A Joint Standard of AASHTO, ITE, and NEMA

NTCIP 1101:1996 v01.12

National Transportation Communications for ITS Protocol

Simple Transportation Management Framework

December 2001

Includes Jointly Approved NTCIP 1101 Amendment 1 v08

This Adobe® PDF copy of an NTCIP standard is available at no-cost for a limited time through support from the U.S. DOT / Federal Highway Administration, whose assistance is gratefully acknowledged.

Published by

American Association of State Highway and Transportation Officials (AASHTO)
444 North Capitol Street, N.W., Suite 249
Washington, D.C. 20001

Institute of Transportation Engineers (ITE)
1099 14th Street, N.W., Suite 300 West
Washington, D.C. 20005-3438

National Electrical Manufacturers Association (NEMA)
1300 North 17th Street, Suite 1847
Rosslyn, Virginia 22209-3801

ACKNOWLEDGEMENTS

This publication was prepared by the NTCIP Base Standards and Protocols Working Group, which is a subdivision of the Joint Committee on the NTCIP. The Joint Committee is organized under a Memorandum of Understanding among the American Association of State Highway and Transportation Officials (AASHTO), the Institute of Transportation Engineers (ITE), and the National Electrical Manufacturers Association (NEMA). The Joint Committee on the NTCIP consists of six representatives from each of the standards organizations, and provides guidance for NTCIP development.

At the time that this document was prepared, the following individuals were active members of the NTCIP Base Standards and Protocols Working Group:

- Robert De Roche
- Robert Force (Chair)
- W. L. (Bud) Kent
- Dave Kingery
- Alexis Mousadi
- Mike Robinson
- Joerg 'Nu' Rosenbohm (former Chair)
- Kenneth Vaughn
- Hoi Wong

Other individuals providing input to the document, include:

- Joey Baumgartner

In addition to the many volunteer efforts, recognition is also given to those organizations who supported the efforts of the working groups by providing comments and funding for the standard, including:

- ARINC, Inc.
- Caltrans
- Eagle Traffic Control Systems
- Econolite Control Products, Inc.
- Ministry of Transportation, Ontario
- Odetics ITS, Inc.
- PB Farradyne, Inc.
- Peek Traffic Systems, Inc.
- Southwest Research Institute
- Vanasse, Hagen, & Brustlin, Inc.

FOREWORD

This document uses only metric units.

This publication describes the Simple Transportation Management Framework used for managing and communicating information between management stations and transportation devices. It presents the rules by which information is described and encoded for transmission. It also describes the Simple Transportation Management Protocol that is used to transfer information across communication links.

The text includes mandatory requirements in Annex A and Annex B that are defined as normative.

For more information about NTCIP standards, visit the NTCIP Web Site at http://www.ntcip.org. For a hardcopy summary of NTCIP information, contact the NTCIP Coordinator at the address below.

In preparation of this NTCIP document, input of users and other interested parties was sought and evaluated. Inquires, comments, and proposed or recommended revisions should be submitted to:

NTCIP Coordinator
National Electrical Manufacturers Association
1300 North 17th Street, Suite 1847
Rosslyn, VA 22209-3801
fax: (703) 841-3331
e-mail: ntcip@nema.org

Approvals

This document was separately balloted and approved by AASHTO, ITE, and NEMA after recommendation by the Joint Committee on the NTCIP. Each organization has approved this standard as the following standard type, as of the date:

AASHTO – Standard Specification; 1997
ITE – Software Standard; 1997
NEMA – Standard; 1996

History

From 1994 to 1999, this document was referenced as NEMA TS 3.2. However, to provide an organized numbering scheme for the NTCIP documents, this document is now referenced as NTCIP 1101. The technical specifications of NTCIP 1101 are identical to the former reference, except as noted in the development history below:

TS 3.2-1996 [assigned version 01.10]. July 1996 – Approved by NEMA and published in October 1996. October 1996 – Accepted as a Recommended Standard by the Joint Committee on the NTCIP. Approved by AASHTO in 1997 and approved by ITE in 1997.

NTCIP 1101:1996 [assigned version 01.11]. August 1999 – Assigned NTCIP 1101 document number in NTCIP Standards Bulletin B0038. NTCIP document cover used over TS 3.2 contents.

NTCIP 1101 Amendment 1, version 98.01.08. October 1998 – Version 98.01.07 accepted as a Recommended Amendment by the Joint Committee on the NTCIP, and referred for balloting and approval by NTCIP Standards Bulletin B0030 in May 1999. Amendment approved by AASHTO in October 1999; approved by ITE in January 2001; and approved by NEMA in December 1999.

NTCIP 1101:1996 v01.12, December 2001. January 2002 – Formatted for printing: incorporated Amendment 1 v08 into text; updated title page date and version number; revised front matter to conform to NTCIP 8002; edited Introduction; and updated headers, footers, and page numbers.

INTRODUCTION

The context of the NTCIP is one part of the Intelligent Transportation Systems standardization activities covering base standards, profiles, and registration mechanisms.

- Base Standards define procedures and rules for providing the fundamental operations associated with communications and information that is exchanged over fixed-point communications links.

- Profiles define subsets or combinations of base standards used to provide specific functions or services. Profiles prescribe particular subsets or options available in base standards necessary for accomplishing a particular function or service. This provides a basis for the development of uniform, nationally recognized conformance.

- Registration Mechanisms provide a means to specify and uniquely identify detailed parameters within the framework of base standards and/or profiles.

In 1992, the NEMA 3-TS Transportation Management Systems and Associated Control Devices Section began the effort to develop the NTCIP. The Transportation Section's purpose was to respond to user needs to include standardized systems communication in the NEMA TS 2 standard, Traffic Controller Assemblies. Under the guidance of the Federal Highway Administration's NTCIP Steering Group, the NEMA effort was expanded to include the development of communications standards for all transportation field devices that could be used in an Intelligent Transportation Systems (ITS) network.

In September 1996, an agreement was executed among AASHTO, ITE, and NEMA to jointly develop, approve, and maintain the NTCIP standards. In 1998, the Joint AASHTO/ITE/NEMA Committee on NTCIP formed the Base Standards and Protocols Working Group (BSP WG) to take over management of the original NEMA TS 3.2 standard and develop other base standards. The first meeting of the working group was in January 1999.

The BSP WG is concerned with the identification of applicable standards to address a transportation need, the definition of base standards and protocols in case existing standards do not address the identified need, and their documentation in Standards Publications. This standard, originally developed by NEMA as TS 3.2, contains both the definition of several base standards and an application profile that uses the base standards and others. As such, the intention of the BSP WG is maintain the standard as originally written and decompose it into its constituent parts. NTCIP 1101 will be replaced by three documents: (a) NTCIP 1102, which defines the Octet Encoding Rules (OER); (b) NTCIP 1103, which defines the Simple Transportation Management Protocol (STMP); and (c) NTCIP 8004, which defines the Structure and Identification of Management Information (SMI). The NTCIP Profiles Working Group will also develop NTCIP 2301, which defines the Simple Transportation Management Framework - Application Profile (AP-STMF).

If you are not willing to abide by the following notices, return these materials immediately.

Joint AASHTO, ITE, and NEMA
NTCIP Management Information Base and Data Dictionary
DISTRIBUTION NOTICE

To the extent and in the limited event these materials are distributed by AASHTO / ITE / NEMA in the form of a Management Information Base ("MIB") or Data Dictionary ("DD"), AASHTO / ITE / NEMA extends the following permissions:

(i) you may make and/or distribute unlimited copies (including derivative works) of the MIB, including copies for commercial distribution, provided that (a) each copy you make and/or distribute contains this Notice and (b) each derivative work of the MIB uses the same module name followed by "-," followed by your Internet Assigned Number Authority (IANA)-assigned enterprise number;
(ii) use of the MIB is restricted in that the syntax field may be modified only to reflect a more restrictive subrange or enumerated values;
(iii) the description field may be modified but only to the extent that: (a) only those bit values or enumerated values that are supported are listed; and (b) the more restrictive subrange is expressed.

These materials are delivered "AS IS" without any warranties as to their use or performance.

AASHTO / ITE / NEMA AND THEIR SUPPLIERS DO NOT WARRANT THE PERFORMANCE OR RESULTS YOU MAY OBTAIN BY USING THESE MATERIALS. AASHTO/ITE/NEMA AND THEIR SUPPLIERS MAKE NO WARRANTIES, EXPRESS OR IMPLIED, AS TO NONINFRINGEMENT OF THIRD PARTY RIGHTS, MERCHANTABILITY, OR FITNESS FOR ANY PARTICULAR PURPOSE. IN NO EVENT WILL AASHTO, ITE, OR NEMA OR THEIR SUPPLIERS BE LIABLE TO YOU OR ANY THIRD PARTY FOR ANY CLAIM OR FOR ANY CONSEQUENTIAL, INCIDENTAL, OR SPECIAL DAMAGES, INCLUDING ANY LOST PROFITS OR LOST SAVINGS, ARISING FROM YOUR REPRODUCTION OR USE OF THESE MATERIALS, EVEN IF AN AASHTO, ITE, OR NEMA REPRESENTATIVE HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES. Some states or jurisdictions do not allow the exclusion or limitation of incidental, consequential, or special damages, or the exclusion of implied warranties, so the above limitations may not apply to you.

Use of these materials does not constitute an endorsement or affiliation by or between AASHTO, ITE, or NEMA and you, your company, or your products and services.

Disclaimer

The information in this publication was considered technically sound by the consensus of persons engaged in the development and approval of the document at the time it was developed. Consensus does not necessarily mean that there is unanimous agreement among every person participating in the development of this document.

AASHTO, ITE, and NEMA standards and guideline publications, of which the document contained herein is one, are developed through a voluntary consensus standards development process. This process brings together volunteers and/or seeks out the views of persons who have an interest in the topic covered by this publication. While AASHTO, ITE, and NEMA administer the process and establish rules to promote fairness in the development of consensus, they do not write the document and they do not independently test, evaluate, or verify the accuracy or completeness of any information or the soundness of any judgments contained in their standards and guideline publications.

AASHTO, ITE, and NEMA disclaim liability for any personal injury, property, or other damages of any nature whatsoever, whether special, indirect, consequential, or compensatory, directly or indirectly resulting from the publication, use of, application, or reliance on this document. AASHTO, ITE, and

NEMA disclaim and make no guaranty or warranty, express or implied, as to the accuracy or completeness of any information published herein, and disclaims and makes no warranty that the information in this document will fulfill any of your particular purposes or needs. AASHTO, ITE, and NEMA do not undertake to guarantee the performance of any individual manufacturer or seller's products or services by virtue of this standard or guide.

