



NTCIP SMI

<https://ite-org.github.io/NTCIP-8004/latest/>

AASHTO / ITE / NEMA

CC-BY-4.0 (<https://creativecommons.org/licenses/by/4.0/>)

Table of contents

Front Matter	5
	5
Notices	6
Acknowledgements	8
Foreword	9
Introduction	11
General	12
Scope	12
References	12
General Statements	13
Glossary	14
Background [Informative]	19
Introduction	19
Management Information Base (MIB) and Module	20
Types of Object Types	21
Document Overview	24
Object Identification [Normative]	26
International Object Identifier Tree	26
Registered Nodes	27
Rules for Module Development [Normative]	30
General	30
Rules for Modules	32
Rules for Values	35
Rules for Macros	37
Additional NTCIP Requirements	68
	68
	75
	75
	75
	75
	75
	75
	75
	75


	76
	76
	76
	76
	76
	76
	76
Object Identifier Layout	76
MIB Design Considerations	78
Requirements for Agent Implementations [Normative]	82
Agent Capabilities	82
Requirements for Non-Standard MIBs	82
Default Values	82
Extensions of Standardized Enumerations	82
Guidelines for Operating Agencies [Informative]	85

Front Matter

Working Group Draft


NTCIP 8004 {{ release_number }}

National Transportation Communications for ITS Protocol
Structure and Identification of Management Information (SMI)

 June 13, 2025

Notices

Copyright

[NTCIP SMI](#) is licensed under the **Creative Commons Attribution 4.0 International (CC BY 4.0)**  by the American Association of State Highway and Transportation Officials ([AASHTO](#)), the Institute of Transportation Engineers ([ITE](#)), and the National Electrical Manufacturers Association ([NEMA](#)).

The CC BY 4.0 license requires that reusers give credit to AASHTO, ITE, and NEMA. It allows reusers to distribute, remix, adapt, and build upon the material in any medium or format, even for commercial purposes.

Content and Liability 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 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.

Trademark Notice

NTCIP is a trademark of AASHTO / ITE / NEMA. All other marks mentioned in this project are the trademarks of their respective owners.

 June 13, 2025

Acknowledgements

This document was prepared through an open-source standards development process with the following active contributors:

contributors **1**

Check out the full list of [contributors here](#).

In addition, the following submitted comments during the process:

- k-vaughn

The resultant document is maintained by the NTCIP Base Standards, Profiles and Protocols (BSP2) Working Group (WG), a subdivision of the Joint Committee on the NTCIP. The Joint Committee on the NTCIP 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 development organizations (SDOs) and provides guidance for NTCIP development.

🕒 June 13, 2025

Foreword

Overview

This document is an NTCIP Open-Source NTCIP Process, Control, and Information Management document provided as Interim for Field Release (IFR).

Open-source documents are developed using the ITS Open-Source Process, as defined in NTCIP 8008. This process provides an open standards development process that accepts issues reported by the community and resolved by peer-reviewed contributions from the community. The open source process concludes with the resultant material being approved by the defined approval process.

IFR documents are approved through a streamlined process focused on the technical experts of the community (e.g., those participating in the open-source development process) rather than through a formal ballot of industry managers.

NTCIP Process, Control, and Information Management documents define the practices and policies used by the NTCIP Joint Committee and its working groups in developing and maintaining NTCIP publications.

This document defines the process for developing projects for the ITS community using an open-source environment (e.g., GitHub). The project can produce any type of product, such as a guide, a technical specification, a test procedure (e.g., including code), etc.

The approval process for the resultant open-source product is based on the target level of specification. For example, an IFR specification undergoes a less formal approval process than a full standard.

Approvals

IFRs are peer reviewed within the open-source process with final approval by an associated WG established by the NTCIP Joint Committee.

Approval information is provided within the online environment.

For more information about NTCIP standards, visit the NTCIP Web Site at www.ntcip.org.

User Comment Instructions

Comments can be submitted at any time. In preparation of this NTCIP standards publication, input of users and other interested parties was sought and evaluated.

Comments on open-source projects can be submitted either on the [discussions](#) or [issues](#) tab of the project.


Discussions can be initiated at any time and anyone in the community can respond, all within a public environment. Responses to discussion comments are strictly informative and may not be accurate. Discussion comments can lead to the submittal of issues that need to be resolved to clarify the standard.

Issues can be submitted at any time. Issues are triaged by the project maintainer, who will evaluate their merit, classify them (e.g., as a bug, documentation issue, omission), and in most cases respond to the submitter. Once ready, issues will be available for contributors to volunteer to address. When a volunteer has a proposed solution, it can be submitted to the project and approved in a relatively short period (when compared to the traditional standards approval process). However, updates to the projects are still version controlled so that users can reference a specific version of the project without fear of it changing.

Comments should use the templates provided on the website; otherwise they may be ignored.

History

For a history of the project, see the projects [releases](#) page.

 June 13, 2025

Introduction


This site defines the ITS Open-Source Process as used by several projects within the ITS standards community. The process follows general practices within the larger open-source community; however, this document:

- provides a step-by-step overview of the process, so that those unfamiliar with open-source processes can better understand the process and become contributors,
- formalizes the process (e.g., by clearly defining what are requirements), and
- tailors the process (e.g., by defining the preferred tools to be used).

This document contains one normative annex.

The following keywords apply to this document: AASHTO, ITE, NEMA, NTCIP, open-source, process.

This document uses only metric units.

 June 13, 2025

Section 1 General

1.1 Scope

This document specifies the process used to produce open-source documents within the field of Intelligent Transportation Systems (ITS).

The process follows general practices within the larger open-source community; however, this document:

- a. provides a step-by-step overview of the process, so that those unfamiliar with open-source processes can better understand the process and become contributors,
- a. formalizes the process (e.g., by clearly defining what are requirements), and
- c. tailors the process (e.g., by defining the preferred tools to be used).

The process to approve the resultant product is defined elsewhere (e.g., NTCIP 8001).

The ITS Open-Source Process is based on the practices defined by [open-sauced](#). However, whereas open-sauced is written as an informative guide and describes how systems can work; this document is written as a specification to define how the ITS Open-Source Process will work. While still providing a discussion of the issues; it highlights the requirements and notable options along the way by stating each in its own paragraph and boldfacing the keywords "shall" and "may" to clearly designate requirements and options. The remaining text provides further guidance and can include additional options that do not necessitate specific numbering.

We recognize that onboarding to a new project can be challenging, especially if you're new to open source development. Be patient, and don't be discouraged by setbacks or mistakes. You'll become more comfortable and confident in your contributions with persistence and practice.

1.2 References

The following documents are referenced by this document. At the time of publication, the editions indicated were valid.

1.2.1 Normative References

Normative references contain provisions that, through reference in this text, constitute provisions of this document. All standards are subject to revision, and parties to agreements

based on this standard are encouraged to investigate the possibility of applying the most recent editions of the standard listed.

- [ISO/IEC/IEEE 24765:2017: Systems and software engineering — Vocabulary, 2017](#)
- [GitHub](#)
- [MkDocs](#)
- [Materials for MkDocs](#)
- [Python](#) (for test procedure projects)

1.2.2 Other References

Other references are included to provide a more complete understanding of this document and its relationship to other documents.

1.2.2.1 Other Resources for Contributors

This document standardizes and tailors certain aspects of the information contained in open-sourced; however, it is not a complete replacement of that material. If you wish to learn more about open-source development, the following materials may be of interest:

- [What is open-source?](#)
- [Why open-source?](#)
- [The Secret Sauce](#)
- [Types of Open-Source Contributions](#)
- [Open Source Guides](#)
- [Introduction to GitHub and Open Source Projects](#)

1.2.2.2 Other Resources for Maintainers

If you wish to learn more about open-source maintenance, the following materials may be of interest:

- [Understanding the Role of an Open Source Maintainer](#)
- [How to Communicate and Collaborate Effectively](#)
- [Building Community](#)
- [Maintainer Power Ups](#)
- [Building Your Team](#)
- [The Power of Open Source Metrics](#)
- [Contributor Ladder Template](#)
- [Maintainer Community](#)

1.3 General Statements

The remainder of this document is broken into the following chapters:

- **Overview:** Provides an overview of the entire process
- **Commenter Responsibilities:** Details the responsibilities of those reviewing open-source materials and provides step-by-step processes for using the preferred tools.
- **Contribution Responsibilities:** Details the responsibilities of those contributing to open-source materials and provides step-by-step processes for using the preferred tools.
- **Maintainer Responsibilities:** Details the responsibilities of those assigned to maintain an ITS open-source project. This includes processes for setting up new projects, managing issues and pull requests, maintaining quality, and coordinating with standard development organizations.
- **WG Responsibilities:** Defines the responsibilities of the working group assigned to manage an ITS open-source project.
- **Code of Conduct:** Defines the default code of conduct for ITS open-source processes. These can be refined for any particular project, if needed, but most projects should be able to use this text without modification.
- **Documentation Conventions:** Defines the preferred styles, processes, and tools for developing documentation for ITS open-source projects, including projects that are 100% documentation (e.g., the ITS Open-Source Process project).
- **Code Conventions:** Defines the styles, processes, and tools for developing computer code for ITS open-source projects, including Python and ASN.1.

Note

It is expected that a future edition will define the responsibilities of the committee that establishes ITS open-source projects.

Note

It is expected that a future edition will define preferred ways to use requirement management tools to produce content that can be easily integrated into the ITS open-source projects while providing clear traceability.

Note

It is expected that a future edition will move information about the approval of releases to NTCIP 8001.

1.4 Glossary

For terms not defined here, English words are used in accordance with their definitions by the [merriam-webster online dictionary](https://www.merriam-webster.com/). Electrical and electronic terms not defined in this section or in Webster's New Collegiate Dictionary are used in accordance with their definitions in ISO/IEC/IEEE 24765:2017.

backlog

A backlog is a list of tasks that need to be completed within a project. Typically, these are tasks that are not yet assigned to a developer and are waiting to be worked on. Sometimes, these could be tasks that were open weeks or months ago and are still waiting to be worked on.

branch

A branch is a separate version of the code that's created for development purposes. Branches allow contributors to experiment with changes without affecting the main codebase. When changes are ready to be merged into the main codebase, they're typically submitted as a pull request.

bug

A bug refers to an error, flaw, or defect in code that adversely affects the proper functioning of the software. Open source projects often depend on contributions from the community to identify and rectify these bugs.

clone: Cloning is the process used to copy an existing Git repository into a new local directory. The `git clone` command will create a new local directory for the repository, copy all the contents of the specified repository, create the remote tracked branches, and checkout an initial branch locally. By default, Git clone will create a reference to the remote repository called origin.

code freeze: A code freeze is a period of time where no new code is added to a project. It is often used to prepare for a release and ensure that the code is stable and ready for production.

code review: A code review is when a maintainer or contributor will review the work of another contributor. This is a great way to ensure that the code is high quality and meets the standards of the project.

containerization: Containerization is a way of packaging and running applications. Instead of installing an app directly on your computer, you put it in a container that includes everything it needs to work. This container can then run on your computer alongside other containers. It's a way to organize and run multiple applications on the same machine, making it easier for developers to manage and scale their applications.

continuous integration (CI): Continuous integration (CI) is a development approach in which developers regularly merge code into a shared repository. For each change, an automated build and test process is run to detect errors as quickly as possible.

continuous deployment (CD): Continuous deployment (CD) is often associated with continuous integration (CI) and refers to keeping your application deployable at any point or even

automatically releasing to production. CD means that every change which passes the automated tests is deployed to production automatically.

contributor: A contributor is anyone who makes changes, additions, or suggestions to an open source project. Contributors can be developers, designers, writers, testers, or anyone else who helps to make the project better.

core member: A core member is a contributor who has been granted additional privileges or responsibilities within an open source project. Core members are typically trusted contributors who have demonstrated a deep understanding of the project and have made significant contributions to its development.

docs: Docs is an abbreviation for "documentation". It primarily explains how to implement and use a product or an open source project. It also provides information on how to contribute to the project and expectations for contributors. Documentation is often written using [Markdown](#), a lightweight markup language.

fork: A fork is a copy of a repository. When you fork a repository, you create a new copy of the codebase that you can modify and experiment with without affecting the original codebase.

