. Issue 3. Washington, D.C.: CCSDS, June 2001.
- [5] *Information Technology—Open Systems Interconnection—Basic Reference Model: The Basic Model*. International Standard, ISO/IEC 7498-1:1994. Geneva: ISO, 1994.
- [6] *Information Technology—Open Systems Interconnection—Basic Reference Model—Conventions for the Definition of OSI Services*. International Standard, ISO/IEC 10731:1994. Geneva: ISO, 1994.

The latest issues of CCSDS documents may be obtained from the CCSDS Secretariat at the address indicated on page i.

2 Overview

2.1 General

2.1.1 Architectural character

A data system based on AMS has the following characteristics:

- Any module may be introduced into the system at any time. That is, the order in which system modules commence operation is immaterial; a module never needs to establish an explicit communication “connection” or “channel” to any other module in order to pass messages to it or receive messages from it.
- Any module may be removed from the system at any time without inhibiting the ability of any other module to continue sending and receiving messages. That is, the termination of any module, whether planned or unplanned, only causes the termination of other modules that have been specifically designed to terminate in this event.
- When a module must be upgraded to an improved version, it may be terminated and its replacement may be started at any time; there is no need to interrupt operations of the system as a whole.
- When the system as a whole must terminate, the order in which the system’s modules cease operation is immaterial.

AMS-based systems are highly robust, lacking any innate single point of failure and tolerant of unplanned module termination. At the same time, communication within an AMS-based system can be rapid and efficient:

- Messages are exchanged directly between modules (nodes) rather than through any central message dispatching nexus.
- Messages are automatically conveyed using the “best” (typically – though not necessarily – the fastest) underlying transport service to which the sending and receiving modules both have access. For example, messages between two ground system modules running in different computers on a common LAN would likely be conveyed via TCP/IP, while messages between modules running on two flight processors connected to a common bus memory board might be conveyed via a shared-memory message queue.

Finally, AMS is designed to be highly scalable: partitioning message spaces into zones enables an application instance to comprise hundreds or thousands of cooperating modules without significant impact on application performance.

2.1.2 Message exchange models

Most message exchange in an AMS-based data system is conducted on a “publish-subscribe” model:

- A node uses AMS procedures to announce that it is “subscribing” to messages of a specified subject.
- From that time on (until the subscription is canceled), whenever any node in the message space uses AMS procedures to “publish” a message of that subject, a copy of the message is automatically delivered to that subscribing node and to all others that have announced similar subscriptions.

This model can greatly simplify application development and integration. In effect, each node plugs itself into a data “grid”, much as producers and consumers of electric power – for example, a hydroelectric plant and a kitchen toaster – plug into an electric power grid. An AMS node can insert into such a data grid whatever data it produces, without having to know much about the consumer(s) of that data, and draw from the grid whatever data it requires without having to know much about the producer(s). The design of a node is largely decoupled from the designs of all other nodes in the same way that the design of a toaster is largely decoupled from the design of a power plant.

For some purposes, though, it may still be necessary for a node to send a message privately and explicitly to some specific node, e.g., in *reply* to a published message. AMS procedures support this communication model as well when it is required.

All AMS message exchange is *asynchronous*. That is, each message is sent in a “postal” rather than “telephonic” manner: upon sending a message, an AMS node need not wait

for arrival of any message (such as a reply to the message it sent) before continuing performance of its functions.

Although all message exchange among nodes is asynchronous, for some purposes it may be desirable to apply the information in a reply message (received asynchronously) to the context in which the antecedent message was published. To this end, AMS enables a node to include a *context number* in any original (non-reply) message; AMS procedures can be used to reply to any original message, whether published or sent privately, and the reply to a message automatically includes an echo of the context number (if any) embedded in the original message. This number can be used to retrieve some block of contextual information, enabling the node to link the information in a reply message to the application activity which caused the antecedent message to be issued, so that this activity may be continued in a pseudo-synchronous fashion. The specific mechanism used to establish this linkage is an implementation matter.

2.2 Architectural elements

2.2.1 General

The architectural elements involved in the asynchronous message service protocol are depicted in Figure 3 and described below.

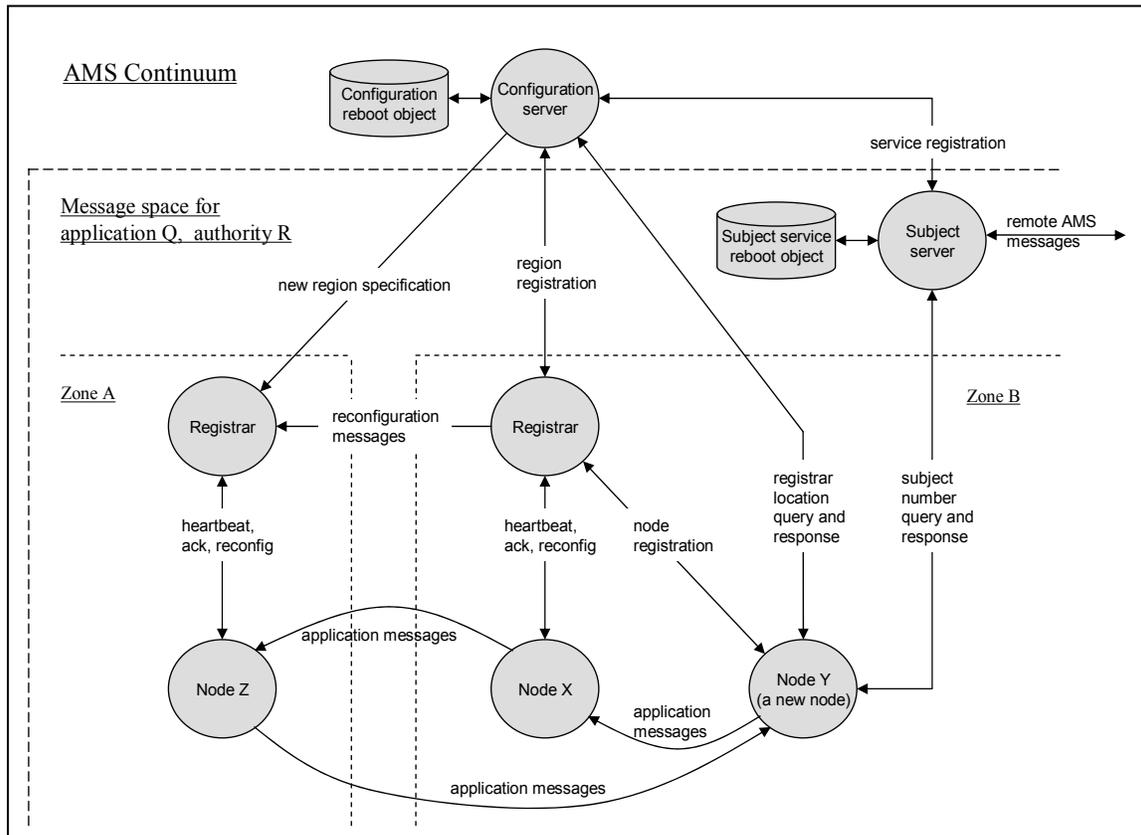


Figure 3: Architectural Elements of AMS

2.2.2 Communicating entities

All AMS communication is conducted among four types of communicating entities: nodes (defined earlier), registrars, subject servers, and configuration servers.

A *registrar* is a communicating entity that catalogues information regarding the nodes that populate a single zone of a message space. It responds to queries for this information, and it updates this information as changes are announced.

A *subject server* is a communicating entity that manages a database of subject names and corresponding subject numbers for a single message space. It responds to queries for this information, and it updates this information as changes are announced.

A *configuration server* is a communicating entity that manages a database of configuration information for a single continuum. In particular, it catalogues information regarding the message spaces that the continuum comprises, notably the locations of all subject servers and all registrars. It responds to queries for this information, and it updates this information as changes are announced.

2.3 Overview of interactions

2.3.1 Transport services for application messages

AMS occupies a position in the OSI protocol stack model somewhere between level 4 (Transport) and level 7 (Application); AMS might best be characterized as a messaging “middleware” protocol. As such, it relies on the capabilities of underlying Transport-layer protocols to accomplish the actual copying of a message from the memory of the sending node to the memory of the receiving node. It additionally relies on those capabilities to accomplish the transmission of *meta-AMS* (or *MAMS*) messages to and from registrars, subject servers, and configuration servers that enables the dynamic self-configuration of AMS message spaces.

For any given AMS continuum, some common transport service must be utilized for MAMS traffic by all communicating entities involved in the operations of all message spaces in the continuum. The transport service selected for this purpose is termed the continuum’s *Primary Transport Service* or *PTS*. Selection of the PTS for a continuum is an implementation matter; the “Bundling” protocol of the Delay-Tolerant Networking architecture is a plausible choice, but for some continua an implementation based on UDP/IP may be more appropriate.