In publishing and making this document available, AASHTO, ITE, and NEMA are not undertaking to render professional or other services for or on behalf of any person or entity, nor are AASHTO, ITE, and NEMA undertaking to perform any duty owed by any person or entity to someone else. Anyone using this document should rely on his or her own independent judgment or, as appropriate, seek the advice of a competent professional in determining the exercise of reasonable care in any given circumstances. Information and other standards on the topic covered by this publication may be available from other sources, which the user may wish to consult for additional views or information not covered by this publication.

AASHTO, ITE, and NEMA have no power, nor do they undertake to police or enforce compliance with the contents of this document. AASHTO, ITE, and NEMA do not certify, test, or inspect products, designs, or installations for safety or health purposes. Any certification or other statement of compliance with any health or safety-related information in this document shall not be attributable to AASHTO, ITE, or NEMA and is solely the responsibility of the certifier or maker of the statement.

CONTENTS

< This page is intentionally left blank. >

Section 1
GENERAL

1.1 SCOPE

The scope covers integrated management of transportation networks, networking devices and transportation specific equipment attached to NTCIP-based networks. This encompasses previous scopes addressed by other networking management frameworks within the Internet community, including the network management applications. It extends the scope beyond the Internet network management by addressing transportation equipment, networks and transportation management applications.

This includes the structure and identification of transportation information within transportation equipment, the protocols that act on defined information entities and the management applications used for controlling the transportation equipment. An extension to the Internet-standard Network Management Framework, called Simple Transportation Management Framework (STMF), allows for the operation, control, monitoring and configuration of NTCIP networks and the transportation equipment participating on the network. This includes networks utilizing legacy communication mediums such as 1200 bps FSK. This framework also details differences between Simple Transportation Management Framework (STMF) and the original Internet-Network Management Framework.

1.2 REFERENCES

For approved revisions, contact:

NTCIP Coordinator
National Electrical Manufacturers Association
1300 North 17th Street, Suite 1847
Rosslyn, VA 22209-3801

For draft revisions, which are under discussion by the relevant NTCIP Working Group, and recommended revisions of the NTCIP Joint Committee, visit the World Wide Web at http://www.ntcip.org.

The following publications are adopted in part, by reference in this publication, and are available from the organizations below.

1.2.1 Normative References

The following normative documents contain provisions, which through reference in this text, constitute provisions of this Standards Publication. By reference herein these publications are adopted, in whole or in part as indicated, in this SP.

International Organization for Standardization (ISO)
Case postale 56
CH-1211 Geneve 20
Switzerland

ISO 8824 *Specification of Abstract Syntax Notation One (ASN.1)*, December 1987

ISO 8825 *Specification of Basic Encoding Rules for Abstract Syntax Notation One (ASN.1),* December 1987

Internet Activities Board
available via electronic file transfer
use anonymous FTP from
NIC.DDN.MIL or DS.INTERNIC.NET

IAB STD 15 (RFC 1157) *Simple Network Management Protocol,* J. Case, M. Fedor, M. Schoffstall, J. Davin, May 1990

IAB STD 16 (RFC 1155) *Structure and Identification of Management Information for TCP/IP based Internets,* M. Rose, K. McCloghrie, May 1990, (RFC 1212) *Concise MIB Definitions*, M. Rose and K. McCloghrie, March 1991

1.2.2 Other References

The following publication(s) may be used to answer questions not covered by this standard.

Stallings, William, *SNMP, SNMPv2, and CMIP: The Practical Guide to Network-Management Standards*, Massachusetts, Addison-Wesley Publishing Company, 1993, ISBN 0-201-63331-0

1.3 TERMS

agent: The entity which receives commands and transmits responses to the received commands**.**

proxy agent: An entity which receives and responds to network management commands on behalf of another entity**.**

manager: The entity which sends commands to entities and processes their responses.

management information base: A structured collection or database of related managed objects defined using Abstract Syntax Notation One (ASN.1).

object: A representation of a resource that is managed.

OBJECT_TYPE: A macro used to define objects in Management Information Bases (MIB).

OBJECT IDENTIFIER: A unique name (identifier) that is associated with each type of object in a MIB. This is a defined ASN.1 type.

1.4 ABBREVIATIONS AND ACRONYMS

ANSI—American National Standards Institute, a standardization group that develops or adopts standards for the United States.

ASCII—American National Standard Code for Information Interchange.

ASN.1—Abstract Syntax Notation One, a formal language for describing information to be processed by computer, an ISO standard.

BER—Basic Encoding Rules, rules for encoding data for transmission used with ASN.1, an ISO standard.

CCITT—International Consultative Committee on Telegraph and Telephony.

IAB—Internet Activities Board, group in charge of authorizing RFCs for the purpose of standardizing Internet operations.

IANA—Internet Activities Numbering Authority.

IETF—Internet Engineering Task Force, a group chartered by the IAB to develop certain RFCs for standardization.

IP—Internet Protocol, primary network layer protocol used by Ethernet based LANs.

ISO—International Organization for Standardization, a group chartered with developing and authorizing worldwide standards.

LAN—Local Area Network, a network that is limited in distance, typically operates within a building or floor.

MIB—Management Information Base, a collection of management objects written in a subset of ASN.1 notation.

NVT—Network Virtual Terminal.

OER—Octet Encoding Rules, a variation of BER developed for use on low-bandwidth communications links, specified in this standards publication.

OSI—Open Systems Interconnection, a standardized model of communications protocols.

PDU—Protocol Data Unit, a part of transmitted data that contains information used by the protocol at a particular layer in the OSI stack.

RFC—Request for Comments, the name given to correspondence and standards by the IAB.

SMI—Structure of Management Information, a definition of how to create management objects and a hierarchical (tree-like) definition of nodes where management objects will be attached for unique identification.

SNMP—Simple Network Management Protocol, a communications protocol developed by the IETF, used for configuration and monitoring of network devices.

SNMPv2—Simple Network Management Protocol version 2, recent modification of SNMP that is undergoing evaluation by the Internet community.

SP—Standards publication, a formal compilation of NEMA text (NEMA Standard, Authorized Engineering Information, Suggested Standard for Future Design) on the same subject, which has gone through a consensus process and been approved according to NEMA Standardization Policies and Procedures.

STMF—Simple Transportation Management Framework.

STMP—Simple Transportation Management Protocol, a variation of SNMP developed by NEMA to address low-bandwidth communication links and real-time device monitoring.

TCP—Transmission Control Protocol, a transport-level protocol in common use on the Internet.

TLV—Type, Length, Value encoding.

TMIB—Transportation Management Information Base.

UDP—User Datagram Protocol, a simple transport level protocol in common use on the Internet.

WAN—Wide Area Network, a network that spans large geographical areas.

Section 2
SIMPLE TRANSPORTATION MANAGEMENT FRAMEWORK

The Simple Transportation Management Framework (STMF) specifies a set of rules and protocols for organizing, describing and exchanging transportation management information between transportation management applications and transportation equipment such that they interoperate with each other. It is based on the Internet-standard Network Management Framework. It is intended that a high level of compatibility, including inter-operability, be maintained between STMF and the Internet-standard Network Management Framework.

There are four major components within the STMF:

a. Structure and Identification of Management Information (SMI)

   This clause is based on the RFC 1155, SMI for TCP/IP. It does not define objects, but instead defines groups (branches) within the management information tree.

   Objects within the greater Internet community included are defined in the Management Information Base II (MIBII), RFC 1213.

b. Transportation Management Information Base (TMIB)

   TMIB contains the groups (branches) below the SMI groups. It contains objects defined by the NEMA NTCIP committee. Annex B contains the ASN.1 definition for the TMIB.

   TMIB also includes the NEMA MIB which contains the groups that NEMA controls, of which transportation is one subgroup. Annex A contains the ASN.1 definition for the NEMA MIB.

   MIBs are created in conformance with RFC 1212. RFC 1212 is a good guide for creating MIB modules or utilizing table column and row creates and deletes.

c. Simple Network Management Protocol version 1 (SNMP)

   SNMP is fully specified in RFC 1157, Simple Network Management Protocol (SNMP).

d. Simple Transportation Management Protocol (STMP)

   STMP is unique to this document and is specified in section 5.

Means exist for defining management information that is exchanged between a device being managed and an application managing the device. The unit of management information is termed a *managed object*. A *managed object* is the smallest entity that can be exchanged between a device and a management application. A collection of related managed objects is defined in a document termed a *Management Information Base* (MIB) module. A single module can define the complete Management Information Base; more often multiple modules are involved. A MIB or collection of MIBs, if organized in a hierarchical manner, constitute a tree-like structure.

A MIB can be organized in various methods. For the purposes of compliance with the pre-existing Internet structure, and because of its flexibility and extensibility, a tree structure is utilized. This tree structure was created within the ISO-CCITT-OSI community and is utilized for OSI System Management,

standards management (ISO, CCITT, ISO-CCITT, ANSI, IANA) and a variety of other information organization tasks. Each object can be uniquely identified by the location of the object within the tree.

Although the scope of a MIB within the Internet is all information within the Internet administrative tree, another common usage reduces the scope to that of the module or information within a subtree. In the STMF, a MIB can refer to either a tree or subtree within any administrative domain.

The SMI describes the common structures and identification scheme for the definition of management information. The SMI is specified in Abstract Syntax Notation One (ASN.1) and can be compiled by MIB compilers, but is not a MIB module. In order to compile MIB modules, SMI definitions (ASN.1 specifications) are included in a MIB module. The distinction is made because SMI defines how to create managed objects. SMI defines how to utilize ASN.1 in order to create and identify management information (objects) within a tree-like structure.

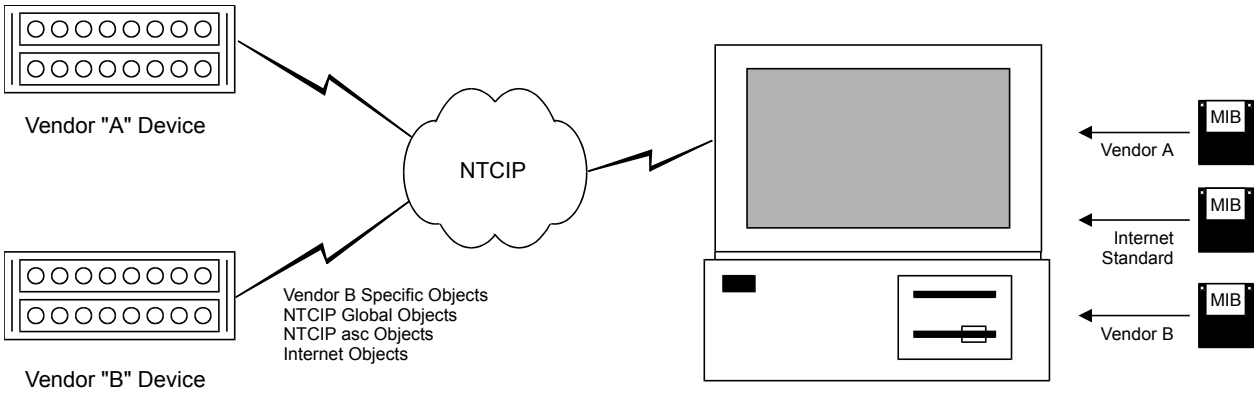

Figure 2–1
MIB INTEGRATION

After all of the information is identified, created, and organized the objects must be communicated. This requires an application layer communications protocol. That is the purpose of the SNMP and STMP components of STMF. The STMP protocol is specifically designed to satisfy some unique technical requirements of the transportation industry. SNMP is used because of its flexibility. These protocols are intended to work together in order to satisfy diverse requirements.