GitHub actions: GitHub Actions are a way to automate tasks within your software development life cycle. GitHub Actions are event-driven, meaning that you can run a series of commands after a specified event has occurred. Examples of GitHub Actions include running tests, deploying to production, and sending notifications.

GitHub discussions: GitHub Discussions are a way to have conversations about your project directly in GitHub. They are a great way to discuss ideas, ask questions, and share knowledge with your community.

issue: An issue is a problem or bug that needs to be addressed in the code. Issues can be created by anyone, and they're often used to keep track of bugs, feature requests, and other tasks that need to be done.

linting: Linting is the process of running a program that will analyze code for potential errors. A popular linting tool used frequently is ESLint. You can setup an action to run ESLint against each pull request that comes in to check for potential errors before it makes it into production.

maintainer: A maintainer is a person or a group of people responsible for maintaining a specific open source project. Maintainers are typically responsible for reviewing and accepting or

rejecting contributions from other contributors. They also have the authority to make final decisions about the direction and scope of the project.

markdown: Markdown is a lightweight markup language commonly used for creating formatted text documents. It is widely used for creating documentation and README files in software development due to its simplicity and readability.

merge: Merging is the process of combining changes from one branch into another. When a pull request is accepted and merged, the changes made in the pull request become part of the main codebase.

onboarding: Onboarding documentation helps new team members or collaborators quickly become familiar with a project's structure, goals, and processes.

OSS Projects: OSS stands for "Open Source Software" projects. These are software projects where the source code is made available to the public, allowing anyone to view, use and modify the software.

pull request: A pull request is a request from a contributor to a maintainer for changes made to the code to be pulled into a codebase.

quality assurance: Quality assurance in open source projects involves testing, reviewing, and ensuring the software meets the desired standards. Community members often contribute to testing and reporting issues to improve the software's quality.

release candidate: A release candidate is a beta version of software with the potential to be a final product. It is typically the last version before the final release.

release notes: Release notes are documents that detail changes, enhancements, bug fixes, and new features in each software release. They inform users and stakeholders about what to expect in a new version of the software.

repository: A repository is a central location where code is stored and managed. In open source, repositories are often hosted on platforms like GitHub, GitLab, or Bitbucket. Each repository can contain one or more projects, and contributors can submit changes to the code by making pull requests.

style guide: A style guide is a set of rules and conventions that define the preferred formatting, writing style, and visual elements used in documentation and other content. This helps maintain consistency and clarity across documents, making them easier to read and understand.

versioning: Versioning is the process of assigning either unique version names or numbers to new releases of your project. Some versions are released as "major" versions, while others are released as "minor" versions.

 June 13, 2025

Background [Informative]

Introduction

Within NTCIP, transportation equipment is managed primarily using the Simple Network Management Protocol (SNMP) to exchange transportation management information with transportation management applications (e.g., within a central system). This type of exchange is commonly called Center-to-Field (C2F) communications. Within SNMP terminology, the transportation equipment includes an SNMP agent that responds to requests received from a transportation management application, which is known as an SNMP management application (or manager). SNMP requests and responses share a similar structure, which can be depicted in the Unified Modeling Language (UML), as shown in Figure 1 (with slight simplifications for the purpose of this discussion).

SNMP Message Structure{width="6.35277777777778in" height="3.983333333333334in"}

Starting in the upper-left corner of Figure 1, a message is the logical SNMP structure that is serialized as a data packet (i.e., byte-stream) and exchanged by lower-layer protocol entities (e.g., TCP/IP). It consists of a variety of information, such as protocol header, version information, security parameters, and a potentially encrypted scoped protocol data unit (PDU). The scoped PDU includes the context for the message, error information, the request identifier, and a series of variable bindings represented as a name/value ordered pair. Within this structure, each variable binding (i.e., name/value ordered pair) is the serialized representation of an object instance stored within the identified SNMP context. An object instance is an instance of a defined object type, which is a formal abstract definition of a piece of data.

For example (in reverse order), NTCIP 1204, which defines data for environmental sensor stations, defines an object type called essAirTemperature. An environmental sensor station might support multiple temperature sensors; each sensor would be associated with its own instance of the object type, identified as "essAirTemperature.\<sensor number>". An SNMP agent could report the value of any object instance by creating an ordered pair of the name and value of the object instance and enclosing it in an SNMP message, potentially with other variable bindings.

In order for the SNMP manager to understand the data contained within each message, the SNMP manager and SNMP agent must agree on:

a) a mechanism to unambiguously identify the object type and object instance,

- b) the precise semantics (i.e., meaning) of the data value,
- c) the way in which the data will be encoded,
- d) the operations that are allowed (e.g., can a manager alter the value or is it read-only), and
- e) the data that should be supported.

These details are specified using a formalized ASCII-NVT text file known as a Management Information Base (MIB) module coupled with the definition of SNMP, as defined in RFCs 2578-2580 and RFCs 3411- 3418.

Management Information Base (MIB) and Module

A Management Information Base (MIB) is the information that a specific SNMP context (typically a device) can make available for management operations. A MIB is defined by a text-based "MIB Module" (or collection of MIB Modules) that specifies the collection of object types that that the device supports. A MIB module is written using formal conventions in a structure that can read by both humans and computers. Object types include controls, configuration parameters, and status information (including sensor values and internal device status). Any data that is to be accessible via SNMP must be defined in a MIB module.

The standardized format of a MIB module allows automating much of the syntactic processing of data with off-the-shelf SNMP tools and allows developers to focus on implementing the semantics according to the documented definitions contained with the MIB module. The rules for the structure and management of this information also allow multiple MIB modules (e.g., NTCIP standard, Internet standard, manufacturer-specific extensions) to be used for a single device. The MIB modules are integrated into the device software and its management stations using various methods (e.g, implemented directly as data structures in the software, compiled into binary tables that facilitate use by operational software). It is common for the MIB modules to be called the MIB but, technically, this is incorrect. The MIB is the instantiated collection of data objects available to management stations to configure, control, and monitor the device. It is not a database in the traditional sense.

Figure 2 illustrates how using MIB modules makes NTCIP C2F communications possible. In this example, the field devices are shown on the left of the figure with their associated MIB modules. The dynamic message sign uses only objects defined in standards while the traffic signal controller includes manufacturer-specific objects. The management station is commonly called a "central system" in transportation systems.

Diagram Description automatically generated{width="5.909722222222222in" height="4.565277777777778in"}

Integrated MIB Modules Make NTCIP C2F Communications Possible

Types of Object Types

Within SNMP, each piece of management information that can be exchanged is called an object instance. Object instances are abstracted into object types and formally defined within the MIB module. However, when reading a MIB module, it is important to understand the different types of object types that might occur.

Leaf Object Types

Leaf object types do not have any globally unique sub-identifiers specified and are the only object types that can be instantiated in a device and retrieved via SNMP operations. The sub-identifiers for leaf object types are reserved for the identification of specific instances of the object type; they are only unique within the indicated device context (e.g., the object type sysName is a leaf object type and has a globally unique identifier; however, every SNMP device that instantiates this object type will use the same instance identifier). Leaf objects can be further classified in three different ways:

- a) Based on whether multiple instances are allowed,
- b) Based on the complexity of the underlying data, and
- c) Based on the types of operations that are permitted.

Classification by Instantiation Rules

If a leaf object type is defined as a part of a conceptual row of a table, multiple instances of the object are allowed, one for each row that has been conceptually instantiated within the table. This type of leaf object type is known as a "columnar object type". The different instances of the columnar object type are defined according to the mechanism defined by the INDEX clause of the conceptual row using one or more sub-identifiers.

If a leaf object type is not defined as a part of a conceptual table, only one instance of the object can exist; this object type is called a \"scalar object type\". The singular instance of a scalar object type is always identified by the single sub-identifier instance number of \"0\".

Complexity of Object Types

SNMP was designed to exchange elemental data; as such, leaf object types are not allowed to directly use any ASN.1 data structure (e.g., SEQUENCE, SEQUENCE OF, CHOICE) for its syntax. This allows SNMP to provide very flexible data exchanges based on the need of the user. However, within a message, each elemental object instance must be uniquely identified, which adds overhead.

Within NTCIP, there are often needs to frequently exchange the same set of object instances in environments where the amount of data that is exchanged is a concern (e.g., communication environments with data usage limitations). These environments result in a need for a more efficient solution than always exchanging data in its most elemental form. To overcome these issues, the NTCIP classifies every leaf object type as a simple, block, or configurable object type.

Simple object types represent elemental data, as traditionally defined with normal SNMP operations. Simple object types include integers, octet strings, object identifiers, bit strings, and other types that derive from these basic types.

Block object types represent data structures (e.g., SEQUENCE, SEQUENCE OF, CHOICE) that are normally prohibited as SNMP object types. NTCIP gets around the SNMP prohibition by setting the SYNTAX clause of the OBJECT-TYPE macro to resolve to an OCTET STRING (typically by using a textual convention). It then defines the value of that OCTET STRING to be a serialization of a defined data structure. From the SNMP perspective, the data is an elemental OCTET STRING, but the sender and receiver perform additional encoding and decoding to allow for the exchange of a more complex structure. Within NTCIP, the preferred way to define the serialization of these data structures is by using an ITSOerString, which is defined by ISO 20684-1 as a data structure specified using X.680 ASN.1 and serialized using the Octet Encoding Rules (OER).

Configurable object types are similar to block objects in that each configurable object represents a data structure that can be serialized into an OCTET STRING and then be exchanged as a single SNMP object type. However, block object types have static definitions of their data structures defined within the MIB module. By comparison, configurable object types allow for a manager to configure the definition of the data structure after the implementation has been deployed and

while the SNMP agent is running. Within NTCIP, configurable object types are typically SEQUENCE structures and the configuration is typically defined in a table where each ordered row of the table represents an ordered ComponentType (i.e., field) within the SEQUENCE structure. The serialization of the configurable object typically uses OER to produce an OCTET STRING that can be exchanged by SNMP.