The PTS clearly can also be used for application message exchange among all nodes in a continuum, as it must be universally available for MAMS message exchange. In some cases, however, improved application performance can be achieved by using a different transport service for message exchange between nodes that share access to some especially convenient communication medium, such as a shared-memory message queue. These performance-optimizing transport services are termed *Supplementary Transport Services* or *STSs*.

2.3.2 Registrar registration

Every message space always comprises at least one zone, and each node resides within (is registered in) some zone; in the simplest case all nodes of the message space reside in the same zone. Each zone is served by a single registrar, which is responsible for monitoring the health of all nodes in the zone and for propagating four kinds of message space configuration changes: node registrations and terminations, subscription assertions, and subscription cancellations. On receipt of one of these reconfiguration messages from a node in its own zone, the registrar immediately propagates the message to every other node in the same zone and then to the registrars of all other zones in the message space; on receiving such a message from a remote zone's registrar, the registrar propagates it to all nodes in its own zone.

The registrars themselves register with the configuration server for the continuum within which the message space is contained. A list of all possible network locations for the configuration server, in order of descending preference, must be “well known” to the registrars for all message spaces in the continuum, and each continuum must have a configuration server in operation at one of those locations at all times in order to enable registrars and nodes to register. (The manner in which this latter requirement is satisfied is an implementation matter. One advantage of selecting the DTN Bundling protocol as the PTS for a continuum is that the “resilient delivery” features of Bundling may provide a simple solution for this problem: when a configuration server registers with its bundle agent, it can supply a script that will reanimate that server automatically in the event that a bundle carrying a MAMS message arrives following a configuration server crash.)

All registrars and nodes of the same message space must register through the same configuration server. The registrars and nodes for any number of different message spaces may register with the same configuration server.

2.3.3 Node registration

Each node has a name, an application-specific ASCII string containing no whitespace, which generally connotes its function within the application but need not uniquely identify it within its message space. Each node also has one or more “network identities”, one for each underlying transport service on which the node is prepared to receive messages.

A new node joins a message space by registering itself within some zone of the message space, i.e., by announcing its name and its network identities to the zone's registrar. However, knowledge of how to communicate with that registrar can't be hard-coded into the node because the relevant registrar might be running at different network locations at different times.

For this reason, the first step in registering a new node is to contact the configuration server at its well-known network location. The configuration server tells the new node how to contact its registrar. The node obtains a unique *node ID* from the registrar and thereby registers. The registrar ensures that all other nodes in the message space learn the

new node's name, node ID, and network identities. Those nodes in turn announce their own names, node IDs, and network identities to the new node.

2.3.4 Monitoring node health

In order to acquire and maintain accurate knowledge of the configuration of a message space (for application purposes, and also to avoid wasting resources on attempts to send messages to nonexistent nodes or per concluded subscriptions), it is important for each registrar always to detect the terminations of nodes in its zone. When a node terminates under application control it automatically notifies its registrar that it is stopping.

However, if a node crashes – or the host on which a node resides is simply powered off or rebooted – no notification is transmitted to the registrar. For this reason, nodes automatically send "heartbeat" messages to their registrars at fixed intervals, normally every 20 seconds. The registrar interprets three successive missing heartbeats as an indication that the node has terminated.

Whenever it detects the termination of a node (either an overt termination or a termination imputed from heartbeat failure), the registrar informs all other nodes in the message space of the node's demise. When the termination is imputed from a heartbeat failure, the registrar also tries to send a message to the terminated node telling it that it has been presumed dead; if this node is in fact still running (perhaps it had hung trying to write on a blocked file descriptor), it terminates immediately on receipt of this message. This minimizes system confusion due to other application behavior that may have been triggered by the imputed termination.

2.3.5 Monitoring registrar health

In addition to monitoring the heartbeats of all nodes in its zone, each registrar issues its own heartbeats to those nodes on the same cycle. Each node interprets three successive missing registrar heartbeats as an indication that the registrar itself has crashed. On detecting a registrar crash, the node presumes that the registrar has been restarted since it crashed; it re-queries the configuration server to determine the new network location of the registrar and resumes exchanging heartbeats.

This presumption is reasonable because the reciprocal heartbeat monitoring relationship between a registrar and its nodes is replicated in the relationship between the configuration server and all registrars, but on a slightly shorter cycle. The configuration server interprets three successive missing registrar heartbeats as an indication that the registrar has crashed; on detecting such a crash it automatically restarts the registrar, possibly on a different host, so by the time the registrar's nodes detect its demise it should already be running again. (The same is true of subject servers, as discussed later.)

Since the maximum heartbeat interval is twenty seconds, within the first sixty seconds after restart the registrar will have received heartbeat messages from every node that is still running in the zone and will therefore know accurately the configuration of the zone. This accurate configuration information must be delivered to new nodes at the time they start up (so that they in turn are qualified to orient a newly-restarted registrar to the zone's

configuration in the event that the registrar crashes). For this reason, during the first sixty seconds after the registrar restarts it accepts only connections from existing nodes in the zone; if it accepted a connection from a new node before being certain of the status of all old ones, it would run the risk of delivering incorrect information to the new node.

2.3.6 Configuration service fail-over

It's of course also possible for a configuration server to be killed (or for its host to be rebooted, etc.). Each registrar interprets three successive missing configuration server heartbeats as an indication that the configuration server has crashed. On detecting such a crash, the registrar begins cycling through all of the well-known possible network locations for the continuum's configuration server, trying to re-establish communication after the server's restart, possibly at an alternate network location. While it is doing so it is not issuing heartbeats or responding to other messages, so eventually all nodes may infer that their registrars have crashed and therefore begin re-querying the now-dead configuration server to re-establish communication with their registrars; when the configuration server fails to respond, the nodes too will begin cycling through network locations seeking a restarted configuration server. While they are doing so, they too will not be responding to messages, so all message space activity will eventually come to a halt.

When the configuration server is restarted at one of its well-known network locations, however, all registrars will eventually find it and re-announce themselves to it, so that when application nodes finally find it they can re-establish communication with their registrars; all application processing will thereupon resume.

It is possible, in this sort of failure scenario, that multiple configuration servers may be operating concurrently for a brief time; for example, the perceived failure of a configuration server might have been caused by a transient network connectivity failure rather than an actual server crash. To resolve this sort of situation, each running configuration server periodically sends an "I am running" MAMS message to every lower-ranking configuration server network location in the well-known list of configuration server locations. When a configuration server receives such a message, it immediately terminates; all registrars and nodes that were communicating it will detect its disappearance and search again for the highest-ranking reachable configuration server, eventually bringing the continuum back to orderly operations.

2.3.7 Configuration resync

Finally, every registrar can optionally be configured to re-advertise to the entire message space the detailed configuration of its zone (all active nodes, all subscriptions) at some user-specified frequency, e.g., once per minute. This capability is referred to as *configuration resync*. Configuration resync generates additional message traffic, and it may be unnecessary in extremely simple or extremely stable operating environments. But it does ensure that every change in application message space configuration will eventually be propagated to every node in the message space, even if some MAMS

messages are lost and even if an arbitrary number of registrars had crashed at the time the change occurred.

Taken together, these measures make AMS applications relatively fault tolerant:

- When a node crashes, its registrar detects the loss of heartbeat within three heartbeat intervals and notifies the rest of the message space. Message transmission everywhere is unaffected.
- When a registrar crashes, its configuration server detects the loss of heartbeat within three heartbeat intervals and restarts the registrar. During the time that the zone has no registrar, transmission of application messages among nodes of the message space is unaffected, but the heartbeat failures of crashed nodes are not detected and reconfiguration messages originating in the zone (registrations, terminations, subscription assertions, and subscription cancellations) are not propagated to any nodes. However, after the registrar is restarted it will eventually detect the losses of heartbeat from all crashed nodes and will issue obituaries to the message space, and if configuration resync is enabled it will eventually re-propagate the lost reconfiguration messages.
- When a configuration server crashes, all activity may eventually come to a standstill. But no application nodes fail (at least, not because of communication failure), and on restart of the configuration server all activity resumes.

2.3.8 Subject service

Message subjects, as noted earlier, are integers with application-defined semantics. This minimizes the cost of including subject information (in effect, message type) in every message, and it can make processing in an AMS implementation simpler and faster: subscription and message handling information may be recorded in dynamically allocated and possibly sparse arrays that are indexed by subject number.

This implementation choice, however, would require that message management control arrays be large enough to accommodate the largest subject numbers used in the application. The use of extremely large subject numbers would therefore cause these arrays to consume huge amounts of memory. In general, it is best for an AMS application to use the **smallest** subject numbers possible, starting with 1.

One way to ensure this is to require that applications cite message subjects by symbolic name, rather than cite the subject numbers themselves, and let the AMS infrastructure automatically assign the smallest unused subject numbers to subjects as their names are declared. To support the mapping of subject names to numbers, and vice versa, subject definition services are provided by the message space's subject server.