2.1 INTEGRATED MANAGEMENT

The result of creating and identifying managed objects using ASN.1 is a computer readable description of the information. These modules are ASCII-NVT text, not binary. MIB modules should be available for transportation products conforming to this standard. Users who purchase the equipment with a MIB module can utilize the module to update their management applications.

Management applications running on central control hosts can read this module, and other modules such as controller MIBs and manufacturer specific MIBs, in order to gain information about the capabilities of remote devices. See Figure 2–1. Many SNMP management applications can dynamically load and unload modules (MIBs) describing the information within remote networking devices.

Manufacturers supplying transportation equipment should make available manufacturer specific MIB modules for the equipment. All manufacturers need to obtain a vendor number from either NEMA or IANA

prior to creating MIB modules that have manufacturer specific data. Transportation equipment includes a list of supported MIBs.

2.2 REGISTRATION AUTHORITY HIERARCHY

The STMF is entirely compatible with the International Standards that cover the structure and management of information and the Internet-standard Network Management Framework. NEMA is registered in the Internet network management part of the *global name tree*. The official number for all NEMA management information is prefixed with, 1.3.6.1.4.1.1206, which can be represented by the textual naming convention, *iso.org.dod.internet.private.enterprises.nema* as shown in Figure 2–2.

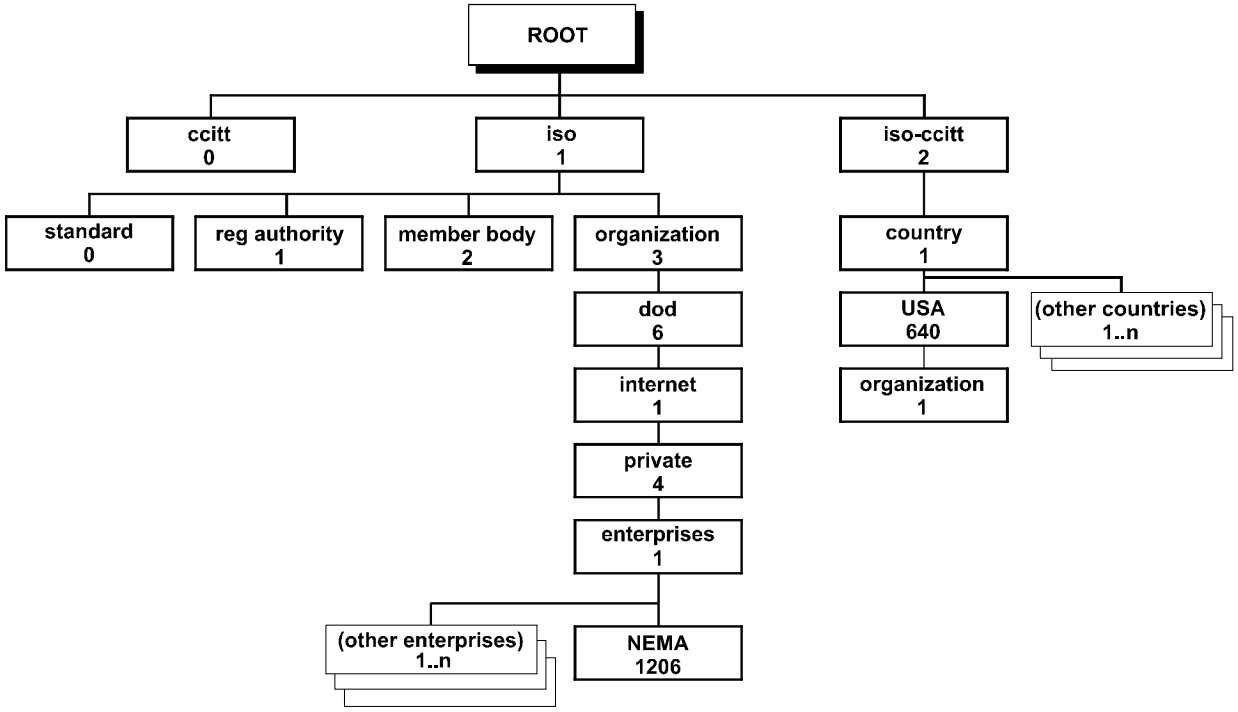

Figure 2–2
THE INTERNET AUTHORITY HIERARCHY

< This page has intentionally been left blank. >

Section 3
NEMA STRUCTURE AND IDENTIFICATION OF MANAGEMENT INFORMATION (NEMA_SMI)

3.1 INTRODUCTION

This section describes the common structures and identification scheme for the definition of management information used in NEMA standards. Included are descriptions of an object information model for network management along with a set of generic types used to describe management information. Formal descriptions of the structure are given using Abstract Syntax Notation One (ASN.1).

This section is largely concerned with organizational issues and administrative policy; it neither specifies the objects which are managed, nor the protocols used to manage those objects. These concerns are addressed by other sections.

This section is based on RFC 1155.

3.2 THE NEMA AUTHORITY

All the areas under the *nema* node shall be under NEMA control and constitute the NEMA MIB (See Figure 3-1.) In context this NEMA tree is a subtree within the Internet tree which is itself a subtree within the ISO tree.

NEMA defines what is in the *nema* subtree. SMI defines the specification of objects and also where all other MIBs or MIB modules are attached. New MIB modules utilized within the pre-existent tree are called MIB extensions. These do not require modification of the SMI. The only time the SMI is modified is if one of the basic groups (branches or subtrees) with the SMI is modified or a change is made to the definition or structure of object types. NEMA_SMI is an extension to the SMI. The NEMA_SMI does not define any objects, it only defines nodes upon which other MIBs may place objects.

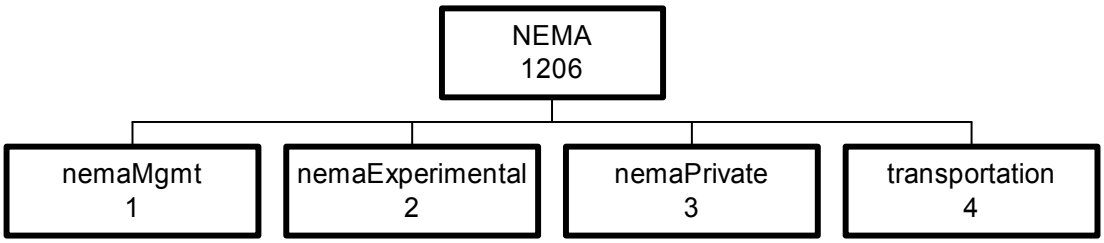

Figure 3-1
NEMA NODE

The OBJECT IDENTIFIERs for the major groups of the NEMA_SMI shall be:

1.3.6.1.4.1.1206.nemaMgmt(1)
1.3.6.1.4.1.1206.nemaExperimental(2)
1.3.6.1.4.1.1206.nemaPrivate(3)
1.3.6.1.4.1.1206.transportation(4)

All information under *transportation* is part of the TMIB, not the NEMA_SMI.

Additional objects or groups can be added at any time. The NEMA_SMI is maintained by NEMA. NEMA may designate areas as *not subject to NEMA* authority and NEMA can define methods for adding to this structure without NEMA's involvement. This is similar to what the Internet does with the *private.enterprises* section of the SMI. Each manufacturer is given a section of the tree, after which it is up to the manufacturer to define it. This allows the definition of device specific MIBs to be delegated to an autonomous group. Since all information defined by the autonomous group, would be within the delegated area, e.g., *experimental.widget*, it is ensured to be uniquely identified.

3.3 STRUCTURE AND IDENTIFICATION OF MANAGEMENT INFORMATION

Managed objects shall be accessed via the Management Information Base (MIB). Objects in the MIB shall be defined using Abstract Syntax Notation One (ASN.1) and shall be in conformance with IAB STD 16 (RFC 1212).

Each object type shall have a name, a syntax, and an encoding. The name shall be represented uniquely as an OBJECT IDENTIFIER. An OBJECT IDENTIFIER shall be an administratively assigned name. The administrative policies discussed in IAB STD 16 (RFC 1155) shall be used for assigning names and identifiers.

The syntax for an object type shall define the abstract data structure corresponding to that object type. For example, the structure of a given object type might be an INTEGER or OCTET STRING. Although in general, any ASN.1 construct should be available for use in defining the syntax of an object type, this standard purposely restricts the ASN.1 constructs that may be used. These restrictions are made solely for the sake of simplicity.

The encoding of an object type determines how instances of that object type are represented when transmitted on the network. Encoding rules are discussed in Section 5.

3.3.1 Names

Names shall be used to identify managed objects. This subclause specifies names that shall be hierarchical in nature. The OBJECT IDENTIFIER concept shall be used to model this notion. OBJECT IDENTIFIERs are a means of identifying some object, regardless of the semantics associated with the object (e.g., a network object, a standards document, etc.). See IAB STD 16 (RFC 1155) for a more complete description of OBJECT IDENTIFIER.

An OBJECT IDENTIFIER is a sequence of integers that are used to traverse a global tree. The tree consists of a root connected to a number of labeled nodes via edges. Each node may, in turn, have branches of its own that are labeled. In this case, we may term the node a subtree. This process may continue to an arbitrary level of depth. A label is a pairing of a brief textual description and an integer. The format used to show this pairing shall be *text*(integer).

In the global tree the root node itself is unlabeled, but has three branches directly under it: one node is administered by the ISO, with label *iso*(1); another is administrated by the CCITT, with label *ccitt*(0); and the third is jointly administered by the ISO and the CCITT, *iso-ccitt*(2).