Object Type Permissions

SNMP object types can also be classified based on the types of data represented and the operations that are allowed under different scenarios. There are four basic types of object types from this perspective, as follows:

- a) Status object types. Read-only objects that report the conditions that can be monitored by the SNMP agent.
- b) Control object types. Writeable object types used to request real-time activation of a feature of a device. In some cases, control objects can also be used to report status.
- c) Parameter object types. Writeable object types used to configure the SNMP agent where the parameter can be set and validated using a single SNMP set operation.
- d) Interrelated parameter object types. Writeable object types used to configure the SNMP agent where the parameter has sufficient interrelationships with other object types to typically require multiple SNMP set operations using multiple SetRequest messages or a complex validation check that might consume more time than is reasonable for a traditional SNMP response.

When a device includes interrelated parameter object types, it defines a mechanism by which the object types can be safely configured. One such approach is to use the database transaction mode as specified in NTCIP 1201.

Conceptual Objects

In addition to the leaf object types described above, MIB modules can define three types of conceptual object types: conceptual tables, conceptual rows, conceptual leafs. Conceptual objects can theoretically be instantiated but because they have a MAX-ACCESS value of \"not-accessible\", the rules of SNMP do not allow their exchange and they do not really exist as protocol entities. An implementation can only exchange instances of the accessible object types.

A conceptual table defines a set of object types that might traditionally be represented in a written document using a table format. Logically, each conceptual table consists of a number of columns and rows, where each column describes a particular type of data and each row represents a unique instance of data for each column. A conceptual table describes the information contained in the table and the rows that the table should have. It has a SYNTAX of "SEQUENCE OF \<EntryType>", where \<EntryType> is the name of the conceptual row's data structure.

A conceptual row defines a set of object types that represent a logical unit of inter-related management information. The row typically supports multiple columns and the object types referenced by that row logically has one instance for each defined row. The row object type defines the rules for the number of rows and the rules for creation and deletion of rows.

A conceptual leaf is a simple object type that has been identified as "\"not-accessible\""; for example an index to a table. While SMIv1 allowed for index object types to be accessible, it provides no real value and increases the time it takes to walk through the data (i.e., by using GetNextRequests) supported by the SNMP agent. As a result, SMIv2 prohibits index object types from being accessible, unless they have been imported from an SMIv1 module. Conceptual leaves can also be defined when a standard wants to declare data that should be supported by a device even if it is never exchanged. For example, the trigger function is required to monitor data within the device and needs to be able to demonstrate that it has appropriate security credentials to access the data. The security credential information is stored in non-accessible objects to indicate that the controller needs to maintain this information but that the information should never be exchanged.

Document Overview

The remaining sections of this document define:

- ¹. The mechanism used to allow multiple independent entities to define object types while maintaining globally unique names
- ². Requirements for the development of MIB modules
- ³. Requirements for agent implementations
- ⁴. Considerations for operating agencies

In addition, Annex A provides the formal definition of the NTCIP 8004 MIB, which must be imported by all other NTCIP MIBs and Annex B provides the rules for converting NTCIP MIBs that are based on SMIv1 into the SMIv2 format required by this document.

🕒 June 13, 2025

Object Identification [Normative]

SNMP allows managers and agents to implement MIB modules from multiple independent sources. Allowing this combination of object types presents a potential for naming conflicts (i.e., two sources defining different pieces of data using the same name). SNMP overcomes this challenge by identifying object types using the international object identifier tree, as defined by the ISO/IEC 9834-1 standard.

International Object Identifier Tree

The international object identifier tree consists of three root nodes^[^3]. Each root node can be assigned multiple sub-nodes. Each sub-node may, in turn, have sub-nodes of its own.

Each node, whether a root node or a sub-node, is managed by some organization. For example, the three root nodes are managed by ISO, the International Telecommunication Union Telecommunication Standardization Sector (ITU-T), and jointly by ISO and ITU-T. The manager of any node may delegate the management responsibility for any sub-node underneath its branch. In this case, the delegated node is termed a subtree. Just as any sub-node may have its own set of sub-nodes, subtrees can have their own subtrees.

Except for the first two root nodes (which are limited to 40 sub-nodes each), the number of sub-nodes that any node may have is unlimited. Likewise, the number of sub-node levels is unlimited. As a result, the international object identifier tree can contain an unlimited number of nodes using a tree structure managed by multiple organizations.^[^4]

Node Identification

Each node is assigned an integral number and optionally a label. The label is called the OBJECT DESCRIPTOR and provides a user-friendly textual name that can concisely express what the node is intended to represent. While the OBJECT DESCRIPTOR provides a useful human-language term to describe the node and should be designed to be unique within the scope of other sibling nodes of the same branch and level, there is no guarantee that it is unique within the entire international object identifier tree. Nonetheless, any node can be uniquely defined by following the sequence of nodes from the root to the specific node that needs to be identified. Unfortunately, this can produce a rather lengthy description since each name might consist of several characters. Alternatively, the same identifier can be produced by using the integral numbers assigned to each node rather than the OBJECT DESCRIPTORS. This representation

typically results in a much more compact identifier that can more easily be processed by computers. This ordered list of integral values is called an OBJECT IDENTIFIER, which is known to identify a globally unique position on the identifier tree when properly registered.

The globally unique identifier can be used for any purpose for which an identifier may be useful. For example, most standards organizations have been assigned an OBJECT IDENTIFIER for the purposes of identification. The tree can also be used for managing groups of related data. For example, all objects related to an Actuated Signal Controller are organized under a node defined as 'asc'. In short, these attributes are a means for identifying some object, regardless of the semantics associated with the object.

Object Identifiers and SNMP

Within SNMP, the international object identifier tree is used to register object types with globally unique identifiers. The formal assignment of an OBJECT IDENTIFIER to each object type is contained in the MIB module.

SNMP also identifies object instances by using the international object identifier tree, but with a slight twist in that the instance portion of the identifier is only unique within its SNMP agent and context^[5] – not globally unique. For example, while the MIB provides a globally unique identifier for the object type of essAirTemperature, there might be multiple environmental sensor stations each containing multiple air temperature sensors. To identify a specific object instance within a specific environmental sensor station, SNMP extends the object type object identifier with an instance identifier (e.g., essAirTemperature.1 to identify the reading from the first sensor). However, the instance value is not unique across different SNMP agents (i.e., every environmental sensor station will use essAirTemperature.1 to identify the reading from its first air temperature sensor). The SNMP manager is responsible for distinguishing among these readings by combining the object instance identifier with the network address of the device and the context used within the message.

Registered Nodes

Root Nodes

The first two root nodes of the naming tree are administered by ITU-T (itu-t) and ISO (iso). The third node is jointly administered by ISO and ITU-T (joint-iso-itu-t, or sometimes shortened to joint).

NOTE---Until 1991, the U.S. name-registration authority conducted its business under the {iso(1) member-body(2) us(840)} arc, registering names for ANSI standards, private organizations with U.S. national standing, and the names of U.S. states and \"state equivalents.\" In 1991, changes in the registration authority procedures standard ISO/IEC 9834 (ITU-T X.660) invalidated this procedure, requiring private organization names with national standing to be registered under the {joint-iso-itu-t(2) country(16) us(840)} arc. The existing register of private organization names moved, in fact, from the {1 2 840} arc to the {2 16 840 1} arc, with ANSI serving as the registration authority. Therefore, two equivalent prefixes exist (in perpetuity) for currently registered organization names. Post-1991 registrations are made only under the {2 16 840 1} arc, and organizations with pre-1991 registrations are encouraged (but not required) to construct no new identifiers under the {1 2 840} arc. Various sources provide current OID descriptions, including www.oid-info.com/index.htm.

NOTE--- The International Telecommunication Union Telecommunication Standardization Sector (ITU-T) is the part of ITU (an agency of the United Nations) that provides standards for telecommunication equipment and systems. The Consultative Committee for International Telephony and Telegraphy (CCITT) was renamed ITU-T in 1993. The {itu-t(0)} arc is also named \"ccitt\" in remembrance that CCITT was previously an organization independent from ITU-T. Similarly, the {joint-iso-itu-t(2)} arc is also named joint-iso-ccitt.

NEMA Node

Under the \"iso\" node, ISO has designated one subtree for use by other (inter)national organizations (org). Under that subtree, one of the U.S. National Institute of Standards and Technology nodes is assigned to the U.S. Department of Defense (dod). The initial development of the Internet was a Department of Defense project and, therefore, the Internet community was assigned a node in the dod subtree. The Internet Architecture Board (IAB) administers the \"internet\" node. The descriptive name \"internet\" is defined as:

internet OBJECT IDENTIFIER ::= { iso org dod 1 }

(also known as 1.3.6.1)

Because of the ease of obtaining a node from IAB, NEMA requested and received a node that NEMA administers. This node is defined as:

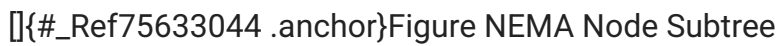
nema OBJECT IDENTIFIER ::= { iso org dod internet private enterprise 1206 }


(also known as 1.3.6.1.4.1.1206)

All NTCIP-defined data related to device data dictionaries or protocols shall be defined under the NEMA branch of the tree. The organization of the naming tree down to the \"nema\" node is shown in Figure 3. The figure also shows the node for the ITS industry as used by the ISO 20684 standard series.

Portion of ISO Global Tree Showing Location of NEMA Node

The subtree for the NEMA node is shown in Figure 4. The description of each of the nodes are found in Annex A.1.

Figure NEMA Node Subtree

 June 13, 2025

Rules for Module Development [Normative]

Except as otherwise stated within this section and in Annex B, all NTCIP MIB modules shall conform to:

- ¹. IAB STD 58, which includes:
 - a. RFC 2578,
 - b. RFC 2579, and
 - c. RFC 2580.
- ². RFC 3584
- ³. RFC 4181
- ⁴. The additional constraints and conventions defined in this section

Annex B defines allowed exceptions that only apply when converting NTCIP MIB modules from the IAB STD 16 (SMIv1) format to the IAB STD 58 (SMIv2) format.

NOTE -- SMIv2 is the second version of the SMI but is used as the MIB module format for SNMPv3.

General

The original text of the RFCs is indispensable for a full understanding of the MIB module format; however, RFCs are typically written in a conversational/explanatory tone which can result in redundancies and ambiguities as to the conformance status of each requirement. Further, some requirements in the referenced RFCs are written to only be applicable to IETF standard MIB modules while others are for non-standard MIB modules and tend to be from a technical perspective of what is allowed as opposed to what should be done. The conformance requirements adopted by NTCIP are intended to establish a policy for writing NTCIP standard MIB modules and as a result reflect a slightly different mix of conformance statuses from what the IETF has defined. Finally, the requirements for developing a MIB module are scattered over multiple RFCs, which describe not only the rules for writing the MIB modules but also on implications to implementations.