The subject server manages a private database of subject definitions for the message space. Each subject definition pairs a subject name (an application-specific ASCII string containing no whitespace) with a subject number and, optionally, a message content format string. The subject server itself assigns numbers sequentially (starting at 1) to

subject names, in the order in which the subject names are declared to it by application nodes. It responds to subject number queries by returning the corresponding subject names and to subject name queries by returning the previously assigned subject numbers.

Subject names are also the basis for Remote AMS communication, described below. Since subject names might be declared in different sequences within different message spaces – for the same application instance, but in different continua – they may be mapped to different numbers. But so long as subject naming consistency is maintained when applications are developed, the inter-continuum Remote AMS communication between subject servers will accurately replicate subscription and publication in all message spaces.

The AMS heartbeat discipline monitors the health of subject servers just as it monitors the health of registrars: the configuration server interprets three successive registrar heartbeat delivery failures as an indication that the subject server has crashed. On detecting such a crash it automatically restarts the subject server, possibly on a different host. Subject servers, again like registrars, also expect heartbeats from the configuration server and respond to a configuration server failure in the same way that registrars do.

2.3.9 Remote AMS message exchange

Because issuance of an asynchronous message need not suspend the operation of the issuing node until a response is received, AMS's message exchange model enables a high degree of concurrency in the operations of data system modules; it also largely insulates applications from variations in signal propagation time between points in the AMS continuum.

However, some critical MAMS communication is unavoidably synchronous in nature: in particular, a newly registering node must wait for responses from the configuration server, the registrar, and the other nodes in its message space before proceeding with application activity. For this reason, the core AMS protocol is only suitable for use in operational contexts characterized by generally uninterrupted network connectivity and relatively small and predictable signal propagation times, such as the Internet or a stand-alone local area network. All nodes of any single AMS continuum must be running within one such “low-latency” network.

AMS application messages may readily be exchanged between nodes in different AMS continua, however, by means of the auxiliary Remote AMS (RAMS) protocol. RAMS procedures are executed by the subject servers of message spaces:

- Each subject server opens persistent, private RAMS communication channels to the subject servers of other message spaces of the same application instance, in other continua.
- The interconnected subject servers use these channels to forward subscription assertions and cancellations among themselves. Each subject server, acting as a

node, subscribes locally to all subjects that are of interest in any of the linked message spaces.

- On receiving its copy of a message of any of these subjects, the subject server uses RAMS to forward the message to every other subject server whose message space contains at least one other node that has subscribed to messages of that subject.
- On receiving a message via RAMS from a subject server in another message space, the subject server simply publishes the message in its own message space.

In this way the RAMS protocol enables the free flow of application messages across arbitrarily long deep space links while protecting efficient utilization of those links: only a single copy of any message is ever transmitted on any RAMS channel, no matter how many subscribers will receive copies when the message reaches its destination continuum.

Again, this extension of the publish/subscribe model to interplanetary communications is invisible to application nodes. Application functionality is unaffected by these details of network configuration, and the only effects on behavior are those that are intrinsic to variability in message propagation latency.

3 Service descriptions

3.1 Services provided to the application

3.1.1 Summary of primitives

The AMS service shall consume the following request primitives:

- a) **Register.request;**
- b) **Unregister.request;**
- c) **Assert_subscription.request;**
- d) **Cancel_subscription.request;**
- e) **Publish.request;**
- f) **Send.request;**
- g) **Reply.request;**
- h) **Declare_subject.request;**
- i) **Look_up_subject.request.**

The AMS service shall deliver the following indication primitives:

- a) **Message.indication;**
- b) **Reply.indication;**
- c) **Fault.indication;**
- d) **Register.indication;**
- e) **Unregister.indication;**
- f) **Assert_subscription.indication;**
- g) **Cancel_subscription.indication;**
- h) **Subject.indication.**

3.1.2 Service primitive parameters

NOTE – The availability and use of parameters for each primitive are indicated in the definitions of primitives below, where parameters that are optional are identified with square brackets [thus]. The following parameter definitions apply.

- 3.1.2.1 The *application name* parameter shall identify the application served by a message space's application instance.
- 3.1.2.2 The *authority name* parameter shall identify the organizational unit that is responsible for a message space's application instance. The combination of application name and authority name shall uniquely identify a message space.
- 3.1.2.3 The *zone name* parameter shall identify, within a given message space, some administrative subset of nodes.
- 3.1.2.4 The *node name* parameter shall indicate the functional nature of a node.
- 3.1.2.5 The *node specification* parameter shall be an ASCII text string that characterizes the manner in which a node is prepared to receive AMS messages. The node specification shall comprise a comma-separated list of one or more *port specifications* in declining order of preference; that is, the port on which the node most prefers to receive messages is specified first, followed by the next-most-preferred port, and so on. Each port specification shall be the concatenation of a *transport service name*, an "equals" (=) symbol, and a *transport service endpoint specification*, in that order. The transport service name shall be the name of the AMS continuum's primary transport service or the name of one of the continuum's supplementary transport services. The syntax in which the transport service endpoint specification is represented shall be specific to the indicated transport service. Definitions of valid endpoint specification syntax for all recognized transport services are given in Annex C.

- 3.1.2.6 The *node ID* parameter shall uniquely identify a node within the message space in which it is registered.
- 3.1.2.7 The *subject name* parameter shall indicate the general nature of the application data in a message.
- 3.1.2.8 The *subject format* parameter shall provide information enabling an application to parse the application data in any message of a given subject.
- 3.1.2.9 The *context* parameter shall identify the application context in which an original (non-reply) message was sent, if any.
- 3.1.2.10 The *content length* parameter shall indicate the length (in octets) of the information in a message.
- 3.1.2.11 The *content* parameter shall be an array of zero or more octets comprising the application data in a message.
- 3.1.2.12 The *fault expression* parameter shall indicate the nature of an operational fault encountered by AMS. The syntax of fault expressions is an implementation matter.

3.1.3 Register.request

3.1.3.1 Function

The **Register.request** primitive shall be used by the node to commence its participation in a message space.

3.1.3.2 Semantics

Register.request shall provide parameters as follows:

Register.request (application name,
authority name,
zone name,
node name,
node specification)

3.1.3.3 When Generated

Register.request is generated by the node at any time while the node is not currently participating in its message space.

3.1.3.4 Effect on Receipt

Receipt of **Register.request** shall cause AMS to add the node to the indicated message space zone.

3.1.3.5 Additional Comments

None.

3.1.4 **Unregister.request**

3.1.4.1 Function

The **Unregister.request** primitive shall be used by the node to terminate its participation in a message space.

3.1.4.2 Semantics

Unregister.request shall provide parameters as follows:

Unregister.request (node ID)

3.1.4.3 When Generated

Unregister.request is generated by the node at any time while the node is participating in its message space.

3.1.4.4 Effect on Receipt

Receipt of **Unregister.request** shall cause AMS to remove the node from the indicated message space zone.

3.1.4.5 Additional Comments

The node ID provided in the **Unregister.request** primitive must be the one that was provided in the **Register.indication** primitive that notified the node of its own successful registration.

3.1.5 **Assert_subscription.request**

3.1.5.1 Function

The **Assert_subscription.request** primitive shall be used by the node to subscribe to published messages of a specific subject.

3.1.5.2 Semantics

Assert_subscription.request shall provide parameters as follows:

Assert_subscription.request (subject name)

3.1.5.3 When Generated

Assert_subscription.request is generated by the node at any time while the node is participating in its message space.

3.1.5.4 Effect on Receipt

Receipt of **Assert_subscription.request** shall cause AMS to notify all nodes in the message space of the node's subscription to the indicated message subject.

3.1.5.5 Additional Comments

None.

3.1.6 **Cancel_subscription.request**

3.1.6.1 Function

The **Cancel_subscription.request** primitive shall be used by the node to subscribe to published messages of a specific subject.

3.1.6.2 Semantics

Cancel_subscription.request shall provide parameters as follows:

Cancel_subscription.request (subject name)

3.1.6.3 When Generated

Cancel_subscription.request is generated by the node at any time while the node is participating in its message space.

3.1.6.4 Effect on Receipt

Receipt of **Cancel_subscription.request** shall cause AMS to notify all nodes in the message space that the node is no longer subscribed to the indicated message subject.

3.1.6.5 Additional Comments

None.

3.1.7 **Publish.request**

3.1.7.1 Function

The **Publish.request** primitive shall be used by the node to publish a message.

3.1.7.2 Semantics

Publish.request shall provide parameters as follows:

Publish.request (subject name,
content length,
[content],
[context])

3.1.7.3 When Generated

Publish.request is generated by the node at any time while the node is participating in its message space.

3.1.7.4 Effect on Receipt

Receipt of **Publish.request** shall cause AMS to construct a message as indicated and send one copy of that message to every node in the message space that is currently subscribed to the indicated message subject.