Under the *iso(1)* node, the ISO has designated one subtree for use by other (inter)national organizations, *org*(3). Of the children nodes present, two have been assigned to the U.S. National Institutes of Standards and Technology (NIST). One of these subtrees has been transferred by the NIST to the U.S. Department of Defense, *dod*(6). The Internet has been assigned a node under *dod* labeled *internet*(1). This node is administered by the Internet Activities Board (IAB). The IAB has assigned a node called *private*(4) which is used to define nodes that have the administration transferred to other organizations. Under *private* a node called *enterprises(1)* has been defined to hold nodes that are assigned to private

enterprises. NEMA has been assigned a node called *nema*(1206) under the *enterprises* node. The IAB has transferred authority for the *nema* node to NEMA.

In ASN.1 syntax, the definition of the *nema* node shall be:

```
nema OBJECT IDENTIFIER ::=
{ iso(1) org(3) dod(6) internet(1) private(4) enterprises(1) 1206 }
```

Four nodes have been assigned under the *nema* node. The definitions shall be:

a. nemaMgmt OBJECT IDENTIFIER ::= { nema 1 }
b. nemaExperimental OBJECT IDENTIFIER ::= { nema 2 }
c. nemaPrivate OBJECT IDENTIFIER ::= { nema 3 }
d. transportation OBJECT IDENTIFIER ::= { nema 4 }

Therefore the full object identifier for *nemaMgmt* would be:
iso.org.dod.internet.private.enterprises.nema.1 or in numeric values only 1.3.6.1.4.1.1206.1. The periods are used to separate the text(integer) for each identifier to unambiguously show the hierarchy.

3.3.1.1 nemaMgmt

The *nemaMgmt(1)* subtree is used to identify objects that are defined in NEMA-approved documents.

3.3.1.2 nemaExperimental

The *nemaExperimental(2)* subtree is used to identify objects used in NEMA experiments. New MIBs, prior to being assigned within the TMIB-II, are put here. In the Internet community, multiple manufacturers must utilize a MIB within the experimental area before it can move to a permanent location. As a part of the assignment process, NEMA will make requirements as to how that subtree is used.

For example, NEMA experiments might include an initial proposed MIB for a loop-detector or any other previously un-MIBed device. An experimenter might receive number 17, and would have available the OBJECT IDENTIFIER { nemaExperimental 17 } or 1.3.6.1.4.1.1206.2.17.

3.3.1.3 nemaPrivate

The *nemaPrivate(3)* subtree is used to identify objects defined unilaterally. Administration of the *nemaPrivate(3)* subtree is by NEMA. NEMA will assign nodes to enterprises for the purpose of defining enterprise specific MIBs.

Upon receiving a node the enterprise may, for example, define new MIB objects under this node. In addition, it is strongly recommended that the enterprise will also register its transportation devices under this subtree, in order to provide an unambiguous identification mechanism for use in management protocols. For example, if the "ABC, Inc."-enterprise produced transportation devices, then they could request a node under the *nemaPrivate* subtree from NEMA. Such a node might be numbered:

1.3.6.1.4.1.1206.3.42

The “ABC, Inc.” enterprise might then register their "Widget Controller" under the name of 1.3.6.1.4.1.1206.3.42.1 ensuring a unique identification. Thereafter, each enterprise is responsible for ensuring unique identification of information objects within their subtree. NEMA delegates the role of assigning numbers under each *nemaPrivate* node to those to which they are assigned, except of course for the initial enterprises number.

3.3.1.4 Transportation

All objects within the Transportation MIB (TMIB) shall be put here. The TMIB-II subtrees are defined in Section 4. This SMI creates the *transportation* node of the *nema* subtree, as shown below.

transportation OBJECT IDENTIFIER ::= { nema 4 }

The *transportation*(4) subtree is used by the NEMA Transportation Management Systems and Associated Control Devices Section to create subtrees for different classes of transportation equipment. The objects defined under each of these subtrees are standard objects specific to each class of transportation device. Please refer to Section 4 for more information on the TMIB-II.

3.3.2 Syntax

Syntax is used to define the structure corresponding to object types. IAB STD 16 (RFC 1155), ASN.1 constructs shall be used to define this structure, although the full generality of ASN.1 as discussed in IAB STD 16 (RFC 1155) is not permitted.

3.3.2.1 Primitive Types

Only the ASN.1 primitive types INTEGER, OCTET STRING, OBJECT IDENTIFIER, and NULL shall be permitted. These are sometimes referred to as non-aggregate types.

3.3.2.2 Guidelines for Enumerated INTEGERs

If an object type is listed as an enumerated INTEGER, then a named number having the value zero shall not be present in the list of enumeration. Use of this value is prohibited.

3.3.2.3 Constructor Types

The ASN.1 constructor type SEQUENCE shall be permitted only to generate either lists or tables. Only the short or long definite length format shall be permitted per IAB STD 16 (RFC 1155).

For lists, the syntax shall take the form:

SEQUENCE { <*type1*>, ..., <*typeN*> }

where each <*type*> shall resolve to one of the ASN.1 primitive types listed in 3.3.2.1. The DEFAULT and OPTIONAL clauses shall not appear in the SEQUENCE definition.

For tables, the syntax shall take the form:

SEQUENCE OF <*entry*>

where <entry> shall resolve to a list constructor. Lists and tables are referred to as aggregate types.

3.3.3 Managed Objects

Although it is not the purpose of this section to define objects in the MIB, IAB STD 16 (RFC 1212) does specify a format to be used by documents that define objects. All objects shall be defined using the OBJECT-TYPE macro described in IAB STD 16 (RFC 1212).

The OBJECT-TYPE macro definition shall contain at least the following five fields:

a. Object Name
b. Syntax
c. Access
d. Status
e. Description

3.3.3.1 Object Name

This field shall consist of a textual name, termed the OBJECT DESCRIPTOR, for the object type being defined. It shall precede the invocation of the OBJECT-TYPE macro. This Object Name takes on the value of the specified OBJECT IDENTIFIER.

3.3.3.2 Syntax

This must resolve to an instance of the ASN.1 type *ObjectSyntax* as described in IAB STD 16 (RFC 1212).

3.3.3.3 Access

Per IAB STD 16 (RFC 1212) shall be one of:

a. read-only
b. read-write
c. write-only
d. not-accessible

3.3.3.4 Status

The object status shall indicate its current status.

Status shall be one of:

a. mandatory
b. optional
c. obsolete
d. deprecated

A status of mandatory shall mean that any agent that implements this MIB module must implement this object. A status of optional shall mean that this object may be optionally implemented in an agent. A status of obsolete shall mean that this object is no longer used and agents are no longer required to implement it. A status of deprecated shall mean that the object is being discouraged and in future releases of the standard may be marked obsolete.

3.3.3.5 Description

Description is an ASCII-NVT string, which defines the semantics of the object type. The object description shall provide an unambiguous definition of what the object does. The description field shall be present in all OBJECT_TYPE definitions.

3.3.4 Extensions to the MIB

Every NEMA-Standard MIB document obsoletes all previous such documents. New versions shall:

a. declare old object types deprecated, but not delete their names;
b. declare previously deprecated object types obsolete (if necessary), but not delete their names;
c. augment the definition of an object type corresponding to a list by appending non-aggregate object types to the object types in the list;
d. define entirely new object types.

New versions shall not change the semantics of any previously defined object without changing the name of that object.

3.3.5 Groups

A group is a basic unit of conformance and is used to specify a collection of related managed objects. The group designation applied to a set of objects provides a systematic means for determining which objects are required to support a function. If a device has multiple functions, a group will be defined for each function. Group definitions will be found in the NTCIP Object Definition Standard documents. The Object Definition Standard may define a group with objects that are not in lexicographic order and only apply to devices of that type.

The related managed objects of a group may include mandatory and/or optional objects. Mandatory objects within a group shall be implemented. Optional objects shall be implemented only if a defined function of the device requires that particular object.

For example, assume a device implements an asynchronous RS-232 interface. It must implement all the mandatory objects in the asynchronous group of the RS-232 MIB. It would not have to implement the synchronous group of objects unless it also provided a synchronous interface.

Assume also that the asynchronous group has a *CRC error counter* object that is optional. The *CRC error counter* object would not have to be implemented unless the device used CRC checking on the asynchronous interface.

Section 4
TRANSPORTATION MANAGEMENT INFORMATION BASE VERSION 2

The Transportation MIB version 2 (TMIB-II) provides a framework for organizing and identifying information in transportation specific equipment. The TMIB-II is located under the *nema* node of the global object tree. This MIB is controlled by NEMA Transportation Management Systems and Associated Control Devices Section (3-TS). NEMA 3-TS may designate subnodes of the TMIB-II to be administered by other groups or organizations.

4.1 TRANSPORTATION TYPE DEFINITIONS

The TMIB-II defines several ASN.1 data types to help clarify information about various objects that can be expected to reside in transportation devices.

4.1.1 Byte and UByte Types

The Byte and UByte types shall be used to define single octet size integers, where Byte is a signed integer that ranges from -128 to +127, and UByte is an unsigned integer that ranges from 0 to 255.

4.1.2 Short and UShort Types

The Short and UShort types shall be used to define double octet size integers, where Short is a signed integer that ranges from -32768 to +32767, and UShort is an unsigned integer that ranges from 0 to 65535.

4.1.3 Long and ULong Types

The Long and ULong types shall be used to define quad octet size integers, where Long is a signed integer that ranges from -2147483648 to +2147483647, and ULong is an unsigned integer that ranges from 0 to 4294967295.

4.1.4 ConfigEntryStatus Type

The *EntryStatus* Type, as defined in TMIB(-I), has been replaced with the *ConfigEntryStatus* type in TMIB-II. The ConfigEntryStatus type shall be used to manage the dynObjDef(Table) that allow new rows to be created by management applications running remotely. For each row in the dynObjDefTable there shall be a columnar object that is defined with a SYNTAX of ConfigEntryStatus (e.g., see dynamic object configuration table in 4.2). ConfigEntryStatus shall be an enumerated INTEGER that can have one of three values: *valid*, *underCreation* and *invalid*.

Other objects in the row shall have operations limited by the current value of the EntryStatus object in the row. The meaning of the values is as follows:

If the status object is *invalid* then the information in the corresponding rows of the dynObjDef(Table) with the same index dynObjNumber shall be considered undefined. Setting the status object to invalid has the effect of invalidating the corresponding rows. It is implementation specific whether the agent clears the values contained in an invalidated row, de-allocates the memory associated with invalidated rows, or simply leaves the last values within the row. When in the *invalid* state, the agent shall reject any request to go to the *valid* state.

The *underCreation* state shall indicate that the memory for the corresponding rows of the dynObjDef(Table) with the same index *dynObjNumber* is allocated, but may contain some invalid data.