The tables presented in this section clarify and highlight the conformance status of certain requirements and is limited to those rules related to writing standard NTCIP MIB modules. Rules

that are specified within X.208 and within the information modules defined within the referenced RFCs are omitted as they are unambiguous, and no changes are made. This document focuses on the requirements that are only defined within the text (non-module) sections of the RFCs.

This document groups requirements by topic, regardless of the source of the requirement. Each requirement is listed in a tabular format with an item number that allows for easy referencing. The tabular format consists of the following columns:

- Item: provides a unique identifier for the requirement.
- Requirement: A requirement stated as a \"shall\" statement. This almost always requires a rewording of the statement compared with original RFC text.
- RFC: When the requirement represents an additional constraint or convention of this document, this column is blank; otherwise, it identifies the RFC number from which the requirement originates.
- Clause: When the requirement represents an additional constraint or convention of this document, this column is blank; otherwise, it identifies the location of the requirement within the referenced RFC in the following format:

```
\<clause>"p"\<paragraph>
```

```
where,
```

- \<clause> indicates the clause number of the RFC where the requirement appears^[6]
- \<paragraph> indicates an ordered integral position of the paragraph within the indicated clause where the requirement appears (numbered and bulleted items are considered to be a part of the prior paragraph)
- RFC status: An indication of the conformance status of the requirement as interpreted from the RFC for RFC standard MIB modules.
- NTCIP status: An indication of the conformance status of the requirement for NTCIP standard MIB modules.

The codes used within the status columns are defined as follows:

Code Meaning

- m Mandatory
- r Recommended
- o Optional
- o.# An option group where at least one of the items with the same number (#) must be supported
- d Discouraged
- x Prohibited
- c Conditional

: Status Codes

Status codes may be supplemented with footnotes that provide further details.

Any violations of the requirements contained within this document shall not negate the intent of any statement contained in a document or MIB module based on this document.

Non-standard (e.g., manufacturer-specific) MIB modules are outside the scope of this document.

Rules for Modules

General Rules for Modules

Requirements for the construction of X.208 modules within NTCIP standards are shown in Table 6. Most X.208 modules defined within NTCIP standards are MIB modules (i.e., they use the OBJECT-TYPE and/or the NOTIFICATION-TYPE macros). NTCIP standards also include a small number of other module types, such as the NTCIP8004-NEMA module, contained in Annex A of this document, which does not include any OBJECT-TYPE or NOTIFICATION-TYPE macros. The requirements presented in Table 2 apply to all X.208 modules contained in NTCIP standards.

Item	Requirement	RFC	Clause	RFC	NTCIP\	
	Status	Status				
a.	X.208 modules shall conform to the	2578	1p1	m	m	
	adapted subset of X.208 ASN.1 as					
	specified in IAB STD 58					

b. Each modulereference (i.e., module | 2578 | 3p6 | m | m |
 || name) shall be unique across all ||||
 || standard modules ||||

+-----+

c. Each NTCIP module shall have a |||| r |
	modulereference (i.e., name) that			
	follows the form:			
	> NTCIP\<StandardNumber>\<NodeName>			
	where,			
	> \<StandardNumber> is the four-digit			
	> standard number			
	>			
	> \<NodeName> is the name of the			
	> MODULE-IDENTITY descriptor in			
	> UpperCamelCase			
	Example: NTCIP1201-RecMechV2			

+-----+

d. Each revision[^7] of a module (e.g., | 2578 | 3p6 | r | m^7^ |
	as contained in a new version of a			
	standard) shall use the same			
	modulereference (i.e., module name)			

+-----+

e. Modules shall use an empty | 2578 | 3p6 | m | m |
	AssignedIdentifier (i.e., there shall			
	not be an OID between the module name			
	and the "DEFINITIONS" keyword)			

+-----+

f. Modules shall not IMPORT any | 2578 | 3.2p4 | m | m |
 || "UNIVERSAL" types defined by ASN.1 ||||

+-----+

g. Modules shall not IMPORT the BITS | 2578 | 3.2p4 | m | m |
 || construct ||||

+-----+

h. Modules shall not IMPORT items not | 4181 | 4.4p4 | r | r |
 || used ||||

+-----+

i. Modules shall not define additional | 2578 | 3p5 | m | m |
 || X.208 ASN.1 macros ||||

+-----+

j. Modules shall not IMPORT or use SMIv1 | 2578 | 3p5 | m | m |
 || macros ||||

+-----+

k. Normative text shall not be placed in | 2578 | 3.4p1 | r | m |
 || module comments ||||

+-----+

l. A standard containing a MIB module | 4181 | 3p2 | m | m[^8] |
 || shall contain certain defined sections ||||

+-----+

m. A text-only, electronic version of the | - | - | m |
 || MIB module shall be produced per the ||||
 || rules of NTCIP 8005 ||||

+-----+

```

| n. | Clause numbers included in the | - | - | m | |
|| standardized NTCIP document shall be || || |
|| converted into comments in the || || |
|| electronic version of the MIB module || || |
+-----+
| o. | A standards-track MIB module shall be | 4181 | 1p1 | m | m |
|| reviewed by an approved steward || || |
+-----+
| p. | The rules defined in this document | 4181 | 4.9p9 | o | o[9] |
|| shall be waived when justification is || || |
|| fully documented || || |
+-----+

```

: General Rules for Modules

Rules for Module Components

The components contained within a module should follow a consistent order to facilitate use. The requirements for components of an NTCIP module are shown in Table 3.

```

+-----+
| Item | Requirement | RFC | Clause | RFC | NTCIP\ |
| || | Status | Status |
+=====+=====+=====+=====+=====+
| a. | Modules shall not use the EXPORTS | 2578 | 3.3p1 | m | m |
|| statement || || |
+-----+
| b. | Modules shall use the IMPORTS | 2578 | 3.2p1 | m | m |
|| statement to import all referenced || || |
|| external objects, including the || || |
|| referenced types and macros defined || || |
|| in [RFC 2578], [RFC 2579], and || || |
|| [RFC 2580] as needed || || |
+-----+
| c. | The module shall include exactly one | 2578 | 3p7 | m | m |
|| MODULE-IDENTITY macro immediately || || |
|| after the IMPORTS statement || || |
+-----+
| d. | The module shall define any local || - | r | |
|| TEXTUAL-CONVENTIONS immediately after || || |
|| the MODULE-IDENTITY macros || || |
+-----+
| e. || || || |
+-----+
| f. || || || |
+-----+
| g. | The module shall present any || - | r | |
|| OBJECT-IDENTITY, OBJECT-TYPE, and || || |
|| NOTIFICATION-TYPE macros after and and || || |
|| all TEXTUAL-CONVENTIONS contained in || || |
|| the module || || |
+-----+
| h. | The module shall define at least one | 2580 | 5p2 | c[10] | c[11] |
|| MODULE-COMPLIANCE macro || || |

```

```

+-----+
| i. | The module shall define the ||| - | r |
| | MODULE-COMPLIANCE macro after all ||||
| | OBJECT-TYPEs and NOTIFICATION-TYPEs ||||
| | contained in the module ||||
+-----+
| j. | The module shall define OBJECT-GROUPs ||| - | r |
| | and NOTIFICATION-GROUPs after all ||||
| | MODULE-COMPLIANCE statements ||||
| | contained in the module ||||
+-----+
| k. |||||
+-----+
| l. |||||
+-----+
| m. |||||
+-----+
| n. |||||
+-----+

```

: Rules for Module Components

Rules for Updating a Module

```

+-----+
| Item | Requirement | RFC | Clause | RFC | NTCIP\ |
| ||| | Status | Status |
+=====+:=====+:=====+:=====+:
| a. | Macro invocations (i.e., | 2578 | 10p3 | r | r[^12] |
| | definitions) shall not be moved ||||
| | from one module to another ||||
+-----+
| b. | Macro invocations (e.g., even with | 4181 | 10p4 | m | m |
| | a status of "obsolete") shall not ||||
| | be removed from an updated module ||||
+-----+
| c. | Normative changes shall not be made | 4181 | 10.1p1 | m | m |
| | to macro invocations (when ||||
| | necessary, a new macro invocation ||||
| | will be used instead) ||||
+-----+
| d. | New macro invocations shall be | 4181 | 10.2p1 | o | o |
| | defined ||||
+-----+

```

: Rules for Updating a Module

Rules for Values

Rules for Descriptors

[RFC 2578](#) uses the term "descriptor" to refer to the concepts that X.208 terms a "valuereference" and it appears to be intended to apply to "typereferences" as well.

X.208 requires a valuereference to begin with a lowercase letter and the statement takes the form \<valuereference> \<macro> ::= \<value>. X.208 requires a typereference to begin with an uppercase letter and the statement takes the form \<typereference> ::= \<macro>. For example, the name "essAirTemperature" is a valuereference; the name "DisplayString" is a typereference.

This document defines "descriptor" to mean "either an X.208 valuereference or typereference" to avoid some ambiguities and misleading statements that exist in [RFC 2578](#). The specific NTCIP requirements for a descriptor are defined in Table 5.

Item	Requirement	RFC	Clause	RFC	NTCIP\
	Status	Status			
a.	Descriptors shall be unique within a module	2578	3.1p3	m	m
b.	Descriptors shall be mnemonic	2578	3.1p3	m	m
c.	Descriptors shall not be longer than 64 characters in length	2578	3.1p3	m	m
d.	Descriptors shall not be longer than 32 characters in length	2578	3.1p3	o ¹³	o
e.	Descriptors shall be unique across all standard modules that are likely to be included within NTCIP devices ¹⁴	2578	3.1p4	m	m ¹⁵
f.	Descriptors shall start with the name of the device-type descriptor or an abbreviation thereof to promote understanding and uniqueness				
g.	Descriptors shall include a suffix that indicates a version number of any object type that is a replacement object type (e.g., essSurfaceConductivityV2)				
h.	If an imported descriptor is duplicated (e.g., same term imported from two different modules) the importing module shall use the Externalvaluereference (or Externaltypereference) format when referring to it (e.g., "\<modulereference>.\<descriptor>")	2578	3.2p2	m	m
i.	A descriptor (i.e., the term itself) shall not change after it is standardized	2578	3.6p1	m	m

```

+-----+-----+-----+-----+
| j. | A descriptor shall not be associated | 2578 | 3.6p1 | m | m |
| | with more than one meaning | | | |
+-----+-----+-----+-----+
| k. | A descriptor shall not be associated | 2578 | 3.6p1 | m | m |
| | with more than one OBJECT IDENTIFIER | | | |
+-----+-----+-----+-----+
| l. | A revision shall not change the | 2578 | 3.6p1 | m | m |
| | semantics of any standardized | | | |
| | descriptor | | | |
+-----+-----+-----+-----+