3.1.7.5 Additional Comments

None.

3.1.8 **Send.request**

3.1.8.1 Function

The **Send.request** primitive shall be used by the node to send a message privately to a single node.

3.1.8.2 Semantics

Send.request shall provide parameters as follows:

Send.request (node ID,
subject name,
content length,
[content],
[context])

3.1.8.3 When Generated

Send.request is generated by the node at any time while the node is participating in its message space.

3.1.8.4 Effect on Receipt

Receipt of **Send.request** shall cause AMS to construct a message as indicated and send it to the specified node.

3.1.8.5 Additional Comments

Node ID identifies the node to which the message is to be sent. Context, if specified, identifies context information that is meaningful to the sending node.

3.1.9 **Reply.request**

3.1.9.1 Function

The **Reply.request** primitive shall be used by the node to reply to a message sent by some node.

3.1.9.2 Semantics

Reply.request shall provide parameters as follows:

Reply.request (node ID,
subject name,
content length,
[content],
context)

3.1.9.3 When Generated

Reply.request is generated by the node at any time while the node is participating in its message space.

3.1.9.4 Effect on Receipt

Receipt of **Reply.request** shall cause AMS to construct a message as indicated and send it to the specified node.

3.1.9.5 Additional Comments

Node ID must identify the node that sent some previously received message, and context must be the context provided with that message (which will be meaningful only to that node).

3.1.10 **Declare_subject.request**

3.1.10.1 Function

The **Declare_subject.request** primitive shall be used by the node to declare the validity of a specified message subject.

3.1.10.2 Semantics

Declare_subject.request shall provide parameters as follows:

Declare_subject.request (subject name,
[subject format])

3.1.10.3 When Generated

Declare_subject.request is generated by the node at any time while the node is participating in its message space.

3.1.10.4 Effect on Receipt

Receipt of **Declare_subject.request** shall cause AMS to recognize the validity of the indicated subject and to note the format in which the content of every message of this subject is represented (if specified). If the subject was previously declared, the new subject format (if specified) supersedes the subject's current format.

3.1.10.5 Additional Comments

AMS message content formats are expected to be generally static. The AMS protocol does not include provisions for actively propagating revised formats to nodes that might previously have cached older formats, so cache coherency failures are possible. Future upgrades to AMS to redress this deficiency may eventually prove necessary.

3.1.11 **Look_up_subject.request**

3.1.11.1 Function

The **Look_up_subject.request** primitive shall be used by the node to determine the validity of a specified message subject and the format in which the content of every message of this subject is represented (if defined).

3.1.11.2 Semantics

Look_up_subject.request shall provide parameters as follows:

Look_up_subject.request (subject name)

3.1.11.3 When Generated

Look_up_subject.request is generated by the node at any time while the node is participating in its message space.

3.1.11.4 Effect on Receipt

Receipt of **Look_up_subject.request** shall cause AMS to validate the indicated subject and report on the format in which the content of every message of this subject is represented (if defined).

3.1.11.5 Additional Comments

None.

3.1.12 **Message.indication**

3.1.12.1 Function

The **Message.indication** primitive shall be used to deliver AMS original (non-reply) message information to the node.

3.1.12.2 Semantics

Message.indication shall provide parameters as follows:

Message.indication (node ID,
subject name,
content length,
[content],
[context])

3.1.12.3 When Generated

Message.indication is generated upon reception of an original (non-reply) message from a node.

3.1.12.4 Effect on Receipt

The effect on reception of **Message.indication** by a node is undefined.

3.1.12.5 Additional Comments

Node ID identifies the node that sent or published the message.

3.1.13 Reply.indication

3.1.13.1 Function

The **Reply.indication** primitive shall be used to deliver AMS reply message information to the node.

3.1.13.2 Semantics

Reply.indication shall provide parameters as follows:

Reply.indication (node ID,
subject name,
content length,
[content],
context)

3.1.13.3 When Generated

Reply.indication is generated upon reception of a reply message from a node.

3.1.13.4 Effect on Receipt

The effect on reception of **Reply.indication** by a node is undefined.

3.1.13.5 Additional Comments

Node ID identifies the node that sent the reply message. Context identifies the context in which the antecedent message (the message to which the reply message is a response) was sent.

3.1.14 Fault.indication

3.1.14.1 Function

The **Fault.indication** primitive shall be used to indicate an AMS fault condition to the node.

3.1.14.2 Semantics

Fault.indication shall provide parameters as follows:

Fault.indication (fault expression)

3.1.14.3 When Generated

Fault.indication is generated when AMS encounters a fault condition.

3.1.14.4 Effect on Receipt

The effect on reception of **Fault.indication** by a node is undefined.

3.1.14.5 Additional Comments

None.

3.1.15 **Register.indication**

3.1.15.1 Function

The **Register.indication** primitive shall be used to notify the node of the addition of some node (including itself) to the message space.

3.1.15.2 Semantics

Register.indication shall provide parameters as follows:

Register.indication (node ID,
node name)

3.1.15.3 When Generated

Register.indication is always generated upon addition of the node to its message space. **Register.indication** may optionally also be generated upon addition of any other node to the message space.

3.1.15.4 Effect on Receipt

The effect on reception of **Register.indication** by a node is undefined.

3.1.15.5 Additional Comments

The node ID in the **Register.indication** primitive that notifies the node of its own successful registration is the one that the node must provide in the **Unregister.request** primitive.

3.1.16 **Unregister.indication**

3.1.16.1 Function

The **Unregister.indication** primitive shall be used to notify the node of the removal of some node (including itself) from the message space.

3.1.16.2 Semantics

Unregister.indication shall provide parameters as follows:

Unregister.indication (node ID)

3.1.16.3 When Generated

Unregister.indication is always generated upon removal of the node from its message space. **Unregister.indication** may optionally also be generated upon removal of any other node from the message space.

3.1.16.4 Effect on Receipt

The effect on reception of **Unregister.indication** by a node is undefined.

3.1.16.5 Additional Comments

None.

3.1.17 **Assert_subscription.indication**

3.1.17.1 Function

The **Assert_subscription.indication** primitive shall be used to notify the node of a newly asserted subscription in the message space.

3.1.17.2 Semantics

Assert_subscription.indication shall provide parameters as follows:

Assert_subscription.indication (node ID,
node name,
subject name)

3.1.17.3 When Generated

Assert_subscription.indication is optionally generated upon assertion of a subscription by some node in the message space.

3.1.17.4 Effect on Receipt

The effect on reception of **Assert_subscription.indication** by a node is undefined.

3.1.17.5 Additional Comments

This indication is provided solely to facilitate message space configuration monitoring. Message publication is accomplished by AMS itself, not by the node, so generation of this indication is strictly optional.

3.1.18 **Cancel_subscription.indication**

3.1.18.1 Function

The **Cancel_subscription.indication** primitive shall be used to notify the node of a newly canceled subscription in the message space.

3.1.18.2 Semantics

Cancel_subscription.indication shall provide parameters as follows:

Cancel_subscription.indication (node ID,
node name,
subject name)

3.1.18.3 When Generated

Cancel_subscription.indication is optionally generated upon cancellation of a subscription by some node in the message space.

3.1.18.4 Effect on Receipt

The effect on reception of **Cancel_subscription.indication** by a node is undefined.

3.1.18.5 Additional Comments

This indication is provided solely to facilitate message space configuration monitoring. Message publication is accomplished by AMS itself, not by the node, so generation of this indication is strictly optional.

3.1.19 **Subject.indication**

3.1.19.1 Function

The **Subject.indication** primitive shall be used to report on the validity and (if specified) defined message format of a subject.

3.1.19.2 Semantics

Subject.indication shall provide parameters as follows:

Subject.indication (subject name,
[subject format])

3.1.19.3 When Generated

Subject.indication is generated upon reception of a subject report message from the subject server of the message space, which in turn is produced in response to a **Declare_subject.request** or **Look_up_subject.request** primitive.

3.1.19.4 Effect on Receipt

The effect on reception of **Subject.indication** by a node is undefined.

3.1.19.5 Additional Comments

None.

3.2 Services required of the transport system

TBD.

4 Protocol specification

4.1 General

All Meta-AMS PDUs (or “MPDUs”) shall be transmitted using the AMS continuum’s primary transport service.

4.2 Configuration procedures

4.2.1 Configuration server initialization

Upon commencing operations, the configuration server shall re-process all *announce_ss_daemon* and *announce_rs_daemon* MPDUs stored in its non-volatile “reboot object”, if any. Procedures for processing these MPDUs are discussed below.

Upon commencing operations and once per minute thereafter, the configuration server shall send an *I_am_running* MPDU to every “lower-ranking” network location at which the configuration server is authorized to operate, as indicated in the node’s Management Information Base. NOTE: since these network locations are listed in descending order of preference, a lower-ranking network location is defined as one that appears at some point in the list after the configuration server’s own network location.