When in this state, the management application is allowed to modify the values of the objects contained in the associated rows of the table. Once this operation is completed, the management station may set the state to valid; alternatively, the management station may cancel the operation by setting the state to invalid. If the agent determines that an entry has been in the *underCreation* state for an abnormally long time, it may set this object to *invalid*.

The valid state shall indicate that the corresponding rows of the dynObjDef(Table) with the same index *dynObjNumber* contain information that is believed to be valid.

The following table indicates the actions that shall take place upon receipt of a set request to change the state of the corresponding rows of the dynObjDef(Table) with the same index *dynObjNumber.* The value of each cell in the table shows the result of receiving the indicated set request (column headings) when the device is in the indicated state (row headings).

Table 4–1
STATE TRANSITION TABLE

		COMMANDED STATE		
		invalid	*underCreation*	*valid*
CURRENT STATE	*invalid*	invalid (1)	underCreation (1)	invalid (3)
	underCreation	invalid (2)	underCreation (3)	valid (4) or underCreation (5)
	valid	invalid (2)	valid (3)	valid (1)

Notes:
(1) no action takes place and response indicates noError.
(2) all entries for the associated ConfigEntryStatus object are deleted or de-referenced and response indicates noError.
(3) no action takes place but response indicates badValue.
(4) if consistency check succeeds then state changes to *valid* and response indicates noError.
(5) if consistency check fails then state remains *underCreation* and response indicates genErr.

Upon receipt of set request for the *valid* state when in the *underCreation* state, the agent shall perform a consistency check on the data contained in the associated row of the dynObjDef(Table) with the same index *dynObjNumber* of the table. If the consistency check is successful, the state shall change to valid, otherwise, the state remains in the *underCreation* state and the device shall return a genErr.

4.1.5 OwnerString Type

The OwnerString type shall be used to allow the owner of a resource to store information about themselves in the object. This information must be taken from the NVT ASCII character set. These objects shall be used to store information about who has control of the associated resources. The object shall be capable of storing 127 NVT ASCII characters.

4.2 STRUCTURE OF TRANSPORTATION MANAGEMENT INFORMATION

The TMIB-II defines very few objects. It defines the tree structure upon which other Standards place objects. Annex B contains the complete ASN.1 description of the TMIB-II. In ASN.1 notation the *transportation* node is defined as:

```
transportation  OBJECT IDENTIFIER ::= { nema 4 }
```

The TMIB-II shall have three nodes located under the *transportation* node. The *protocols* node shall be the beginning of a subtree that holds information about various communications protocols. The *devices*

node shall be the beginning of a subtree that holds information about various standard transportation device objects. The *tcip* node shall be the beginning of a subtree that holds information about various standard transit objects. The ASN.1 notation for these three nodes shall be defined as:

```
protocols   OBJECT IDENTIFIER ::= { transportation 1 }
devices     OBJECT IDENTIFIER ::= { transportation 2 }
tcip        OBJECT IDENTIFIER ::= { transportation 3 }
```

4.2.1 Protocols Subtree

The *protocols* subtree contains nodes that have information about specific implementations of STMP on different protocols. The *protocols* tree shall have three nodes in it which are defined as:

a. layers—OBJECT IDENTIFIER ::= { protocols 1 }
b. profiles—OBJECT IDENTIFIER ::= { protocols 2 }
c. dynObjMgmt—OBJECT IDENTIFIER ::= { protocols 3 }

4.2.1.1 Dynamic Object Management (dynObjMgmt)

The *dynObjMgmt* subtree is used by the STMP to reduce the bandwidth requirements of communicating sets of objects between a management station and agents. The *dynObjMgmt* tree contains three nodes: *dynObjDef, dynObjData, and dynObjConfigTable*. All of the nodes under dynObjData have been deprecated.

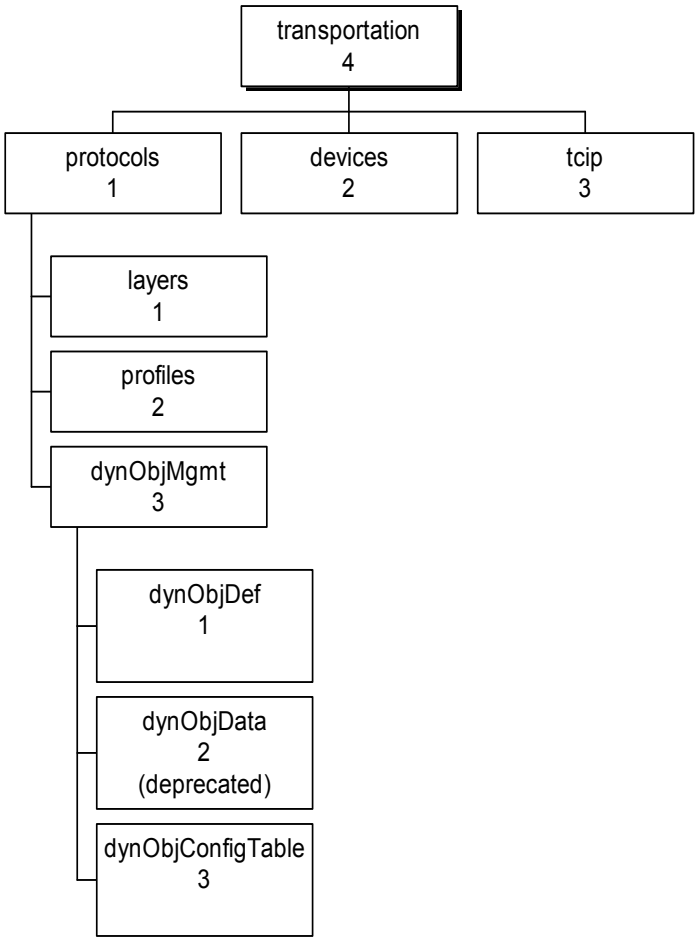

Figure 4-1
TMIB TREE STRUCTURE

4.2.1.1.1 Dynamic Object Definition (dynObjDef)

The *dynObjDef* node is a table of OBJECT IDENTIFIERs that make up the 13 dynamic objects supported by STMP. Each row (entry) of the table shall identify a specific object within a dynamic object, and its position(order) in the dynamic object. The dynamic object definition table has rows that contain the following objects:

dynObjNumber	dynObjIndex	dynObjVariable

The INDEX for a particular row in the *dynObjDef* table is defined as a concatenation of both the *dynObjNumbe*r and *dynObjIndex* elements.

The *dynObjNumber* shall be of type **INTEGER (1...13).** It identifies which of the 13 dynamic objects this row of the table is associated with. This field shall consist of an integer between 1 and 13. Values other than these will cause an error to be returned.

The *dynObjIndex* shall be of type **INTEGER (1..255)**. It identifies one of the possible 255 object identifiers that is associated with each of the dynamic objects. The sequence of indexes used to fill out the table is not critical. However, once completed, the indexes must be in order with no gaps.

The *dynObjVariable* shall be of type **OBJECT IDENTIFIER**. It must point to an object supported by the device. For some columnar objects, only the prefix portion of an object identifier can be validated. The suffix or index portion of an object identifier need not be instantiated or existing at the time a *dynObjVariable* is defined.

The *dynObjOwner* and *dynObjStatus* objects (from TMIB version 1) have been deprecated and replaced with similar objects contained in the *dynObjConfigTable* as defined in 4.2.1.1.3. This has the effect of associating a single owner and status with each dynamic object rather than with each indexed item within a dynamic object.

4.2.1.1.2 Dynamic Object Data (dynObjData)

The objects under this node have been deprecated.

4.2.1.1.3 Dynamic Object Configuration Table

The *dynObjConfigTable* is a table indicating the owner and status of each dynamic object. It is an integral part of the *dynObjDef*(Table) and uses the same *dynObjNumber* as its index. The *dynObjConfigTable* shall have conceptual rows that contain the following objects:

dynObjNumber	dynObjConfigOwner	dynObjConfigStatus

The INDEX for a particular row in the *dynObjDef* table is defined by *dynObjNumbe*r. As stated previously, the *dynObjNumber* shall be of type **INTEGER (1..13).** It identifies with which of the 13 dynamic objects this row of the table is associated.

The *dynObjConfigOwner* shall be of type **OwnerString.** Its only purpose is to indicate the identity of the management station that defined the dynamic object. The default value shall be a zero (0) length string.

The *dynObjConfigStatus* shall be of type **ConfigEntryStatus**. It shall be used as described in 4.1.4. The *invalid* state shall indicate the associated dynamic object is not defined. The *underCreation* state shall indicate the management station is currently configuring the dynamic object (i.e., setting the values of the dynObjConfigOwner and various dynObjVariable objects associated with the subject dynamic object). The *valid* state shall indicate that the management station has completed the setup, and that the agent has successfully completed a validity check on the data.

When validating the data entered for a dynamic object, an agent shall perform the following consistency checks:

1. *dynObjIndex* 1 *has* an *dynObjVariable* entry.
2. All *dynObjVariable*s have sequential *dynObjIndex*s
3. There are no skipped *dynObjIndex*s.

Failure to pass these consistency checks shall prevent the state from changing to valid.

A management station must take into account the variable binding list processing nature of SNMP. In SNMP, all objects contained in a single set-request data packet appear to be set to their new values simultaneously. Therefore, a management station must not combine a state change request with a request to set an instance value associated with that state change. If such an operation is attempted, the operation may not be correctly processed.

4.2.1.2 Profiles Subtree

This subtree shall be used for standard objects that affect or describe various communications profiles that may be defined for use with STMF. The definition of the objects in this subtree is left to other

documents. Each profile must be assigned a number under this node by NEMA and all objects defined under these nodes must be reviewed by NEMA before they are used.

4.2.1.3 Layers Subtree

This subtree shall be used for standard objects that affect or describe various communication protocol layers that may be defined for use with STMF. The definition of the objects in this subtree is left to other documents. Each layer must be assigned a number under this node by NEMA and all objects defined under these nodes must be reviewed by NEMA before they are used.

4.2.2 Devices Subtree

This subtree shall be used for standard objects that are defined for specific types of transportation equipment. Each type of equipment (i.e. actuated traffic controller, variable message sign, camera control, etc.) will have its own subtree in the devices tree. The definition of the objects under this tree is left to other documents. Each device type must be assigned a number under this node by NEMA and all objects defined under these nodes must be reviewed by NEMA before they are used.

4.2.3 TCIP Subtree

This subtree shall be used for standard objects that are defined for specific types of transit data. The definition of the objects under this tree is left to other documents. The registration authority for this node is the Institute of Transportation Engineers.