```

: Rules for Descriptors

Rules for OBJECT IDENTIFIER Values

Table 6 defines additional requirements related to the assignment of OBJECT IDENTIFIER values.

```

+-----+-----+-----+-----+
| Item | Requirement | RFC | Clause | RFC | NTCIP\ |
| | | | Status | Status |
+=====+=====+=====+=====+=====+=====+
| a. | OBJECT IDENTIFIER values shall not | 2578 | 3.5p1 | m | m |
| | exceed 128 sub-identifiers | | | |
+-----+-----+-----+-----+
| b. | The value of each sub-identifier | 2578 | 3.5p1 | m | m |
| | within an OBJECT IDENTIFIER value | | | |
| | shall not exceed 2^32-1 | | | |
+-----+-----+-----+-----+
| c. | An assignment of an OBJECT IDENTIFIER | 2578 | 3.6p1 | m | m |
| | value shall not contain a NameForm | | | |
| | component | | | |
+-----+-----+-----+-----+

```

: Rules for OBJECT IDENTIFIER Values

Rules for Macros

Rules for MODULE-IDENTITY Macro

The MODULE-IDENTITY macro is used to identify who is responsible for the MIB and to show the history of revisions. Requirements for the completion of the MODULE-IDENTITY macro within an NTCIP MIB module are as shown in Table 7.

```

+-----+-----+-----+-----+
| Item | Requirement | RFC | Clause | RFC | NTCIP\ |
| | | | Status | Status |
+=====+=====+=====+=====+=====+=====+
| a. | The LAST-UPDATED clause shall | 2578 | 5.1p1 | m | m |
| | indicate the date and time that | | | |

```

```

|| this module was last edited ||||
+-----+
| b. | The ORGANIZATION clause shall | 4181 | 4.5p2 | m | m |
|| provide the full name of the ||||
|| working group ||||
+-----+
| c. | The CONTACT-INFO clause shall | 2578 | 5.3p1 | m | m[^17] |
|| include the name of the person to ||||
|| whom technical queries should be ||||
|| sent ||||
+-----+
| d. | The CONTACT-INFO clause shall | 2578 | 5.3p1 | m | m^19^ |
|| include the postal address of the ||||
|| person to whom technical queries ||||
|| should be sent ||||
+-----+
| e. | The CONTACT-INFO clause shall | 2578 | 5.3p1 | m | x |
|| include the telephone number of the ||||
|| person to whom technical queries ||||
|| should be sent ||||
+-----+
| f. | The CONTACT-INFO clause shall | 4181 | 4.5p2 | m | m^19^ |
|| include an email address ||||
+-----+
| g. | The CONTACT-INFO clause shall | 4181 | 4.5p2 | r | o |
|| include the working group's website ||||
|| URL ||||
+-----+
| h. | The DESCRIPTION shall contain a | 2578 | 5.3p1 | m | m |
|| high-level textual description of ||||
|| the MIB module ||||
+-----+
| i. | The REVISION clause shall be | 4181 | 4.5p2 | m | m |
|| present for each standardized ||||
|| revision of the module ||||
+-----+
| j. | The time of the most recent | 4181 | 4.5p2 | m | m |
|| REVISION clause shall match the ||||
|| value of the LAST-UPDATED clause ||||
+-----+
| k. | The DESCRIPTION clause associated | 4181 | 4.5p2 | m | m |
|| with each REVISION clause shall ||||
|| indicate the version of the ||||
|| standard in which it is defined ||||
+-----+
| l. | The DESCRIPTION clause associated | 4181 | 4.5p2 | r | m |
|| with each REVISION clause shall ||||
|| provide a list of all significant ||||
|| changes ||||
+-----+
| m. | The value assigned to the | 4181 | 4.5p2 | m | m |
|| MODULE-IDENTITY descriptor shall be ||||
|| unique for the standardized module ||||
+-----+
| n. | The MODULE-IDENTITY macro shall be | 2578 | 10p2 | m | m |
|| updated for each revision ||||

```

: Rules for the MODULE-IDENTITY Macro

Rules for the OBJECT-IDENTITY macro

The OBJECT-IDENTITY macro is the preferred method to declare a node on the international object identifier tree for administrative use. For example, the NEMA "devices" node is defined in NTCIP 8004 v03 using an OBJECT-IDENTITY macro. Requirements for the completion of the MODULE-IDENTITY macro within an NTCIP MIB module are as shown in Table 8.

Item	Requirement	RFC	Clause	RFC	NTCIP\
	Status	Status			
a.	All administrative OIDs below the	4181	6p1	r	m
	nema devices node shall be defined				
	via the OBJECT-IDENTITY macro				
b.	All normative text about an object	2578	7.5p1	m	m
	identity shall be included within its				
	DESCRIPTION clause				

: Rules for the OBJECT-IDENTITY macro

Rules for the OBJECT-TYPE Macro

General Rules for the OBJECT-TYPE Macro

The OBJECT-TYPE macro is used to specify the object types contained in a MIB module. There are several groups of requirements related to the OBJECT-TYPE macro. General requirements for the OBJECT-TYPE macro are provided in Table 9.

Item	Requirement	RFC	Clause	RFC	NTCIP\
	Status	Status			
a.	Each unit of management	2578	7p1	m	m
	information shall be specified				
	using an instance of the				
	OBJECT-TYPE macro				
b.	The value assigned in the STATUS	2578	10.2p1	r	r
	clause shall not be updated to				
	restore an items status (e.g., do				
	not change "deprecated" to				
	"current"				

```

| c. | All normative text about an | 2578 | 7.5p1 | m | m |
|| object type shall be included | | | |
|| within its DESCRIPTION clause | | | |
|| (but this may include references | | | |
|| to other text) | | | |
+-----+-----+-----+-----+
| d. | The DESCRIPTION clause of each | 4181 | 4.6.2p3 | r | m[^18] |
|| read-write and read-create object | | | |
|| type shall define its behavior in | | | |
|| response to an agent reboot | | | |
+-----+-----+-----+-----+
| e. | Clarifications and additional | 2578 | 10.2p1 | o | o |
|| explanations shall be added to | | | |
|| the DESCRIPTION clause as needed | | | |
+-----+-----+-----+-----+
| f. | | | | |
+-----+-----+-----+-----+
| g. | The length of any binary string | 2578 | 7.9p7 | m | m |
|| appearing within a DEFVAL clause | | | |
|| shall be divisible by 8 | | | |
+-----+-----+-----+-----+
| h. | The length of any hexadecimal | 2578 | 7.9p7 | m | m |
|| string appearing within a DEFVAL | | | |
|| clause shall be divisible by 2 | | | |
+-----+-----+-----+-----+
| i. | Any character string appearing | 2578 | 7.9p7 | m | m |
|| within a DEFVAL clause shall not | | | |
|| include any tab or line | | | |
|| terminator characters | | | |
+-----+-----+-----+-----+
| j. | A DEFVAL clause shall be revised | 2578 | 10.2p1 | o[^19] | d |
|| as needed | | | |
+-----+-----+-----+-----+
| k. | The object identifier assigned to | 4181 | 4.6.5p1 | m | m |
|| an invocation of the OBJECT-TYPE | | | |
|| macro shall not end with a | | | |
|| sub-identifier of zero | | | |
+-----+-----+-----+-----+
| l. | The object identifier assigned to | 4181 | 4.6.5p1 | m | m |
|| an invocation of the OBJECT-TYPE | | | |
|| macro shall be unique | | | |
+-----+-----+-----+-----+

```

: General Rules for the OBJECT-TYPE Macro

Rules for Leaf Object Types

Leaf object types are the only object types that have instances that can be accessed via SNMP. Leaf object types include both scalar objects (i.e., with only one instance) and columnar objects (i.e., where there is one instance per conceptual row of a table). Table 10 provides additional requirements for the specification of leaf object types.

```

+-----+-----+-----+-----+

```


Item	Requirement	RFC	Clause	RFC	NTCIP\
	Status	Status			
+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+					
a.	The value of the SYNTAX clause	2578	7.1p1	c	c
	shall be a base type, a BITS				
	construct, or a textual				
	convention				
+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+					
b.	When a standard textual	4181	4.6.1.4p2	r	m
	convention provides the desired				
	semantics, it shall be used in				
	the SYNTAX clause for a new				
	object type				
+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+					
c.	When a standard textual	4181	4.6.1.4p2	r	m
	convention provides the desired				
	semantics and uses a consistent				
	syntax, it shall be used to				
	replace the SYNTAX clause in a				
	revised object type				
+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+					
d.	Object types with Boolean values	4181	4.6.1.9p2	r	r ^[^20]
	shall be assigned the SYNTAX of				
	the TruthValue textual				
	convention				
+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+					
e.					
+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+					
f.	Constraints shall be added to	4181	4.9p3	o	m
	the SYNTAX of an object type				
	when the restrictions are				
	implicit in the original				
	definition but not defined				
+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+					
g.	The SYNTAX of OBJECT IDENTIFIER	4181	4.6.1.5p2	r	m ^[^21]
	shall be used when the set of				
	identification values need to be				
	independently extensible without				
	a centralized registry				
+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+					
h.	The "Opaque" SYNTAX defined in	2578	7.1.9p4	m ^{^27^}	m ^{^27^}
	SMLv1 shall not be used				
+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+					
i.	The SYNTAX of IpAddress shall	4181	4.6.1.7p1	m ^[^23]	m ^{^27^}
	not be used ^[^22]				
+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+					
j.	The UNITS clause shall be	2578	72p1	o	r
	present for object types with				
	syntaxes that express values in				
	units (e.g., time, distance,				
	weight, volume)				
+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+					
k.	When possible, the text string		r		
	for the UNITS clause shall				
	conform to the International				
	System of Units (SI) as defined				

```

|| by the International Bureau of || || |
|| Weights and Measures (BIPM) || || |
+-----+-----+-----+-----+
| l. | The text string of the UNITS || | m | |
|| clause shall indicate the || || |
|| decimal position (e.g., || || |
|| "hundredths of seconds", || || |
|| "centiseconds") || || |
+-----+-----+-----+-----+
| m. | The IndexPart of the OBJECT-TYPE | 2578 | 7.7p1 | m | m |
|| macro shall not be included || || |
+-----+-----+-----+-----+
| n. | No nodes shall be defined | 2578 | 7.10p2 | m | m |
|| underneath an OBJECT IDENTIFIER || || |
|| that is associated with a leaf || || |
|| object type. || || |
+-----+-----+-----+-----+

```

: Rules for Leaf Object Types

Rules for Table Object Types

Table object types represent conceptual tables, which is the mechanism defined by SNMP to support multiple instances of management information. They are referred to as "conceptual" since an instance of the object type is not directly accessible; SNMP only allows access to cells within the table. Table 11 provides additional requirements for the specification of object types that represent a conceptual table.

```

+-----+-----+-----+-----+
| Item | Requirement | RFC | Clause | RFC | NTCIP\ |
| || || Status | Status |
+=====+=====+=====+=====+=====+=====+
| a. | The descriptor for a conceptual | 4181 | Cp1 | r | m |
|| table object type shall end with || || |
|| the term \"Table\" || || |
+-----+-----+-----+-----+
| b. | The SYNTAX shall be "SEQUENCE OF | 2578 | 7.1.12p1 | m | m |
|| \<EntryType>[^24] || || |
+-----+-----+-----+-----+
| c. | The MAX-ACCESS clause shall be | 2578 | 7.1.12p2 | m | m |
|| "not-accessible" || || |
+-----+-----+-----+-----+
| d. | The DESCRIPTION clause shall || | - | m |
|| define the purpose of the table || || |
|| and summarize its contents || || |
+-----+-----+-----+-----+
| e. | Static tables shall be associated || | m | |
|| with a scalar object type that || || |
|| indicates the number of rows in || || |
|| the table || || |
+-----+-----+-----+-----+
| f. | An object type that indicates the || | r |