On receipt of an *I_am_running* MPDU, the configuration server shall cease operations.

4.2.2 Configuration server location

When the network location of the continuum's configuration server is unknown:

- An *are_you_active* MPDU shall be sent to all network locations at which the configuration server is authorized to operate, as indicated in the node's Management Information Base.
- On receipt of an *are_you_active* MPDU, the configuration server shall return a *config_msg_ack* MPDU.
- On receipt of a *config_msg_ack* MPDU in response to an *are_you_active*, the network location of the originating configuration server shall be noted. If no responding *config_msg_ack* MPDU is received within 5 seconds, a **Fault.indication** primitive shall be delivered.
- This procedure shall be repeated until successful or until AMS activity is terminated.

4.2.3 Subject server initialization

Upon commencing operations, the subject server shall:

- Re-process all *subject_svc_request* MPDUs stored in its non-volatile "reboot object", if any. Procedures for processing these MPDUs are discussed below.
- Locate the configuration server.
- Send an *announce_ss_daemon* MPDU to the configuration server.

On receipt of an *announce_ss_daemon* MPDU, the configuration server shall:

- Determine whether or not a subject server for the indicated message space is already known to be running.
- If so, return a *rejection* MPDU.
- Otherwise:
 - Note the new subject server.
 - Retain the MPDU in the configuration server's non-volatile "reboot object" for automatic message space recovery.
 - Return a *config_msg_ack* MPDU.

- commence a 10-second heartbeat cycle for heartbeat exchange with the subject server. (See discussion of heartbeats below.)

On receipt of a *rejection* MPDU the subject server shall cease operations.

On receipt of a *config_msg_ack* MPDU, the subject server shall commence a 10-second heartbeat cycle for heartbeat exchange with the configuration server; see discussion of heartbeats below.

4.2.4 Subject server location

When the network location of the message space's subject server is unknown:

- The network location of the configuration server shall be determined as necessary.
- A *subject_svc_query* MPDU shall be sent to the configuration server.
- On receipt of a *subject_svc_query* MPDU, the configuration server shall return a *subject_svc_spec* MPDU indicating the network location of the subject server.
- On receipt of a *subject_svc_spec* MPDU in response to a *subject_svc_query*, the network location of the subject server shall be noted. If no responding *subject_svc_spec* MPDU is received within 5 seconds, the network location of the configuration server shall be deemed unknown.
- This procedure shall be repeated until successful or until AMS activity is terminated.

4.2.5 Registrar initialization

Upon commencing operations, the registrar shall:

- Locate the configuration server.
- Send an *announce_rs_daemon* MPDU to the configuration server.

On receipt of an *announce_rs_daemon* MPDU, the configuration server shall:

- Determine whether or not a registrar for the indicated zone of the indicated message space is already known to be running.
- If so, return a *rejection* MPDU.
- Otherwise:
 - If the indicated zone is previously unknown in this message space, i.e., the zone has not yet been assigned a zone number, then assign it the message space's smallest unused zone number.

- Note the registrar for this zone.
- Retain the MPDU in the configuration server's non-volatile "reboot object" for automatic message space recovery.
- Return a *zone_nbr* MPDU indicating the zone's assigned number.
- commence a 10-second heartbeat cycle for heartbeat exchange with the registrar. (See discussion of heartbeats below.)

On receipt of a *rejection* MPDU the registrar shall cease operations.

On receipt of a *zone_nbr* MPDU, the registrar shall:

- Commence a 10-second heartbeat cycle for heartbeat exchange with the configuration server; see discussion of heartbeats below.
- Send a *msg_space_query* MPDU to the configuration server.

On receipt of a *msg_space_query* MPDU, for each zone in the message space the configuration server shall return one *zone_spec* MPDU indicating the network location of the zone's registrar.

On receipt of each *zone_spec* MPDU, the registrar shall:

- Note the network location of the indicated zone's registrar.
- Send a *note_zone* MPDU to that registrar, announcing itself.

On receipt of a *note_zone* MPDU for a previously unknown zone, the receiving registrar will forward the MPDU to all nodes in its zone.

Each node, upon receiving a *note_zone* MPDU, shall note the name of the zone.

4.2.6 Registrar location

When the network location of the zone's registrar is unknown:

- The network location of the configuration server shall be determined as necessary.
- A *registrar_query* MPDU shall be sent to the configuration server.
- On receipt of a *registrar_query* MPDU, the configuration server shall return a *zone_spec* MPDU indicating the network location of the registrar.
- On receipt of a *zone_spec* MPDU in response to a *registrar_query*, the network location of the registrar shall be noted. If no responding *zone_spec* MPDU is

received within 5 seconds, the network location of the configuration server shall be deemed unknown.

- This procedure shall be repeated until successful or until AMS activity is terminated.

4.2.7 Node registration

On receipt of a **Register.request** primitive:

- The network location of the configuration server shall be determined.
- The network location of the registrar for the node's zone shall be determined.
- The node shall send a *node_registration* MPDU to that registrar.
- On receipt of a *node_registration* MPDU:
 - If the number of nodes registered in the zone is already equal to the maximum (as indicated in the registrar's Management Information Base), the registrar shall return a *rejection* MPDU indicating the reason for refusing the registration.
 - If the registrar has been operating for less than 60 seconds (three times the standard node heartbeat period) then it does not yet have an authoritative list of all nodes currently registered in the zone and therefore cannot yet construct a reliable "configuration needed" list for the new node. In this case the registrar shall return a *rejection* MPDU indicating the reason for refusing the registration.
 - Otherwise:
 - The registrar shall assign an unused node number to the node and return a *you_are_in* MPDU, indicating the node's own node number and containing a "configuration needed" list of the assigned node numbers of all other nodes currently registered in the zone.
 - The registrar shall also return one *note_zone* MPDU, indicating zone name, for each zone in the message space.
 - The registrar shall commence a 20-second heartbeat cycle for heartbeat exchange with the new node; see discussion of heartbeats below.
- On receipt of a *rejection* MPDU a **Fault.indication** primitive shall be delivered and the registration attempt shall be abandoned.

- As noted earlier, on receipt of each *note_zone* MPDU the indicated zone name shall be noted.
- On receipt of a *you_are_in* MPDU:
 - The node shall commence a 20-second heartbeat cycle for heartbeat exchange with the registrar; see discussion of heartbeats below.
 - The “configuration needed” list shall be noted.
 - An *I_am_starting* MPDU containing the node’s registration string shall be sent to the registrar.
- On receipt of an *I_am_starting* MPDU from a node in its zone, the registrar shall forward the MPDU to every other node in its zone and to every other registrar in the message space.
- On receipt of an *I_am_starting* MPDU from another registrar, the registrar shall forward the MPDU to every node in its zone.
- On a node’s receipt of an *I_am_starting* MPDU from a registrar:
 - The new node shall be noted.
 - Optionally, a **Register.indication** primitive shall be delivered.
 - An *I_am_here* MPDU containing the receiving node’s registration string and all of its current subscriptions shall be sent to the new node.
- On receipt of an *I_am_here* MPDU:
 - The issuing node’s registration string shall be noted.
 - Optionally, a **Register.indication** primitive shall be delivered.
 - For each of the issuing node’s subscriptions:
 - The subscription shall be noted.
 - Optionally, a **Assert_subscription.indication** primitive shall be delivered.
 - If the issuing node is registered in the new node’s own zone, it shall be removed from the “configuration needed” list.
 - If, at this time, the new node is already subscribed to one or more subjects, a *subscriptions* PDU containing all of those subscriptions shall be returned to the node that sent the *I_am_here* MPDU.

- On receipt of a *subscriptions* MPDU:
 - For each of the issuing node's subscriptions:
 - The subscription shall be noted.
 - Optionally, a **Assert_subscription.indication** primitive shall be delivered.

4.2.8 Unregistration

On receipt of an **Unregister.request** primitive:

- The network location of the registrar shall be determined.
- An *I_am_stopping* MPDU shall be sent to the registrar.

On receipt of an *I_am_stopping* MPDU from a node in its zone, the registrar shall note the termination of the node and shall forward the MPDU to every other node in its zone and to every other registrar in the message space.

On receipt of an *I_am_stopping* MPDU from another registrar, the registrar shall forward the MPDU to every node in its zone.

On a node's receipt of an *I_am_stopping* MPDU from a registrar:

- The node's removal shall be noted.
- Optionally, an **Unregister.indication** primitive shall be delivered.
- For each of the terminated node's subscriptions:
 - The subscription shall be forgotten.
 - Optionally, a **Cancel_subscription.indication** primitive shall be delivered.
- If the receiving node is the terminated node itself (i.e., the node's termination was imputed from a failure to deliver heartbeats to the registrar on a timely basis, as discussed below), the node shall terminate immediately.