Section 5
PROTOCOL DEFINITIONS

Applications conforming to this standard shall support the Simple Network Management Protocol (SNMP) and optionally support the Simple Transport Management Protocol (STMP). The conformance level of a device shall determine whether it supports only SNMP or both STMP and SNMP.

The Protocol Data Unit (PDU) shall consist of a PDU Header field and when required, a PDU Information field. The high order bit of the PDU Header field shall designate the PDU format as follows:

BIT	CONTENTS	Description
7	*PDU Format*	
	0	Format dependent upon seven lower order bits
	1	Standard STMP format

If the PDU Format bit is 0, then the seven low order bits of the STMF PDU Header field are defined as follows:

BIT	CONTENTS	Description
6-0	*Detailed PDU Format*	
	000 0000 - 010 1111	Reserved
	011 0000	Standard SNMP Format
	011 0001 - 111 1111	Reserved

If the PDU Header field equals 0x30 (Standard SNMP format) then the PDU Header field shall be considered the first byte of the SNMP message.

5.1 SIMPLE TRANSPORTATION MANAGEMENT PROTOCOL

The STMP is modeled after the SNMP, however, several features are modified so that the STMP can provide more efficient operation.

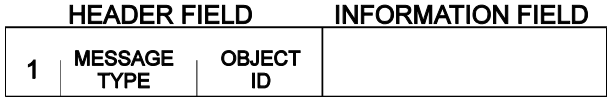

Figure 5-1
STMP PDU FIELDS

5.1.1 Protocol Data Unit

For STMP the PDU Header field shall conform to the following format:

BIT	CONTENTS	Description
7	*PDU Format*	
	1	Standard STMP format
6-4	*Message Type*	
	000	GET request is contained in the packet.
	001	SET request is contained in the packet.
	010	SET request no reply is contained in the packet.
	011	GET NEXT request is contained in the packet.
	100	GET response is contained in the packet (positive ACK).
	101	SET response is contained in the packet (positive ACK).
	110	ERROR response is contained in the packet.
	111	TRAP response is contained in the packet.
3-0	*Object ID*	
	0000	reserved
	0001-1101	ID of STMP "dynamic object"
	1110	reserved
	1111	reserved

5.1.1.1 PDU Format Bit

The value of the PDU Format bit shall be used to determine if the protocol is STMP.

5.1.1.2 Message Type Bit Field

The message type field shall be used to determine what type of STMP message is contained in the packet. There are eight types of messages: *get request*, *set request*, *set request no reply*, *get next request*, *get response*, *set response*, *error response*, and *trap response.*

A *get request* shall be sent from a manager to an agent to request the values of specified objects.

A *set request* shall be sent from a manager to an agent to request that specified objects be set to specified values.

A *set request no reply* shall be sent from a manager to an agent to request that specified objects be set to specified values.

A *get next* request shall be sent from a manager to an agent to request the value of the next object instance in lexicographic order.

A *get response* shall be used to send specified object data from an agent to a manager. This response contains the object values that correspond to the prior *get request* or *get next* request from the manager. *Get response* messages shall not be sent unless the agent has received a *get request* packet from the manager.

A *set response* shall be sent from an agent to the manager to indicate that the *set request* for the specified objects was completed without error.

An *error response* shall be sent from the agent to the manager in response to a *set request*, *get request*, or *get next* request that contained an error. The error information field is described in 5.1.1.5.

A *trap response* shall be used to send specified object data from an agent to a manager. The trap is generated due to some event occurring within the agent, and is not a response to any manager request.

The STMP *trap response* message is under development and will be completed under a future revision of this Standard.

5.1.1.3 Object ID Field

The object ID field shall be used to determine how the following information in the packet is encoded. STMP supports Octet Encoding Rules (OER) and provides compression to reduce the bandwidth requirements of the protocol.

If the value of the OBJECT ID field is in the range 1...13 (0x1...0xD) then the following information in the packet is encoded using Octet Encoding Rules (OER). Only one of thirteen dynamic objects can be accessed in this mode. The value of the OBJECT ID field selects one of the thirteen objects.

The *dynObjData* objects can be defined to point to any objects supported by the agent. This is accomplished using the *dynObjDef* table to specify the OBJECT IDENTIFIERs that will make up the dynamic object. Thirteen sets of objects can be defined and operated on with minimal overhead. The *dynObjDef* table is discussed in Section 4 (*Transportation Management Information Base*). In addition, the OER eliminates other redundant information to further reduce the amount of data that is transmitted to operate on objects. The OER is discussed in 5.1.2.2 below.

If the value of the OBJECT ID field is 0 (0x0), 14 (0xE), or 15 (0xF) then the packet shall be ignored.

All STMP messages shall conform to the Operations and Semantics specified in 5.1.2.

5.1.1.4 PDU Information Field

The form of the PDU Information field for a *set request, set request no reply,* and *get response* shall be an ASN.1 SEQUENCE of the objects referenced by the associated *dynObjVariables*, in order. For example, if the dynamic object is defined to have two indexed variables, the first referencing an object of SYNTAX INTEGER (0..255) and the other referencing an object of SYNTAX OCTET STRING (SIZE (0..127)), the STMP PDU Information field would be defined as:

```
SEQUENCE {
    value1              INTEGER (0..255),
    value2              OCTET STRING (SIZE (0..127))
    }
```

The PDU Information field of an STMP *error response* shall only contain the error status and error index:

```
    STMPErrorResponse ::=
        SEQUENCE {
                error-status
                  ErrorStatus,

                error-index
                  ErrorIndex
        }
```

Set response, get request, and get-next request messages shall not include a PDU Information field.

The form of the PDU Information field for an STMP *trap response* is still under development and will be completed under a future revision of this Standard.

5.1.1.5 Error Responses

STMP shall not return the information that was included in the request packet when an error is returned. STMP shall return an error number and an index number only. The manager should retain the last request transmitted to an agent in order to determine what part of the request caused the error.

The error number shall be encoded as a single octet. A type and length are not encoded. The possible errors shall be:

a. *tooBig(1):* this error is returned if the PDU was larger than expected. The index number shall be set to zero.

b. *noSuchName(2):* the object identifier indicated by the index number is not supported by the agent.

c. *badValue(3):* this error can only occur during a set operation. The object indicated by the index number value shall be the first that is not valid (out of range).

d. *readOnly(4):* this error can only occur during a set operation. The index number indicates which object could not be written.

e. *genErr(5):* this error indicates that some other error has occurred that does not conform to one of the specified errors above. It is application specific and requires referencing the agents documentation to determine what the error may be.

The index number of the error response will indicate the DynObjIndex of the entry in the Dynamic Object Table that resulted in the error. In other words, the index number of the error response will indicate the field within the dynamic object that resulted in an error. The index numbers shall be encoded at INTEGER (0..255).

The index number shall be encoded as a single octet for index number values ranging from 0 through 127 (0x7F). For index values larger than 127, the first octet shall have the most significant bit set to 1 and the remaining seven bits set to indicate how many octets follow for the index number. For example for an index value of 192 (0xC0) the index number would be encoded as two octets with the values 0x81 and 0xC0. This encoding is identical to the length encoding used by BER. Since most requests contain less than 127 fields, it should be rare to encounter an index number that exceeds a single octet.

5.1.2 Operations and Semantics

The operations of the STMP are identical to SNMP with the following differences:

a. When a STMP *get request, get next request,* or *set request* message is rejected by an agent, the STMP *error response* PDU Information field shall only consist of an error number and index number as specified in 5.1.1.5.

b. An agent receiving a *set request noReply* shall not respond under any condition.

5.1.2.1 Basic Encoding Rules (BER)

[Editor's note: This clause was deleted by Amendment 1 but the clause title is retained to maintain numbering.]

5.1.2.2 Octet Encoding Rules (OER)

Within the STMF, OER only applies to STMP. The STMP utilizes a more compact encoding of data to transmit *dynamic objects.* This compact encoding is important to improve the bandwidth utilization for sequences of objects that are frequently requested.

Normal BER uses a three *tuple* to encode data for transmission. The first element of the tuple is an octet that specifies what type of data follows. The second element of the tuple is an octet(s) that specifies how many octets the data occupies. The third and final element of the tuple is the actual data being transmitted. This provides a very flexible method of encoding information for transmission. This encoding is sometimes referred to as *TLV* encoding which stands for type length value.

OER takes advantage of predefined data types to eliminate certain elements of the BER encoding. Since dynamic objects must be defined prior to their use, both the manager and the agent know what each type is and for types of fixed length (e.g.Integer32) what the length of the data is. This allows the OER encoding to eliminate the type element and on fixed length data objects the length element can be eliminated as well. When a syntax definition includes some type of limitation on the range of values, this is referred to as a visible constraint.

Some rules can be applied to determine the proper encoding based on BER.

a. All TYPE fields shall be removed in the PDU Information field.

b. All objects that resolve to the Primitive type NULL shall not be encoded.

c. All objects that resolve to a Primitive type of variable sized OCTET STRING or OBJECT IDENTIFIER shall be encoded as a variable length field.

d. If the type is fixed length then the length information shall be removed.

e. If the type is variable length, except for SEQUENCE and SEQUENCE OF, then the length information shall be encoded identically to the BER.

f. All objects that resolve to the Primitive type INTEGER and contain named values shall be constrained to the range (0..127). This provides backward compatibility with those standards already approved. It is expected that a mechanism will be provided to support larger ranges.

g. All objects that resolved to the Primitive Type INTEGER, not covered above, shall be encoded according to the rules defined in 5.1.2.3.

h. All objects that resolve to a Primitive Type of a fixed sized OCTET STRING (e.g., OCTET STRING (SIZE (5)) shall be encoded as a fixed length field.

5.1.2.3 Encoding of an Integer Value

The encoding of an integer type is dependent upon the range of valid values for the type, including any possible future extensions based on the presence of an extension marker.

5.1.2.3.1 Rules When Negative Values Not Allowed

If an OER-visible constraint specification does not allow any negative values, the following rules apply.

If the constraint specification limits the possible values to less than or equal to 255, the length octets shall be omitted and the contents octet shall be encoded as a one-byte unsigned integer (see 5.1.2.4.2.1).

If the constraint specification allows values greater than 255, but limits the values to less than or equal to 65535, the length octets shall be omitted and the contents octet shall be encoded as a two-byte unsigned integer (see 5.1.2.4.2.2).

If the constraint specification allows values greater than 65535, but limits the values to less than or equal to 4294967295, the length octets shall be omitted and the contents octet shall be encoded as a four-byte unsigned integer (see 5.1.2.4.2.3).

If the constraint specification allows values greater than 4294967295, the length octets shall be present and the contents octet shall be encoded as an unrestricted unsigned integer (see 5.1.2.4.2.4).