```

```

|| number of rows in a table shall || || ||
|| have a SYNTAX with a lower limit || || ||
|| equal to or greater than one || || ||
|| (1)[^25] || || ||
+-----+
| g. | The IndexPart of the OBJECT-TYPE | 2578 | 7.7p1 | m | m |
|| macro shall not be included || || ||
+-----+
| h. | The OBJECT IDENTIFIER associated | 2578 | 7.10p2 | m | m | |
|| with a table object type shall || || ||
|| have exactly one sub-node || || ||
+-----+
| i. | The sub-node to the table shall | 2578 | 7.10p2 | m | m |
|| have a sub-identifier of 1 || || ||
+-----+
| j. | The sub-node to the table shall be | 2578 | 7.10p2 | m | m | |
|| the row object type that defines a || || ||
|| conceptual row within the table || || ||
+-----+

```

: Rules for Table Object Types

Rules for Row Object Types

Within SNMP, each conceptual table type has exactly one conceptual row type. Similar to the conceptual table, SNMP does not allow direct access to an instance of a row, which is why it is referred to as "conceptual". Row object types identify the columns contained within each conceptual row instance of the table and provides information about managing the information within a row. Table 12 provides additional requirements for the specification of object types that represent a conceptual row within a table.

```

+-----+
| Item | Requirement | RFC | Clause | RFC | NTCIP\ |
| || || Status | Status |
+=====+=====+=====+=====+=====+=====+
| a. | The descriptor for a conceptual | 4181 | Cp1 | r | m | |
|| row object type shall be the || || ||
|| descriptor for the conceptual || || ||
|| table object type with the || || ||
|| suffix \"Table\" replaced with || || ||
|| the suffix \"Entry\" || || ||
+-----+
| b. | The SYNTAX shall be | 2578 | 7.1.12p1 | m | m |
|| \"<EntryType>\"^28^ || || ||
+-----+
| c. | The descriptor used for | 4181 | Cp1 | r | m | |
|| \"<EntryType>\" shall be || || ||
|| identical to the descriptor for || || ||
|| the conceptual row, except the || || ||
|| first letter will be in upper || || ||
|| case || || ||
+-----+

```

| d. | A revision shall add columns to | | | - | o[^26] |
 | | the "<EntryType>" as needed | | | |
 +-----+-----+-----+-----+
 | e. | The MAX-ACCESS clause shall be | 2578 | 7.1.12p2 | m | m |
 | | "not-accessible" | | | |
 +-----+-----+-----+-----+
f.	The DESCRIPTION clause shall	2578	7.7p8	m	m
	explain how any INDEX objects				
	that are not from the table are				
	used to uniquely identify				
	instances of objects				
+-----+-----+-----+-----+					
g.	The IndexPart of the OBJECT-TYPE	2578	7.7p1	m	m
	macro shall be included				
+-----+-----+-----+-----+					
h.	The IndexPart shall use the	2578	7.8.1p1	m	m
	AUGMENTS clause if the table has				
	a one-to-one correspondence to				
	the conceptual rows of an				
	existing table[^27]				
+-----+-----+-----+-----+					
i.	The IndexPart shall use the	2578	7.8.1p1	m	m
	INDEX clause if the table does				
	not have a one-to-one				
	correspondence to the conceptual				
	rows of an existing table				
+-----+-----+-----+-----+					
j.	The INDEX clause of a conceptual	4181	4.6.4p1	r	m
	row that extends another table				
	with a row relationship that is				
	not one-to-one[^28] shall use an				
	INDEX clause where the initial				
	indices are the indices of the				
	parent table listed in the same				
	order.				
+-----+-----+-----+-----+					
k.	The INDEX clause shall specify	2578	7.7p2	m	m
	object types that allow				
	unambiguous identification of a				
	conceptual row				
+-----+-----+-----+-----+					
l.	The INDEX clause shall not	2578	7.7p2	m	m
	reference scalar objects				
+-----+-----+-----+-----+					
m.	The INDEX clause shall not	2578	7.7p2	r	m
	reference the same object type				
	more than once				
+-----+-----+-----+-----+					
n.					
+-----+-----+-----+-----+					
o.	If the INDEX clause references	4181	4.6.4p1	m	m
	an object type that is external				
	to the table, the Entry shall				
	identify which rows will exist				
	within the local table				
 +-----+-----+-----+-----+

p.	The INDEX clause shall not	2578	7.7p5	m	m
	reference object types that have				
	a SYNTAX of Counter32 or				
	Counter64				
+-----+					
q.	The INDEX clause shall not	4181	4.6.1.1p3	r	r
	reference an object type that				
	allows the value zero				
+-----+					
r.	The INDEX clause shall not	4181	4.6.1.1p3	m	m
	reference an object type that				
	allows negative values				
+-----+					
s.	The IMPLIED keyword shall not be	2578	7.7p4	m	m
	used with an object type having				
	a fixed length				
+-----+					
t.	The IMPLIED keyword shall not be	2578	7.7p4	m	m
	used with any object type other				
	than the last object type listed				
	in the INDEX clause				
+-----+					
u.	The IMPLIED keyword shall not be	2578	7.7p4	m	m
	used with an object type of				
	variable length string that				
	might have zero-length				
+-----+					
v.	The IMPLIED keyword shall not be	4181	4.6.4p1	r	m
	used within an INDEX clause for				
	a conceptual row that might be				
	expanded (i.e., extended with				
	additional index columns)				
+-----+					
w.	The MAX-ACCESS clause of each	2578	7.7p7	m[^29]	m^33^
	object type listed in the INDEX				
	clause of its same table shall				
	be "not-accessible"				
+-----+					
x.	The SYNTAX and number of object	4181	4.6.6p2	r	m
	types listed in the INDEX clause				
	shall be designed to ensure that				
	the OBJECT IDENTIFIERS of				
	columnar object instances do not				
	exceed 128 sub-identifiers				
+-----+					
y.	Each subordinate columnar object	2578	7.10p2	m	m
	shall be assigned an OBJECT				
	IDENTIFIER that is a direct				
	sub-identifier of the conceptual				
	row.				
+-----+					
z.	Each subordinate columnar object	4181	Cp1	r	m
	shall be assigned a descriptor				
	that starts with the conceptual				
	row descriptor minus the				
	\"Entry\" suffix				

: Rules for Row Object Types

Rules for Tables that Support Creation/Deletion of Rows

Some conceptual tables allow for row creation and deletion through SNMP operations or internally by the agent. These operations require proper management to ensure that the data within them do not produce logical inconsistencies. Table 13 provides requirements specific to conceptual tables that allow for row creation or deletion internally or by an SNMP operation (i.e., dynamic and managed tables). Table 14 provides requirements specific to conceptual tables that allow for row creation or deletion by an SNMP operation.

Item	Requirement	RFC	Clause	RFC	NTCIP\
	Status	Status			
a.	The DESCRIPTION clause of the	4181	4.6.4p1	m	m
	conceptual row shall define any				
	conditions under which an <u>agent</u>				
	can independently create/delete				
	rows				
b.	The DESCRIPTION clause of the	4181	4.6.4p1	o.1	o.1
	conceptual row shall define what				
	happens to dynamically created				
	rows upon reboot				
c.					

: Rules for Tables that Support Creation/Deletion of Rows

Table 14: Rules for Tables that Support Creation/Deletion of Rows

Item	Requirement	RFC	Clause	RFC	NTCIP\
	Status	Status			
d.	The conceptual row shall contain a	4181	4.6.4p1	o.1	o.1
	columnar object type with the				
	SYNTAX of StorageType				
e.	The StorageType object type, if	4181	4.6.4p1	m	m
	present, shall have a MAX-ACCESS				
	of read-create				
f.	The DESCRIPTION clause of the	4181	4.6.4p1	m	m
	StorageType object type, if				
	present, shall specify which				
	columns within a permanent row can				
	be modified				

+-----+ | g. | The conceptual row shall contain a | 2578 | 7.1.12.1p1 | m | m | | | columnar object type with the | | | | | SYNTAX of RowStatus | | | | +-----+-----+-----+-----+
+-----+-----+ | h. | The RowStatus object type shall | 2578 | 7.1.12.1p1 | m | m | | | have a MAX-ACCESS of read-create | | | | +-----+-----+-----+-----+-----+-----+ | i. | No object type contained within | 2578 | 7.3p4 | m | m | | | the conceptual row shall have a | | | | | MAX-ACCESS of "read-write" [^30] | | | | +-----+-----+-----+-----+-----+-----+
| j. | The DESCRIPTION clause of the | 4181 | 4.6.4p1 | m | m | | | RowStatus object type shall | | | | |
| | specify the columnar objects that | | | | | must be set with valid values | | | | | before row activation | | | | +-----+-----+-----+-----+-----+ | k. | The DESCRIPTION clause of the | 4181 | 4.6.4p1 | m | m | | | RowStatus object type shall | | | | | specify whether it is possible to | | | | | modify other columns when the | | | | | RowStatus is active | | | | +-----+-----+-----+-----+-----+-----+

Rule for the Syntax Specification of the OBJECT-TYPE Macro

General Rules for SYNTAX Specification

Table 15 provides requirements for the specification of the SYNTAX clause for leaf object types.