4.2.9 Heartbeats

Upon expiration of any heartbeat period, a *heartbeat* MPDU shall be sent to the relevant communicating entity and a reciprocating *heartbeat* MPDU shall be expected from that entity. Whenever three successive heartbeat periods lapse after reception of a *heartbeat* MPDU before reception of another, an unplanned termination of the relevant communicating entity is imputed.

Imputed termination of the configuration server shall simply cause the network location of the configuration server to be unknown for future communication purposes, forcing re-location of the configuration server.

Upon imputed termination of a subject server or registrar, the configuration server shall attempt to restart the communicating entity at the same network location at which it was previously operating.

The effect of imputed registrar termination on each node in the registrar's zone shall be as follows:

- If the node's "configuration needed" list is non-empty (that is, the node has newly entered the message space and has not yet learned enough about all other nodes in the zone to be able to re-orient a restarted registrar), then the node terminates immediately.
- Otherwise:
 - The restarted registrar is located.
 - A *reconnect* MPDU indicating the numbers of all nodes in the zone shall be sent to the restarted registrar.
- On receipt of a *reconnect* MPDU from a node, the registrar shall:
 - Return a *you_are_dead* MPDU if the registrar has been operating for more than 60 seconds (during which time all nodes in the zone should already have contacted it); this serves to defend the registrar against spurious reconnections.
 - Return a *you_are_dead* MPDU if a *reconnect* MPDU had previously been received from some other node, and the node census in that *reconnect* MPDU did not include the node that issued this new *reconnect* MPDU; this serves to defend the registrar against inconsistent zone censuses.
 - Otherwise:
 - Commence a 20-second heartbeat cycle for heartbeat exchange with this node.
 - Return a *config_msg_ack* MPDU.
- On receipt of a *config_msg_ack* MPDU, the receiving node shall re-commence its 20-second heartbeat cycle for heartbeat exchange with the registrar.

Upon imputed termination of a node, the registrar shall send an *I_am_stopping* MPDU on behalf of this node to every other node in its zone and to every other registrar in the

message space. This will have the effect of unregistering the node in the manner discussed above.

On reception of a *heartbeat* MPDU from an unknown subject server or registrar, the configuration server shall return a *you_are_dead* MPDU.

On reception of a *heartbeat* MPDU from an unknown node, the receiving registrar shall return a *you_are_dead* MPDU.

On reception of a *you_are_dead* MPDU, the receiving communicating entity shall terminate immediately.

4.2.10 Resynchronization

Each registrar, upon expiration of its configuration resynchronization interval (as indicated in the registrar's Management Information Base), shall:

- Select one zone of the message space (possibly its own) for configuration resynchronization.
- Send a *note_zone* MPDU to that zone.
- Send a *zone_status* MPDU to that zone's registrar or, if the selected zone is the registrar's own zone, to every node in the zone.
- Send an *announce_status* MPDU to every node in its zone.

On receipt of an *announce_status* MPDU, the receiving node shall return a *my_status* MPDU to the registrar.

On receipt of each *my_status* MPDU, the registrar shall forward the MPDU as a *node_status* MPDU to the selected zone's registrar or, if the selected zone is the registrar's own zone, to every other node in the zone.

On receipt of a *zone_status* or *node_status* MPDU from another registrar, the receiving registrar shall forward the MPDU to every node in its zone.

On receipt of a *zone_status* or *node_status* MPDU, the receiving node shall note the current status of the zone or node. For each difference in registration or subscription between the new status and the status as previously noted by the receiving node, an optional **Assert_subscription.indication**, **Cancel_subscription.indication**, **Register.indication**, or **Unregister.indication** primitive shall be delivered as appropriate.

4.2.11 Subscription assertion

On receipt of an **Assert_subscription.request** primitive:

- The network location of the registrar shall be determined.
- A *subscribe* MPDU shall be sent to the registrar.

On receipt of a *subscribe* MPDU from a node in its zone, the registrar shall forward the message to every node in its zone and to every other registrar in the message space.

On receipt of a *subscribe* MPDU from another registrar, the registrar shall forward the message to every node in its zone.

On a node's receipt of a *subscribe* MPDU from a registrar, the subscription shall be noted and, optionally, an **Assert_subscription.indication** primitive shall be delivered.

4.2.12 Subscription cancellation

On receipt of a **Cancel_subscription.request** primitive:

- The network location of the registrar shall be determined.
- An *unsubscribe* MPDU shall be sent to the registrar.

On receipt of an *unsubscribe* MPDU from a node in its zone, the registrar shall forward the message to every node in its zone and to every other registrar in the message space.

On receipt of an *unsubscribe* MPDU from another registrar, the registrar shall forward the message to every node in its zone.

On a node's receipt of an *unsubscribe* MPDU from a registrar, the indicated subscription shall be forgotten and, optionally, a **Cancel_subscription.indication** primitive shall be delivered.

4.3 Dictionary procedures

On receipt of a **Declare_subject.request** primitive:

- The network location for the message space's subject server shall be determined as necessary.
- A *subject_svc_request* MPDU shall be sent to the subject server requesting declaration of the indicated subject.

On receipt of a **Look_up_subject.request** primitive:

- The network location for the message space's subject server shall be determined as necessary.
- A *subject_svc_request* MPDU shall be sent to the subject server requesting a report on the indicated subject.

On receipt of a *subject_svc_request* MPDU the subject server shall proceed as follows:

- If the requested service is declaration of a subject:
 - If the subject has not previously been declared:
 - The smallest subject number not currently assigned in this message space shall be assigned to the subject.
 - The subject's name, assigned number, and message content format (if specified) shall be catalogued for future reference. Note that the definition and management of the subject service catalogue is an implementation matter.
 - Otherwise, if the MPDU's argument specifies message content format, that format shall replace any format currently catalogued for this subject.
 - The MPDU shall be retained in the subject server's non-volatile "reboot object" for automatic subject catalog recovery.
- In any case, a *subject_definition* MPDU shall be issued to the node that issued the *subject_svc_request*.

On receipt of a *subject_definition* MPDU from the subject server:

- The definition shall be locally noted, to minimize the need for subsequent *subject_svc_requests*.
- A **Subject.indication** primitive shall be delivered.

4.4 Communication procedures

4.4.1 Message transmission

Whenever an AMS communication PDU is to be transmitted from one node to another, the transport service to use for this transmission shall be selected as follows:

- The access port names in the node registration string for the destination node shall be examined, in order, until one is encountered whose namespace is equal to the namespace in one of the transport service endpoint IDs by which the source node can transmit AMS messages, as indicated in the node's Management Information Base.
- If there is no such access port name, then transmission of the message is impossible; a **Fault.indication** primitive shall be delivered.
- Otherwise, the selected access port name shall be used as the destination endpoint for the PDU and the indicated transport service shall be used to accomplish the

transmission. If the transmission is determined to have failed for any reason, a **Fault.indication** primitive shall be delivered.

4.4.2 Publish

On receipt of a **Publish.request** primitive, a non-reply AMS PDU shall be constructed using the provided parameters and one copy of this PDU shall be transmitted to every node that is currently subscribed to the indicated subject.

4.4.3 Send

On receipt of a **Send.request** primitive, a non-reply AMS PDU shall be constructed using the provided parameters and the PDU shall be transmitted to the indicated node.

4.4.4 Reply

On receipt of a **Reply.request** primitive, a reply AMS PDU shall be constructed using the provided parameters and the PDU shall be transmitted to the indicated node.

4.4.5 Receive

On receipt of a non-reply AMS PDU, a **Message.indication** primitive shall be delivered.

On receipt of a reply AMS PDU, a **Reply.indication** primitive shall be delivered.

5 Protocol data units

5.1 Meta-AMS Protocol Data Units

5.1.1 Meta-AMS protocol data unit format

The protocol data units used to effect configuration and dictionary procedures are termed *Meta-AMS (MAMS)* PDUs. Each MPDU shall consist of a header in fixed format followed by supplemental data; the length of the supplemental data in an MPDU shall vary from zero to 4096 octets depending on the MPDU type indicated in the header.

The MPDU header shall consist of the fields shown in table 5-1. The MPDU header fields shall be transmitted in the order of presentation in table 5-1.

Table 5-1: MAMS PDU Header Fields

Field	Length (bits)	Values	Comment
Supplementary data flag	1	'0' — no supplementary data '1' — PDU has supplementary data	
MPDU type	7	See Table 5-2 below.	
Memo	32	See Table 5-3 below.	Semantics vary by MPDU type.

Argument	32	See Table 5-3 below.	If supplementary data flag is 1, argument indicates the length of the supplementary data. Otherwise, the semantics of the argument field vary by MPDU type.
----------	----	----------------------	---

5.1.2 Meta-AMS protocol data unit types

The MAMS protocol data unit type shall indicate the nature of the MPDU as noted in table 5-2.