5.1.2.3.2 Rules When Negative Values Allowed

If an OER-visible constraint specification allows negative values, the following rules apply. Note that, by definition, these rules apply when there are no OER-visible constraints.

If the constraint specification limits the maximum possible value to less than or equal to 127 and limits the minimum possible value to greater than or equal to -128, the length octets shall be omitted and the contents octet shall be encoded as a one-byte signed integer (see 5.1.2.4.1.1).

If the conditions of the preceding paragraph are not met **and** the constraint specification limits the maximum value to less than or equal to 32767 **and** the constraint specification limits the minimum value to greater than -32768, the length octets shall be omitted and the contents octet shall be encoded as a two-byte signed integer (see 5.1.2.4.1.2).

If the conditions of the preceding paragraph are not met **and** the constraint specification limits the maximum value to less than or equal to 2147483647 **and** the constraint specification limits the minimum value to greater than -2147483648, the length octets shall be omitted and the contents octet shall be encoded as a four-byte signed integer (see 5.1.2.4.2.3).

If the conditions of the preceding paragraph are not met, the length octets shall be present and the contents octet shall be encoded as an unrestricted signed integer (see 5.1.2.4.2.4).

5.1.2.3.3 Examples

INTEGER would be encoded as an unrestricted signed integer.

Counter would be encoded as a four-byte unsigned integer.

TimeTicks would be encoded as a four-byte unsigned integer.

Gauge would be encoded as a four-byte unsigned integer.

INTEGER (0..MAX) would be encoded as an unrestricted unsigned integer.

INTEGER (0..255) would be encoded as a one-byte unsigned integer.

Counter (0..255) would be encoded as a one-byte unsigned integer.

INTEGER (0..2000) would be encoded as a two-byte unsigned integer.

INTEGER (1999..2000) would be encoded as a two-byte unsigned integer.

Gauge (1200..1250) would be encoded as a two-byte unsigned integer.

INTEGER (0..255, …) would be encoded as an unrestricted signed integer due to the extension marker.

INTEGER (-128..127) would be encoded as a one-byte signed integer

INTEGER (-1000..1000) would be encoded as a two-byte signed integer.

INTEGER { a(1), b(2) } would be encoded as a one-byte unsigned integer. Although the "NamedNumberList" is not significant according to ASN.1 rules, Section 5.1.2.2, Item f. is applied to constrain the range to INTEGER (0..127).

5.1.2.4 Encoding of Whole Numbers

5.1.2.4.1 Encoding of a Signed Integer

5.1.2.4.1.1 Encoding of a One-Byte Signed Integer

The value shall be encoded as an eight-bit 2s complement integer.

5.1.2.4.1.2 Encoding of a Two-Byte Signed Integer

The value shall be encoded as an sixteen-bit 2s complement integer.

5.1.2.4.1.3 Encoding of a Four-Byte Signed Integer

The value shall be encoded as an thirty-two bit 2s complement integer.

5.1.2.4.1.4 Encoding of an Unrestricted Signed Integer

The value shall be encoded according to clauses 8.3.2 and 8.3.3 of BER

5.1.2.4.2 Encoding of an Unsigned Integer

5.1.2.4.2.1 Encoding of a One-Byte Unsigned Integer

The value shall be encoded as an eight-bit binary-integer.

5.1.2.4.2.2 Encoding of a two-byte unsigned integer

The value shall be encoded as a 16-bit binary-integer.

5.1.2.4.2.3 Encoding of a four-byte unsigned integer

The value shall be encoded as a 32-bit binary-integer.

Encoding of an unrestricted unsigned integer

The value shall be encoded as a minimum octet non-negative-binary-integer.

5.2 SIMPLE NETWORK MANAGEMENT PROTOCOL

SNMP shall conform to IAB STD 15 (RFC 1157).

5.3 TRANSPORT MAPPINGS

STMP protocol can operate over a connectionless protocol. However, like SNMP, STMP has no checksum or CRC and relies on a lower layer protocols to detect and discard corrupted packets.

Section 6
CONFORMANCE LEVEL

6.1 GENERAL CONFORMANCE

Devices shall conform to either level 1 or 2 as described in 6.1.1 and 6.1.2.

Minimum and maximum ranges different from the values of the SYNTAX field may be enforced by an application running on a device.

A device that enforces range limits within the bounds specified by the values of the SYNTAX field shall not be categorized as being non-compliant with NTCIP.

6.1.1 Conformance Level 1

In order to be at conformance level 1 the following shall be supported:

a. SNMP version 1: All supported objects including Internet standard objects are accessible. This level shall support the Internet system (group) objects.

6.1.2 Conformance Level 2

In order to be at conformance level 2 the following shall be supported:

a. Conformance level 1.

b. STMP: All STMP operations.

c. TMIB: All mandatory objects and groups defined within the TMIB.

d. Dynamic objects including: definition creation; definition deletion and object get and set data functions.

6.2 MANAGEMENT APPLICATION CONFORMANCE

Management Applications shall conform to either Conformance level 1 or 2 as described in 6.1.1 and 6.1.2. In addition, management applications may provisionally support the ability to read ASCII text ASN.1 MIB modules such that the management applications may update their MIB views when appropriate.

< This page has intentionally been left blank. >

Annex A
SMI FOR NEMA

(Normative)


```
NEMA_SMI DEFINITIONS ::= BEGIN

-- Initial definitions for NEMA Structure of Management Information.
-- This contains only structure information, no managed objects are defined.

IMPORTS
     enterprises
          FROM RFC1155-SMI


EXPORTS -- EVERYTHING
-- Major groups of the NEMA_SMI
     nema, nema.nemaMgmt, nema.nemaExperimental, nema.nemaPrivate, nema.transportation

-- NEMA has received ID 1206 from IANA
-- NEMA starts at { iso org dod internet private enterprises 1206 } in the global naming tree.

     nema                OBJECT IDENTIFIER  ::=  { enterprises 1206 }

     nemaMgmt            OBJECT IDENTIFIER  ::=  { nema 1 }
     nemaExperimental    OBJECT IDENTIFIER  ::=  { nema 2 }
     nemaPrivate         OBJECT IDENTIFIER  ::=  { nema 3 }
     transportation      OBJECT IDENTIFIER  ::=  { nema 4 }

-- The nemaMgmt subtree is used for standard NEMA object definitions that span
-- different NEMA sections.
-- The nemaExperimental subtree is used for technical committees developing objects
-- that may become standard in the future.
-- The nemaPrivate subtree is used for unilaterally defined objects. One common use is
-- for manufacturer specific or customer specific MIB definitions.
-- The transportation subtree is used by the NEMA 3TS Section to define standard
-- objects specific for the transportation industry.

END -- NEMA_SMI Definitions
```


< This page has intentionally been left blank. >

Annex B
TRANSPORTATION MANAGEMENT INFORMATION BASE
Version 2

(Normative)

```
-- Copyright 1996 by National Electrical Manufacturers Association.


TMIB-II DEFINITIONS ::= BEGIN
-- Transportation Management Information Base and Framework - Version 2


IMPORTS
     ObjectName
          FROM RFC1155-SMI
     DisplayString
          FROM RFC1158-MIB
     OBJECT-TYPE
          FROM RFC1212
-- NEMA has received ID 1206 from IANA
     nema, transportation
          FROM NEMA_SMI

EXPORTS -- EVERYTHING
-- Major groups of the TMIB.
     protocols, dynObjMgmt, dynObjDef, dynObjData, profiles, layers, devices
     dynObjEntry, dynObjNumber, dynObjIndex, dynObjVariable, dynObjOwner,
     dynObjStatus

--This MIB module uses the extended OBJECT-TYPE macro as
-- defined in IAB STD 16 (RFC 1212);


---- ****************************************************************
----      Special Object Type Definitions
---- ****************************************************************


-- The following are subtypes of conventional types formed by
-- subtyping ASN types.


Byte ::=       INTEGER (-128..127)
UByte ::=      INTEGER (0..255)

Short ::=      INTEGER (-32768..32767)
UShort ::=     INTEGER (0..65535)

Long ::=       INTEGER (-2147483648..2147483647)
ULong ::=      INTEGER (0..4294967295)
```

```
ConfigEntryStatus  ::= INTEGER
    { valid (1),
      underCreation (2),
      invalid (3)
    }
    -- See Section 4.1.4 for the complete definition of this Type.


OwnerString ::= DisplayString (SIZE (0..127))
-- This data type is used to model an administratively assigned name of the owner of
-- a resource. This information is taken from the NVT ASCII character set.
-- It is suggested that this name contain one or more of the following:
-- management station name, manager's name, location or phone number.
--
-- SNMP access control is articulated entirely in terms of the contents of MIB views;
-- access to a particular SNMP object instance depends only upon its presence or
-- absence in a particular MIB view and never upon its value or the value of related object
-- instances. Thus, objects of this type afford resolution of resource contention only among
-- cooperating managers; they realize no access control function with respect to
-- uncooperative parties.
--
-- By convention, objects with this syntax are declared as having
--
--    SIZE ( 0..127 )
```



```
-- ************************************************************
--          GROUPS WITHIN THE TMIB
-- ************************************************************
-- ************************************************************
--          Protocol Group
-- ************************************************************
-- The protocols group contains information about specific implementations
-- of different protocols.
--
protocols               OBJECT IDENTIFIER ::= { transportation 1 }

layers                  OBJECT IDENTIFIER ::= { protocols 1 }

profiles                OBJECT IDENTIFIER ::= { protocols 2 }

dynObjMgmt    OBJECT IDENTIFIER ::= { protocols 3 }

    dynObjData          OBJECT IDENTIFIER ::= { dynObjMgmt 2 }

--*******************************************************************
--          Device Group
--*******************************************************************
devices                 OBJECT IDENTIFIER ::= { transportation 2 }



--*******************************************************************
--          Dynamic Object Group
-- *******************************************************************


dynObjDef  OBJECT-TYPE
    SYNTAX  SEQUENCE OF DynObjEntry
    ACCESS  not-accessible
    STATUS  mandatory
    DESCRIPTION
        "A list of objects to be included in dynamic objects"
::= { dynObjMgmt 1 }

    dynObjEntry  OBJECT-TYPE
        SYNTAX  DynObjEntry
        ACCESS  not-accessible
        STATUS  mandatory
        DESCRIPTION
                "A list of OBJECT IDENTIFIERs that make up a dynamic object"
        INDEX  { dynObjNumber  dynObjIndex }
    ::= { dynObjDef 1 }

DynObjEntry ::= SEQUENCE {
    dynObjNumber      INTEGER (1..13),
    dynObjIndex       UByte,
    dynObjVariable    OBJECT IDENTIFIER,
    dynObjOwner       OwnerString,
```


dynObjStatus EntryStatus }

dynObjNumber OBJECT-TYPE
SYNTAX INTEGER (1..13)
ACCESS read-only
STATUS mandatory
DESCRIPTION
"The dynamic object number that this entry is to be associated with."
::= { dynObjEntry 1 }

dynObjIndex OBJECT-TYPE
SYNTAX INTEGER (1..255)
ACCESS read-only
STATUS mandatory
DESCRIPTION
"An index that uniquely identifies an entry in the dynamic object table. Each entry defines an object that is to be associated with a dynamic object number. The dynObjIndex determines the order in which objects are transmitted for the associated dynamic object. The lower dynObjIndex numbers are transmitted before larger numbers for entries within the same dynamic object."
::= { dynObjEntry 2 }

dynObjVariable OBJECT-TYPE
SYNTAX OBJECT IDENTIFIER
ACCESS read-write
STATUS mandatory
DESCRIPTION
"The object identifier of the particular variable to be included in the specified dynamic object number. Care must be taken when defining dynamic objects so that the maximum size of all the objects included in a dynamic object do not exceed the maximum packet size of the communications network.