+-----+-----+-----+-----+-----+				
Item	Requirement	RFC	Clause	RFC NTCIP\
	Status	Status		
+=====+=====+=====+=====+=====+=====+=====+=====+=====+=====+				
a.	An <u>object type</u> that is	4181	4.6.1.1p1	m m
	represented as an integer-valued			
	enumeration shall have a SYNTAX			
	of "INTEGER" with a			
	NamedNumberList			
+-----+-----+-----+-----+-----+				
b.	An <u>object</u> with a SYNTAX of an		r	
	enumerated INTEGER that might			
	conceivably be extended shall			
	assign value 1 to \'other\' to			
	allow manufacturer-specific			
	objects to extend the range[^31]			
+-----+-----+-----+-----+-----+				
c.	Except for enumerations, the	4181	4.6.1.1p3	r m[^32]
	keyword \'INTEGER\' shall not be			
	used			
+-----+-----+-----+-----+-----+				
d.	An <u>object type</u> that is	4181	4.6.1.1p3	r m
	represented as an integer, needs			
	to support negative values, and			
	can be represented within the			
	range of a signed 32-bit integer			
	shall have a SYNTAX based on			
	Integer32			
+-----+-----+-----+-----+-----+				
e.	An <u>object type</u> whose actual	4181	4.6.1.1p3	r m

```

|| value 1) can increase above the ||||
|| maximum reported value or ||||
|| decrease below the minimum ||||
|| reported value, 2) does not ||||
|| latch at the extreme value when ||||
|| the value normalizes, and 3) ||||
|| should be represented within the ||||
|| range of an unsigned 32-bit ||||
|| integer shall have a SYNTAX ||||
|| based on Gauge32 ||||
+-----+
| f. | An object type whose actual | 4181 | 4.6.1.3p2 | d[^33] | d |
|| value 1) can increase above the ||||
|| maximum value or decrease below ||||
|| the minimum value, 2) does not ||||
|| latch at the extreme value when ||||
|| the value normalizes, and 3) ||||
|| should be represented within the ||||
|| range of an unsigned 64-bit ||||
|| integer shall have a SYNTAX ||||
|| based on CounterBasedGauge64 ||||
+-----+
| g. | An object type that is | 2578 | 7.1.6p1 | m | m |
|| represented as an integer that ||||
|| is intended to monotonically ||||
|| increase until a maximum value ||||
|| and then wrap shall have a ||||
|| SYNTAX based on an accepted ||||
|| counter type (See Section ||||
|| 4.4.4.3) ||||
+-----+
| h. | An object type that represents a | 2578 | 7.1.8p1 | r | m |
|| duration between two epochs and ||||
|| measured in hundredths of a ||||
|| second shall have a SYNTAX based ||||
|| on TimeTicks ||||
+-----+
| i. | An object type that is | 4181 | 4.6.1.1p3 | r | r[^34] |
|| represented as an integer; does ||||
|| not have the semantics of a ||||
|| counter, gauge, or TimeTicks; ||||
|| and can be represented within ||||
|| the range of an unsigned, 32-bit ||||
|| integer shall have a SYNTAX ||||
|| based on Unsigned32 ||||
+-----+
| j. | An object type that is | 4181 | 4.6.1.1p3 | m | m |
|| represented as an integer; does ||||
|| not have the semantics of a ||||
|| counter, gauge, or TimeTicks; ||||
|| and supports values greater than ||||
|| 2^31-1 shall have a SYNTAX ||||
|| based on Unsigned32 ||||
+-----+
| k. | An object type with a syntax ||| - | r |
|| dealing with time shall use a ||||

```



```

|| textual convention defined in |||
|| ISO 20684-1 (e.g., |||
|| ITSDailyTimeStamp) |||
+-----+
| l. | An object type that represents | 2578 | 3.6p1 | r | m |
|| an object identifier shall have |||
|| a SYNTAX based on OBJECT |||
|| IDENTIFIER[^35] |||
+-----+
| m. | An object type that represents a | 2578 | 7.1.4p1 | r | r[^36] |
|| series of named items where each |||
|| item can have a Boolean |||
|| representation shall have a |||
|| SYNTAX based on the BITS |||
|| construct |||
+-----+
| n. | Any generic network/transport ||| m |
|| address shall be represented as |||
|| a pair of object types using 1) |||
|| a TransportDomain or |||
|| TransportAddressType and 2) a |||
|| TransportAddress from RFC 3419 |||
+-----+
| o. | An IP-specific network address | 4181 | 4.6.1.7p1 | r | r |
|| shall be represented with the |||
|| InetAddressType and InetAddress |||
|| pair from RFC 4001 |||
+-----+
| p. | All other object types shall | 2578 | 7.1.2p1 | r | r |
|| have a SYNTAX based on OCTET |||
|| STRING |||
+-----+
| q. | Object types shall not have a | 4181 | Bp3 | m | m |
|| SYNTAX of DisplayString unless |||
|| its values are intended to be |||
|| explicitly limited to NVT ASCII |||
+-----+
| r. | The SYNTAX shall be constrained | 4181 | 4.6.1.4p1 | r | m |
|| using range or SIZE constraints |||
|| as appropriate[^37] |||
+-----+
| s. | If the SYNTAX value references a | 2578 | 11.2p1 | m | m |
|| textual convention, a |||
|| (potentially additional) |||
|| constraint shall be allowed to |||
|| further reflect the appropriate |||
|| value |||
+-----+
| t. | The constraints on the SYNTAX of ||| m |
|| a standardized object type shall |||
|| not be revised if such a |||
|| revision changes the meaning or |||
|| validity of any value[^38] |||
+-----+
| u. |||
+-----+

```

```

| v. | All values that are outside the ||| m |
|| standardized named bits and |||
|| enumerations are reserved by the |||
|| standard[^39] |||
+-----+
| w. | If the SYNTAX of a standardized ||| m |
|| object type is revised (which is |||
|| only allowed in a backwards |||
|| compatible manner per other |||
|| rules), an explanation shall be |||
|| added to the "<Informative>" |||
|| subclause of the object type |||
+-----+
| x. | If the SYNTAX of a standardized | 4181 | 4.6.1.10p3 | m | m |
|| object type is revised and the |||
|| object type is writeable or is |||
|| used as an index to a table that |||
|| supports creation and/or |||
|| deletion of rows, the module's |||
|| compliance statement shall |||
|| indicate the values that must be |||
|| supported |||
+-----+

```

: General Rules for SYNTAX Specification

Rules for Enumerations

Table 16 provides additional requirements for the specification of leaf object types that are based on an INTEGER with a NamedNumberList.

```

+-----+
| Item | Requirement | RFC | Clause | RFC | NTCIP\ |
| ||| | Status | Status |
+-----+-----+-----+-----+-----+-----+
| a. | The NamedNumberList shall | 2578 | 7.1.1p2 | m | m |
|| specify all allowed values[^40] |||
+-----+
| b. | The NamedNumberList shall start | 2578 | 7.1.1p2 | r | r |
|| with the enumeration value of 1 |||
+-----+
| c. | The NamedNumberList shall use | 2578 | 7.1.1p2 | r | r |
|| contiguously assigned values |||
+-----+
| d. | The identifier of a named number | 2578 | 7.1.1p3 | m | m |
|| shall not exceed 64 characters |||
|| in length |||
+-----+
| e. | The identifier of a named number | 2578 | 7.1.1p3 | o[^41] | o |
|| shall not exceed 32 characters |||
|| in length |||
+-----+
| f. | New enumerations shall be added | 2578 | 10.2p1 | o | o |
|| during a revision |||

```

```

+-----+-----+-----+-----+
| g. | An identifier for a standardized | 2578 | 10.2p1 | m | r[^42] | |
|| named number shall not be || || ||
|| changed || || ||
+-----+-----+-----+-----+
| h. | A standardized named number | - | - | - | o | |
|| shall be marked via a comment as || || ||
|| "deprecated" or "obsolete", as || || ||
|| appropriate. || || ||
+-----+-----+-----+-----+
| i. | The DEFVAL clause, if present, | 4181 | 4.6.1.1p3 | r | m | |
|| shall use the identifier of the || || ||
|| named number rather than the || || ||
|| equivalent integer || || ||
+-----+-----+-----+-----+

```

: Rules for Enumerations

Rule for Counters

Table 17 provides additional requirements for the specification of leaf object types that are based on a counter.

```

+-----+-----+-----+-----+
| Item | Requirement | RFC | Clause | RFC | NTCIP\ |
|| || Status | Status |
+=====+=====+=====+=====+=====+=====+
| a. | The valuereference assigned to | 2578 | 3.1p5 | m | m | |
|| the counter object type shall || || ||
|| denote plurality || || ||
+-----+-----+-----+-----+
| b. | If the information is ever likely | 2578 | 7.1.10p4 | r | r | |
|| to wrap in less than an hour, the || || ||
|| SYNTAX shall be based on a 64-bit || || ||
|| counter syntax || || ||
+-----+-----+-----+-----+
| c. | Counters that satisfy the | 4181 | 4.6.1.2p1 | r | m | |
|| semantics of Counter32/Counter64 || || ||
|| as defined in RFC 2578 Clause || || ||
|| 7.1.6 (e.g., have no defined || || ||
|| initial value and have no || || ||
|| requirements to reset) shall have || || ||
|| a SYNTAX of Counter32 or || || ||
|| Counter64, as appropriate || || ||
+-----+-----+-----+-----+
| d. | Counters that satisfy the | 4181 | 4.6.1.2p1 | r | m | |
|| semantics of ZeroBasedCounter32/ || || ||
|| ZeroBasedCounter64 (e.g., || || ||
|| initialize at zero and have no || || ||
|| requirements to reset) shall have || || ||
|| a SYNTAX of ZeroBasedCounter32 or || || ||
|| ZeroBasedCounter64, as || || ||
|| appropriate || || ||
+-----+-----+-----+-----+

```

```

| e. | Counters that satisfy the ||| | r |
|| semantics of ITSCounter32 (e.g., ||| |
|| initialize at zero and can be ||| |
|| reset) shall have a SYNTAX of ||| |
|| ITSCounter32 ||| |
+-----+
| f. | Counters that do not satisfy the | 4181 | 4.6.1.3p2 | m | m |
|| semantics of the other counter ||| |
|| types shall use another textual ||| |
|| convention that explicitly ||| |
|| defines the intended ||| |
|| semantics.[^43] ||| |
+-----+
| g. | The MAX-ACCESS clause shall have | 2578 | 7.1.6p3 | m | m |
|| a value of "read-only" or ||| |
|| "accessible-for-notify" ||| |
+-----+
| h. | For Counter32 and Counter64 | 2578 | 7.1.6p2 | m | m |
|| object types, the DESCRIPTION ||| |
|| clause shall not define an ||| |
|| initial value ||| |
+-----+
| i. | For Counter32, Counter64, | 4181 | 4.6.1.2p1 | m | m |
|| ZeroBasedCounter32, and ||| |
|| ZeroBasedCounter64 object types, ||| |
|| the DESCRIPTION clause shall not ||| |
|| require the counter to reset in ||| |
|| response to any event ||| |
+-----+
| j. | The DESCRIPTION clause shall | 4181 | 4.6.1.2p1 | m | m |
|| identify any events that might ||| |
|| cause the counter to have a ||| |
|| discontinuity and how these ||| |
|| events can be detected ||| |
+-----+
| k. | If discontinuities can occur at | 2578 | 7.1.6p2 | r | m |
|| any time other than ||| |
|| re-initialization of the device, ||| |
|| the MIB module shall define a ||| |
|| discontinuity object type that ||| |
|| provides a reference as to when ||| |
|| the last discontinuity occurred ||| |
+-----+
| l. | The object type shall not have a | 2578 | 7.9p8 | m | m |
|| DEFVAL clause ||| |
+-----+

```

: Rules for Counters

Rule for TimeTicks

Table 18 provides additional requirements for the specification of leaf object types that are based on the TimeTicks syntax.

```

+-----+-----+-----+-----+-----+
| Item | Requirement | RFC | Clause | RFC | NTCIP\ |
| || || Status | Status |
+-----+-----+-----+-----+-----+-----+
| a. | The DESCRIPTION clause shall | 2578 | 7.1.8p1 | m | m |
| | define the two reference epochs | || ||
| | for an object type with a SYNTAX | || ||
| | of TimeTicks | || ||
+-----+-----+-----+-----+-----+