Table 5-2: MAMS PDU Types

MPDU Type (decimal)	Function
00	(reserved)
01	heartbeat
02	rejection
03	you_are_dead
04	config_msg_ack
05	are_you_active
06	announce_ss_daemon
07	announce_rs_daemon
08	zone_nbr
09	(reserved)
10	zone_spec
11	note_zone
12	subject_svc_query
13	subject_svc_spec
14	subject_svc_request
15	subject_definition
16	msg_space_query
17	(reserved)
18	registrar_query
19	node_registration
20	you_are_in
21	I_am_starting
22	I_am_here
23	subscriptions

MPDU Type (decimal)	Function
24	subscribe
25	unsubscribe
26	I_am_stopping
27	reconnect
28	zone_status
29	announce_status
30	my_status
31	node_status
32	I_am_running
All other values	(reserved)

5.1.3 Meta-AMS Memo Values

5.1.3.1 Query number

A query number sequentially assigned by AMS for this entity.

5.1.3.2 Echo

The additive inverse of the memo number of the received MPDU that caused this MPDU to be sent.

5.1.3.3 Heartbeat source

If heartbeat is from configuration server, memo value is 1. If heartbeat is from subject server, memo value is 3. If heartbeat is from registrar, memo value is 2. If heartbeat is from a node, memo value is 4.

5.1.3.4 Reconnect echo

If MPDU is sent in response to a reconnect MPDU, then echo; otherwise zero.

5.1.3.5 Message source

If MPDU is directly from a node (not relayed), memo value is 4. Otherwise, memo value is zero.

5.1.3.6 Destination zone

If MPDU is sent to a registrar, memo value is that registrar's zone number. Otherwise, memo value is zero.

5.1.4 Meta-AMS Argument Values

5.1.4.1 SDL

Supplementary data length (SDL), the length of the MPDU's supplementary data including any terminating NULL.

5.1.4.2 Heartbeat sender

If heartbeat is from a node, the number of the node. Otherwise zero.

5.1.5 Meta-AMS Supplementary Data Values

5.1.5.1 General

The term *string* is used here to signify an array of text characters formed by the concatenation of one or more lexical tokens – and/or other strings – delimited by single spaces. All string text is in ASCII representation. The last character of the last text character in a supplementary data value is immediately followed by a NULL character, terminating the string.

5.1.5.2 Subject declaration

Subject declaration is the concatenation of a “!” character, the name of the subject, a space, and (if provided) the character string defining the subject format.

5.1.5.3 Subject lookup string

Subject lookup string is the concatenation of a “?” character and the name of the subject.

5.1.5.4 Subject definition

Subject definition is a string containing the ASCII representation of the number assigned to this subject, the name of the subject, and (if defined) the character string defining the subject format

5.1.5.5 MAMS endpoint ID

A MAMS endpoint ID is a PTS endpoint ID. For a continuum that uses UDP as its PTS, a MAMS endpoint ID is the concatenation of the ASCII representation of port number, a colon, and the ASCII representation of Internet host number.

5.1.5.6 Message space name

Message space name is a string containing the message space's application name and authority name.

5.1.5.7 Subject server boot string

Subject server boot string is string containing message space name, the name of the subject server's reboot object, and the subject server's MAMS endpoint ID.

5.1.5.8 Zone descriptor

A zone descriptor is a string containing the zone's name, the MAMS endpoint ID of its registrar, and ASCII representations of the maximum number of nodes allowed in the zone and the registrar's configuration resynchronization interval.

5.1.5.9 Registrar boot string

Registrar boot string is a string containing message space name and the zone descriptor for the registrar's zone.

5.1.5.10 Zone query string

Zone query string is a string containing message space name and the ASCII representation of zone number.

5.1.5.11 Zone specification

Zone specification is a string containing the ASCII representation of the zone's number and the zone descriptor for that zone.

5.1.5.12 Qualified zone name

Qualified zone name is a string containing message space name and zone name.

5.1.5.13 Node list

A node list is the concatenation of an 8-bit integer indicating the number of nodes in the list, followed by that number of 8-bit integer node numbers.

5.1.5.14 Enrollment string

Enrollment string is the concatenation of an 8-bit integer containing the assigned node number followed by a node list enumerating all the nodes in the zone (including the new one).

5.1.5.15 Access port name

Access port name is the concatenation of transport service namespace name, an equals (=) symbol, and the name of an endpoint in that namespace at which the node can receive AMS PDUs.

5.1.5.16 Node access string

Node access string is a string containing MAMS endpoint ID, a comma-delimited list of all access port names, and a comma-delimited list of the names of all transport service namespaces in which the node can transmit AMS PDUs.

5.1.5.17 Registration string

Registration string is a string containing node name, zone name, ASCII representation of node number, and node access string.

5.1.5.18 Subscription list

A subscription list is the concatenation of a 16-bit integer indicating the number of subscriptions in the list, followed by that number of 16-bit integer subject numbers.

5.1.5.19 Node status structure

A node status structure is the concatenation of a registration string (which is NULL-terminated, as it is the last string in the supplementary data value) and a subscription list.

5.1.5.20 Node ID structure

A node ID structure is the concatenation of two 8-bit integers indicating the zone number and node number of the node.

5.1.5.21 Declaration structure

A declaration structure is the concatenation of a node ID structure and a subscription list.

5.1.5.22 Subscription structure

A subscription structure is the concatenation of a node ID structure and a 16-bit integer indicating the subject number of the message subject.

5.1.5.23 Reconnect structure

A reconnect structure is the concatenation of an 8-bit integer indicating the node number of the node, followed by the name of the node (which is NULL-terminated, as it is the last string in the supplementary data value), followed by the zone's node list.

5.1.5.24 Zone status structure

A zone status structure is the concatenation of an 8-bit integer indicating the zone number of the zone, followed by the zone's node list.

5.1.6 Meta-AMS Protocol Data Unit Structures

The contents of MAMS protocol data units shall be as specified in Table 5-3.

Table 5-3: MAMS PDU Structures

Type	Memo	Argument	Supplementary Data
01	Heartbeat source	Heartbeat sender	none
02	Echo	0	none
03	Reconnect echo	0	none
04	Echo	0	none
05	Query number	0	none
06	Query number	SDL	Subject server boot string
07	Query number	SDL	Registrar boot string
08	Echo	Zone number	none
09	Query number	SDL	Zone query string
10	Echo	SDL	Zone specification
11	Zone number	SDL	Zone name
12	Query number	SDL	Message space name
13	Echo	SDL	MAMS endpoint ID
14	Query number	SDL	Subject declaration or subject lookup string
15	Echo	SDL	Subject definition
16	Query number	SDL	Message space name
17	Echo	Zone count	None
18	Query number	SDL	Qualified zone name
19	Query number	SDL	Node name
20	Echo	SDL	Enrollment string
21	Message source	SDL	Registration string
22	0	SDL	Node status structure
23	0	SDL	Declaration structure
24	Message source	SDL	Subscription structure
25	Message source	SDL	Subscription structure
26	Message source	SDL	Node ID structure

27	Query number	SDL	Reconnect structure
28	0	SDL	Zone status structure
29	0	Number of zone to sync with	none
30	Number of zone to sync with	SDL	Node status structure
31	Destination zone	SDL	Node status structure
32	0	0	none

5.2 AMS Communication PDUs

5.2.1 General

Each AMS PDU shall consist of a header in fixed format followed by zero or more octets of content. The length of the content in an AMS PDU shall vary as indicated by the content length field of the header.

The AMS PDU header shall consist of the fields shown in Table 5-4. The PDU header fields shall be transmitted in the order of presentation in Table 5-4.

Table 5-4: AMS Communication PDU Header Fields

Field	Length (bits)	Values	Comment
Source zone number	8		
Source node number	8		
Destination zone number	8		
Destination node number	8		
Message subject number	16		
Context number	32		
Context cycle number	16	Zero.	For future use.
Content length	32		

5.2.2 Replies

For any AMS PDU issued in response to a **Publish.request** primitive or a **Send.request** primitive, context number shall be either zero or an integer greater than zero (inviting a reply). Such PDUs are termed “non-reply” messages.

For any AMS PDU issued in response to a **Reply.request** primitive, context number shall be the additive inverse of the context number of the message to which this reply message is a response. Such PDUs are termed “reply” messages.

6 Remote AMS (RAMS)

6.1 Remote AMS (RAMS) procedures

6.1.1 Initial declaration

Upon instantiation, the subject server for a message space identified by a given application name and authority name shall obtain from its Management Information Base (MIB) the DTN Bundling protocol endpoint IDs of all *counterpart subject servers*. Each such counterpart subject server shall be the subject server, in some other known AMS continuum, for the message space identified by the same application name and authority name (if any). The subject server shall use DTN Bundling to send to each counterpart subject server a RAMS PDU with content length -1, indicating initial declaration, and subject name length zero.