When set to reference a columnar object, an agent may wish to only validate the prefix portion of the object identifier. The suffix or index portion of an object identifier need not be instantiated or exist at the time the time a dynObjVariable is defined.

This object may not be modified unless the associated dynObjStatus object is equal to underCreation."
::= { dynObjEntry 3 }

dynObjOwner OBJECT-TYPE
SYNTAX OwnerString
ACCESS read-write
STATUS deprecated
DESCRIPTION
"The entity that configured this entry and is therefore using the resources assigned to it.

This object may not be modified if the associated dynObjStatus object is equal to valid(1).

This object has been replaced with dynObjConfigOwner."
::= { dynObjEntry 4 }


```
dynObjStatus OBJECT-TYPE
    SYNTAX  EntryStatus
    ACCESS  read-write
    STATUS  deprecated
    DESCRIPTION
        "The status of this dynamic object definition entry. See description of
        EntryStatus above for restrictions on accesses.

        This object has been replaced with dynObjConfigStatus."
::= { dynObjEntry 5 }

--*****************************************************************************
--          Dynamic Object Data
--*****************************************************************************
-- ALL OBJECTS UNDER THIS NODE HAVE BEEN DEPRECATED
-- DUE TO THEIR LIMITED USEFULNESS

dynObj1  OBJECT-TYPE
    SYNTAX  OCTET STRING
    ACCESS  read-write
    STATUS   deprecated
    DESCRIPTION
        "The value of this object is determined by the dynObjDef entries with
        dynObjNumber equal to 1. Packed Encoding Rules are utilized to encode
        the objects for transmission. This object is intended for use with the Simple
        Transportation Management Protocol, and provides little advantage if used
        with SNMP.
        If no objects are defined for this dynamic object number, then an error of
        noSuchName shall be returned by the agent."
::= { dynObjData 1 }

dynObj2  OBJECT-TYPE
    SYNTAX  OCTET STRING
    ACCESS  read-write
    STATUS deprecated
    DESCRIPTION
        "The value of this object is determined by the dynObjDef entries with
        dynObjNumber equal to 2. Packed Encoding Rules are utilized to encode
        the objects for transmission. This object is intended for use with the Simple
        Transportation Management Protocol, and provides little advantage if used
        with SNMP.
        If no objects are defined for this dynamic object number, then an error of
        noSuchName shall be returned by the agent."
::= { dynObjData 2 }
```

dynObj3 OBJECT-TYPE
SYNTAX OCTET STRING
ACCESS read-write
STATUS deprecated
DESCRIPTION
"The value of this object is determined by the dynObjDef entries with dynObjNumber equal to 3. Packed Encoding Rules are utilized to encode the objects for transmission. This object is intended for use with the Simple Transportation Management Protocol, and provides little advantage if used with SNMP.
If no objects are defined for this dynamic object number, then an error of noSuchName shall be returned by the agent."
::= { dynObjData 3 }

dynObj4 OBJECT-TYPE
SYNTAX OCTET STRING
ACCESS read-write
STATUS deprecated
DESCRIPTION
"The value of this object is determined by the dynObjDef entries with dynObjNumber equal to 4. Packed Encoding Rules are utilized to encode the objects for transmission. This object is intended for use with the Simple Transportation Management Protocol, and provides little advantage if used with SNMP.
If no objects are defined for this dynamic object number, then an error of noSuchName shall be returned by the agent."
::= { dynObjData 4 }

dynObj5 OBJECT-TYPE
SYNTAX OCTET STRING
ACCESS read-write
STATUS deprecated
DESCRIPTION
"The value of this object is determined by the dynObjDef entries with dynObjNumber equal to 5. Packed Encoding Rules are utilized to encode the objects for transmission. This object is intended for use with the Simple Transportation Management Protocol, and provides little advantage if used with SNMP.
If no objects are defined for this dynamic object number, then an error of noSuchName shall be returned by the agent."
::= { dynObjData 5 }

dynObj6 OBJECT-TYPE
SYNTAX OCTET STRING
ACCESS read-write
STATUS deprecated
DESCRIPTION
"The value of this object is determined by the dynObjDef entries with dynObjNumber equal to 6. Packed Encoding Rules are utilized to encode the objects for transmission. This object is intended for use with the Simple Transportation Management Protocol, and provides little advantage if used with SNMP.
If no objects are defined for this dynamic object number, then an error of

noSuchName shall be returned by the agent."
::= { dynObjData 6 }

dynObj7 OBJECT-TYPE
SYNTAX OCTET STRING
ACCESS read-write
STATUS deprecated
DESCRIPTION
"The value of this object is determined by the dynObjDef entries with dynObjNumber equal to 7. Packed Encoding Rules are utilized to encode the objects for transmission. This object is intended for use with the Simple Transportation Management Protocol, and provides little advantage if used with SNMP.
If no objects are defined for this dynamic object number, then an error of noSuchName shall be returned by the agent."
::= { dynObjData 7 }

dynObj8 OBJECT-TYPE
SYNTAX OCTET STRING
ACCESS read-write
STATUS deprecated
DESCRIPTION
"The value of this object is determined by the dynObjDef entries with dynObjNumber equal to 8. Packed Encoding Rules are utilized to encode the objects for transmission. This object is intended for use with the Simple Transportation Management Protocol, and provides little advantage if used with SNMP.
If no objects are defined for this dynamic object number, then an error of noSuchName shall be returned by the agent."
::= { dynObjData 8 }

dynObj9 OBJECT-TYPE
SYNTAX OCTET STRING
ACCESS read-write
STATUS deprecated
DESCRIPTION
"The value of this object is determined by the dynObjDef entries with dynObjNumber equal to 9. Packed Encoding Rules are utilized to encode the objects for transmission. This object is intended for use with the Simple Transportation Management Protocol, and provides little advantage if used with SNMP.
If no objects are defined for this dynamic object number, then an error of noSuchName shall be returned by the agent."
::= { dynObjData 9 }

dynObj10 OBJECT-TYPE
SYNTAX OCTET STRING
ACCESS read-write
STATUS deprecated

DESCRIPTION
"The value of this object is determined by the dynObjDef entries with dynObjNumber equal to 10. Packed Encoding Rules are utilized to encode the objects for transmission. This object is intended for use with the Simple Transportation Management Protocol, and provides little advantage if used with SNMP.
If no objects are defined for this dynamic object number, then an error of noSuchName shall be returned by the agent."
::= { dynObjData 10 }

dynObj11 OBJECT-TYPE
SYNTAX OCTET STRING
ACCESS read-write
STATUS deprecated
DESCRIPTION
"The value of this object is determined by the dynObjDef entries with dynObjNumber equal to 11. Packed Encoding Rules are utilized to encode the objects for transmission. This object is intended for use with the Simple Transportation Management Protocol, and provides little advantage if used with SNMP.
If no objects are defined for this dynamic object number, then an error of noSuchName shall be returned by the agent."
::= { dynObjData 11 }

dynObj12 OBJECT-TYPE
SYNTAX OCTET STRING
ACCESS read-write
STATUS deprecated
DESCRIPTION
"The value of this object is determined by the dynObjDef entries with dynObjNumber equal to 12. Packed Encoding Rules are utilized to encode the objects for transmission. This object is intended for use with the Simple Transportation Management Protocol, and provides little advantage if used with SNMP.
If no objects are defined for this dynamic object number, then an error of noSuchName shall be returned by the agent."
::= { dynObjData 12 }

dynObj13 OBJECT-TYPE
SYNTAX OCTET STRING
ACCESS read-write
STATUS deprecated
DESCRIPTION
"The value of this object is determined by the dynObjDef entries with dynObjNumber equal to 13. Packed Encoding Rules are utilized to encode the objects for transmission. This object is intended for use with the Simple Transportation Management Protocol, and provides little advantage if used with SNMP.
If no objects are defined for this dynamic object number, then an error of noSuchName shall be returned by the agent."
::= { dynObjData 13 }


```
--*********************************************************************
--          Dynamic Object Configuration Group
-- *********************************************************************

dynObjConfigTable  OBJECT-TYPE
     SYNTAX  SEQUENCE OF DynObjConfigEntry
     ACCESS  not-accessible
     STATUS  mandatory
     DESCRIPTION
          "A table consisting of an owner and status for each of the 13 dynamic object definitions."
::= { dynObjMgmt 3 }

dynObjConfigEntry  OBJECT-TYPE
     SYNTAX  DynObjConfigEntry
     ACCESS  not-accessible
     STATUS  mandatory
     DESCRIPTION
          "A table consisting of an owner and status for each of the 13 dynamic object definitions."
     INDEX  { dynObjNumber }
::= { dynObjConfigTable  1 }

DynObjConfigEntry ::= SEQUENCE {
     dynObjConfigOwner       OwnerString,
     dynObjConfigStatus      DefEntryStatus }

dynObjConfigOwner OBJECT-TYPE
     SYNTAX  OwnerString
     ACCESS  read-write
     STATUS  mandatory
     DESCRIPTION
          "The entity that configured the associated dynamic object."
::= { dynObjConfigEntry 1 }


dynObjConfigStatus OBJECT-TYPE
     SYNTAX  ConfigEntryStatus
     ACCESS  read-write
     STATUS  mandatory
     DESCRIPTION
          "Indicates the state of the associated dynamic object.  Depending on the validity checks that are
          performed on the dynamic object definition, a set request may or may not be honored.  See
          Section 4.1.4 for a complete description."
::= { dynObjConfigEntry 2 }


END --x TMIB-II DEFINITIONS
```

< This page has intentionally been left blank. >

§