6.1.2 Orientation

Upon reception from a counterpart subject server of an initial declaration RAMS PDU, the subject server shall use DTN Bundling to send to that counterpart subject server one RAMS PDU with content length -2, indication subscription assertion, for each subject to which at least one node in its message space (aside from the subject server itself) is currently subscribed. Each such RAMS PDU shall contain the name of the relevant subject.

6.1.3 Assertion reporting

Upon detection of a subscription assertion within its message space, where the asserted subscription is for a subject to which no other node in the message space (aside from the subject server itself) is currently subscribed, the subject server shall use DTN Bundling to send to each counterpart subject server a RAMS PDU with content length -2, indicating subscription assertion, for that subject. The RAMS PDU shall contain the name of the relevant subject. This RAMS PDU shall signify that the subject server's continuum is subscribed to the indicated subject.

6.1.4 Cancellation reporting

Upon detection of a subscription cancellation within its message space, where the canceled subscription is for a subject to which no other node in the message space (aside from the subject server itself) is currently subscribed, the subject server shall use DTN Bundling to send to each counterpart subject server a RAMS PDU with content length -3, indicating subscription cancellation, for that subject. The RAMS PDU shall contain the name of the relevant subject. This RAMS PDU shall signify that the subject server's continuum is no longer subscribed to the indicated subject.

6.1.5 Assertion replication

Upon reception from a counterpart subject server of a subscription assertion RAMS PDU, for a subject to which no other continuum is currently subscribed, the subject server shall itself subscribe to that subject. The subject server shall not subscribe to any subjects under any other circumstances.

6.1.6 Cancellation replication

Upon reception from a counterpart subject server of a subscription cancellation RAMS PDU, for a subject to which no other continuum is currently subscribed, the subject server shall cancel its own subscription to that subject. The subject server shall not cancel its subscriptions to any subjects under any other circumstances.

6.1.7 Remote publication

On receiving a message for a subject to which it has subscribed, the subject server shall use DTN Bundling to send to its counterpart subject server in each continuum that is subscribed to this subject a RAMS PDU containing the content of that message. The RAMS PDU shall contain the name of the relevant subject.

6.1.8 Local re-publication

Upon reception from a counterpart subject server of a RAMS PDU with non-negative content length, the subject server shall publish a message whose subject and content are those of the RAMS PDU.

6.2 Remote AMS (RAMS) PDUs

Each RAMS PDU shall consist of a header in fixed format followed by from 1 to 255 octets of message subject name, followed by zero or more octets of content. The length of the message subject name in a RAMS PDU shall vary as indicated by the subject name length field of the header. The length of the content in a RAMS PDU shall vary as indicated by the content length field of the header.

The RAMS PDU header shall consist of the fields shown in Table 6-1. The PDU header fields shall be transmitted in the order of presentation in Table 6-1.

Table 6-1: RAMS PDU Header Fields

Field	Length (bits)	Values	Comment
Source continuum number	8		
Source zone number	8	0 if content length is negative; otherwise identifies the source node.	

Source node number	8	0 if content length is negative; otherwise identifies the source node.	
Destination continuum number	8		
Destination zone number	8	0 if content length is negative or message is a publication; otherwise identifies the destination node.	
Destination node number	8	0 if content length is negative or message is a publication; otherwise identifies the destination node.	
Context number	32	0 if content length is negative; otherwise, as specified by the source node.	
Context cycle number	16	0 if content length is negative; otherwise, as specified by the source node.	
Content length	32	-1 if PDU is an initial declaration; -2 if PDU is a subscription assertion; -3 if PDU is a subscription cancellation; otherwise, length of message content.	
Subject name length	8	0 if PDU is an initial declaration; otherwise, the length of the subject name of the message or subscription assertion or cancellation.	

7 Management information base

7.1 Node MIB

The MIB of each node shall include:

- The Primary Transport Service endpoint IDs of all network locations at which the configuration server for this continuum is authorized to operate, in descending order of preference.
- The period on which to issue and expect heartbeat messages to and from the registrar.
- The transport service endpoint IDs for all transport services by which the node is able to transmit AMS messages.

7.2 Registrar MIB

The MIB of each node shall include:

- The Primary Transport Service endpoint IDs of all network locations at which the configuration server for this continuum is authorized to operate, in descending order of preference.
- The period on which to issue and expect heartbeat messages to and from all nodes in the zone.
- The period on which to issue and expect heartbeat messages to and from the configuration server.
- The maximum number of nodes that may be registered in the zone at any one time.
- The period on which to autonomously re-advertise the configuration of the zone.

7.3 Subject Server MIB

The MIB of each subject server shall include:

- The Primary Transport Service endpoint IDs of all network locations at which the configuration server for this continuum is authorized to operate, in descending order of preference.
- The period on which to issue and expect heartbeat messages to and from the configuration server.
- The continuum number of the local continuum.
- The continuum numbers of all other AMS continua with which Remote AMS communication must be conducted.
- The DTN Bundling endpoint IDs of the counterpart subject servers in all other AMS continua with which Remote AMS communication must be conducted.

7.4 Configuration Server MIB

The MIB of each configuration server shall include:

- The Primary Transport Service endpoint IDs of all network locations at which the configuration server for this continuum is authorized to operate, in descending order of preference.
- The period on which to issue and expect heartbeat messages to and from subject servers and registrars.

Annex A: Informative References

TBD.

Annex B: Acronyms

TBD.

Annex C: Recognized Transport Services

tcp

TCP endpoint specifications may be represented in any of the following ways:

- *portnumber:hostname*
- *portnumber* [indicating that the hostname is the known name of the local host]
- ? [instructing AMS to select an available port on the local host]

The namespace for all TCP access port names is “tcp”. The endpoint ID of each TCP access port name is “*portnumber:hostname*”.

Each TCP access port name is therefore an ASCII string of the form “tcp=*portnumber:hostname*”.

udp

UDP endpoint specifications may be represented in any of the following ways:

- *portnumber:hostname*
- *portnumber* [indicating that the hostname is the known name of the local host]
- ? [instructing AMS to select an available port on the local host]

The namespace for all UDP access port names is “udp”. The endpoint ID of each UDP access port name is “*portnumber:hostname*”.

Each UDP access port name is therefore an ASCII string of the form “udp=*portnumber:hostname*”.

fifo

FIFO (i.e., named pipe) endpoint specifications may be represented in either of the following ways:

- *pathname*
- ? [instructing AMS to use “/tmp/*zonename.nodename*” as FIFO path name]

The namespace for all FIFO access port names is “*fifo**hostnumber*” where *hostnumber* is the 32-bit IP address of the host machine in which the FIFOs are constructed. [Note that Tramel communication via FIFO is only possible between nodes residing on a common host machine.] The endpoint ID of each FIFO access port name is “*pathname*”.

Each FIFO access port name is therefore an ASCII string of the form “*fifo**hostnumber*=*pathname*”.

vxpipe

VxWorks pipe endpoint specifications may be represented in either of the following ways:

- *queuelength*
- ? [instructing AMS to use a queue of default length]

The namespace for all VxWorks pipe access port names is “*vxpipe**hostnumber*” where *hostnumber* is the 32-bit IP address of the host machine in which the pipes are constructed. [Note that Tramel communication via VxWorks pipe is only possible between nodes residing on a common host machine.] The endpoint ID of each VxWorks pipe access port name is “*/pipe/owntaskID*”.

Each VxWorks pipe access port name is therefore an ASCII string of the form “*vxpipe**hostnumber*=*/pipe/owntaskID*”.

vxmsgq

VxWorks on-board message queue endpoint specifications may be represented in either of the following ways:

- *queuelength*
- ? [instructing AMS to use a queue of default length]

The namespace for all VxWorks on-board message queue access port names is “*vxmsgq**hostnumber*” where *hostnumber* is the 32-bit IP address of the host machine in which the message queues are constructed. [Note that Tramel communication via VxWorks on-board message queue is only possible between nodes residing on a common host machine.] The endpoint ID of each VxWorks on-board message queue access port name is “*msgqID*”.

Each VxWorks on-board message queue access port name is therefore an ASCII string of the form “*vxmsgq**hostnumber*=*msgqID*”.

smmsgq

VxWorks shared-memory (off-board) message queue endpoint specifications may be represented in any of the following ways:

- *queuelength*
- ? [instructing AMS to use a queue of default length]

The namespace for all VxWorks shared-memory message queue access port names is “smmsgqbusnumber” where *busnumber* is the administratively assigned 32-bit number (e.g., spacecraft ID) that uniquely identifies the set of processors sharing access to the memory board in which the shared-memory message queues are constructed. [Note that Tramel communication via VxWorks shared-memory message queue is only possible between nodes residing on a common bus.] The endpoint ID of each VxWorks shared memory message queue access port name is “msgqID”.

Each VxWorks shared-memory message queue access port name is therefore an ASCII string of the form “smmsgqbusnumber=msgqID”.