```

: Rules for TimeTicks

Rule for OIDs

Table 19 provides additional requirements for the specification of leaf object types that are based on the OBJECT IDENTIFIER syntax.

```

+-----+-----+-----+-----+-----+
| Item | Requirement | RFC | Clause | RFC | NTCIP\ |
| || || Status | Status |
+-----+-----+-----+-----+-----+-----+
| a. | OBJECT IDENTIFIER values shall be | 2578 | 7.1.3p1 | m | m |
| | designed to limit their structure to | || ||
| | 128 sub-identifiers | || ||
+-----+-----+-----+-----+-----+
| b. | OBJECT IDENTIFIER values shall be | 2578 | 7.1.3p1 | m | m |
| | designed to limit each | || ||
| | sub-identifier value to less than | || ||
| | 2^32 | || ||
+-----+-----+-----+-----+-----+
| c. | The SYNTAX of an object type that is | 4181 | 4.6.1.5 | r | m |
| | intended to refer to an object | || ||
| | instance shall use the | || ||
| | VariablePointer textual convention | || ||
| | defined in RFC 2579 | || ||
+-----+-----+-----+-----+-----+
| d. | The SYNTAX of an object type that is | 4181 | 4.6.1.5 | r | m |
| | intended to refer to a row within a | || ||
| | conceptual table shall use the | || ||
| | RowPointer textual convention | || ||
| | defined in RFC 2579 | || ||
+-----+-----+-----+-----+-----+
| e. | The SYNTAX of an object type that is | - | - | - | - |
| | intended to refer to any other | || ||
| | entity registered on the | || ||
| | international object identifier tree | || ||
| | shall use the AutonomousType textual | || ||
| | convention defined in RFC 2579 | || ||
+-----+-----+-----+-----+-----+
| f. | An object type that is designed to | - | - | - | m |
| | reference and access a value stored | || ||
| | in another object instance shall be | || ||
| | associated with security credentials | || ||
| | that are to be used when accessing | || ||

```

|| that data. || || ||

+-----+-----+-----+-----+

| g. | The DEFVAL clause shall use a single | 2578 | 7.9p4 | m | m |

|| X.208 ASN.1 identifier || || ||

+-----+-----+-----+-----+

: Rules for OIDs

Rule for BITS

Table 20 provides additional requirements for the specification of leaf object types that are based on the BITS construct.

+-----+-----+-----+-----+

| Item | Requirement | RFC | Clause | RFC | NTCIP\ |

|| || | Status | Status |

+=====+=====+=====+=====+=====+=====+=====+=====+

| a. | A SYNTAX based on the BITS | 2578 | 7.1.4p1 | m | m |

|| construct shall specify all || || ||

|| allowed values^[44] || || ||

+-----+-----+-----+-----+

| b. | The named bits in the BITS | 4181 | 4.6.1.6p1 | m | m |

|| construct shall start at 0 || || ||

+-----+-----+-----+-----+

| c. | The named bits shall be assigned | 2578 | 7.1.4p2 | r^[45] | r⁴⁹ |

|| to contiguous bit positions || || ||

+-----+-----+-----+-----+

| d. | The identifier of a named bit | 2578 | 7.1.4p4 | m | m |

|| shall not exceed 64 characters || || ||

|| in length || || ||

+-----+-----+-----+-----+

| e. | The identifier of a named bit | 2578 | 7.1.4p4 | o^[46] | o |

|| shall not exceed 32 characters || || ||

|| in length || || ||

+-----+-----+-----+-----+

| f. | New bit assignments shall be | 2578 | 7.1.4p2 | o | o |

|| added to existing BITS || || ||

|| constructs during a revision || || ||

+-----+-----+-----+-----+

| g. | Existing bit assignments shall || || o |

|| be aged as appropriate during a || || ||

|| revision (e.g., denoted with a || || ||

|| \"- deprecated\" statement) || || ||

+-----+-----+-----+-----+

| h. | An identifier for a standardized | 2578 | 7.1.4p2 | m | m |

|| named bit shall not be changed || || ||

: Rules for BITS

Rule for Block Objects

Table 21 provides additional requirements for the specification of leaf object types that are based on the OCTET STRING syntax.

Item	Requirement	RFC	Clause	RFC	NTCIP\
	Status	Status			
a.	The descriptor for a block <u>object</u> m				
	shall have a suffix of \"Block\"				
b.	Block <u>object</u> types shall have a r[^47]				
	SYNTAX of ITSOerString as defined				
	by ISO 20684-1				
c.	Block <u>object</u> types shall define - r				
	the structure of their embedded				
	data using the format defined in				
	Clause 4.5.4.2				
d.	Values of block <u>object</u> instances - r				
	shall be encoded using the Octet				
	Encoding Rules (OER) as defined in				
	ITU-T X.696 (ISO/IEC 8825-7)				
e.	The MAX-ACCESS of a block <u>object</u> - m				
	type shall not be greater than the				
	MAX-ACCESS of any <u>object</u> type				
	referenced by the block <u>object</u>				
	type data structure[^48]				

: Rules for Block Objects

Rule for Octet Strings

Table 22 provides additional requirements for the specification of leaf object types that are based on the OCTET STRING syntax.

Item	Requirement	RFC	Clause	RFC	NTCIP\
	Status	Status			
a.	SIZE constraints shall be defined 4181 4.6.1.4p1 r m				
	for OCTET STRINGS if they are				
	more limited than the maximum of				
	0..65535				

```

+-----+-----+-----+-----+
| b. | OCTET STRINGS shall be | 2578 | 7.1.2p1 | o | o |
|| constrained to 255 octets ||||
+-----+-----+-----+-----+
| c. | The size of any OCTET STRING used | 4181 | 4.6.1.4p1 | r | m |
|| as an index shall be size ||||
|| constrained to facilitate object ||||
|| instance identification ||||
+-----+-----+-----+-----+

```

: Rules for Octet Strings

Rule for the DESCRIPTION Clause of the OBJECT-TYPE Macro

The DESCRIPTION clause consists of a text string intended to define the object type. NTCIP extends this definition by defining a series of subclauses to be embedded within this text string with precise rules for each subfield as defined in Table 23.

```

+-----+-----+-----+-----+
| Item | Requirement | RFC | Clause | RFC | NTCIP\ |
|||| Status | Status |
+=====+=====+=====+=====+=====+=====+
| a. | The DESCRIPTION clause shall be ||| m |
|| divided into the following ||||
|| subclauses, each with their own ||||
|| label as indicated: ||||
||||||
|| \<Definition> ||||
||||||
|| \<Format> ||||
||||||
|| \<Superseded by> ||||
||||||
|| \<Informative> ||||
||||||
|| \<Object Identifier> ||||
+-----+-----+-----+-----+
| b. | The \<Definition> subclause shall ||| m |
|| be present for all object types. ||||
+-----+-----+-----+-----+
| c. | The \<Definition> subclause shall ||| m |
|| contain the normative definition ||||
|| of the object type ||||
+-----+-----+-----+-----+
| d. | The \<Definition> subclause shall ||| r |
|| not be revised, once standardized, ||||
|| except to fix typos ||||
+-----+-----+-----+-----+
| e. | The \<Format> subclause shall be ||| o |
|| used to define additional ||||
|| semantics for specific values in ||||
|| object types (e.g., a special ||||
|| meaning of the value 0)[^49] ||||

```



```

+-----+
| f. | The \<Format> subclause shall be ||| m |
|| present for leaf object types that |||
|| use a bitmaps or enumerations for |||
|| the representation of data |||
+-----+
| g. | For bitmapped formats, the ||| m |
|| \<Format> subclause shall |||
|| normatively define the format per |||
|| Clause 4.5.3 |||
+-----+
| h. | The \<Format> subclause shall not ||| r |
|| be revised, once standardized, |||
|| except to fix typos or to change |||
|| the layout of the information |||
+-----+
| i. |||
+-----+
| j. |||
+-----+
| k. |||
+-----+
| l. | The \<Superseded by> subclause ||| o |
|| shall be present for any object |||
|| type that has a STATUS of |||
|| deprecated or obsolete |||
+-----+
| m. | The \<Superseded by> subclause ||| m |
|| shall contain a reference to one |||
|| or more object types that are |||
|| intended to replace the |||
|| functionality of the containing |||
|| object type or shall indicate that |||
|| the object was not replaced |||
+-----+
| n. | The \<Superseded by> subclause ||| r |
|| shall not be revised once |||
|| standardized[^50] |||
+-----+
| o. | The \<Informative> subclause ||| o |
|| shall be present for all object |||
|| types |||
+-----+
| p. | The \<Informative> subclause ||| m |
|| shall contain any informative |||
|| information about the object type |||
|| that the authors might deem to be |||
|| useful for implementors and other |||
|| users |||
+-----+
| q. | The \<Informative> subclause ||| o |
|| shall be revised as appropriate to |||
|| provide further clarification of |||
|| the meaning of the object type |||
+-----+
| r. | If the STATUS of the object type ||| m |

```

```

|| is aged (e.g., deprecated or made || || |
|| obsolete), the \<Informative> || || |
|| subclause shall explain why it was || || |
|| aged[^51] || || |
+-----+
| s. | If the object type is a || || | r |
|| replacement for a deprecated || || |
|| object type, the \<Informative> || || |
|| subclause shall identify the || || |
|| deprecated object type and explain || || |
|| how the issue was addressed || || |
+-----+
| t. | The \<Object Identifier> || || | m |
|| subclause shall be present for all || || |
|| object types || || |
+-----+
| u. | The \<Object Identifier> || || | m |
|| subclause shall provide an || || |
|| informative, human-readable || || |
|| version of the full object || || |
|| identifier in integer dot notation || || |
|| from the root node || || |
+-----+
| v. | The \<Object Identifier> || || | m |
|| subclause shall not be revised, || || |
|| once standardized, except to fix || || |
|| typos || || |
+-----+

```

: Rules for the DESCRIPTION Clause

Mapping of the NOTIFICATION-TYPE macro

Object types are typically exchanged via get and set operations that are initiated by a manager; however, SNMP also allows for the definition of notification types, which are notices initiated by a device and sent to a manager. Notification types can be sent in either an unacknowledged mode (called a "trap") or in an acknowledged mode (called an "inform"); both mechanisms rely upon the NOTIFICATION-TYPE macro to define the content and meaning of each notification type.^[52]

```

+-----+
| Item | Requirement | RFC | Clause | RFC | NTCIP\ |
| || || | Status | Status |
+=====+=====+=====+=====+=====+
| a. | Each notification type shall be | 2578 | 8p1 | m | m |
|| specified using an instance of the || || |
|| NOTIFICATION-TYPE macro || || |
+-----+
| b. | The OBJECTS clause shall not | 2578 | 8.1p1 | m | m |
|| include any object types that have || || |
|| a MAX-ACCESS clause value of || || |
|| "not-accessible" || || |

```

```

+-----+
| c. | The value assigned in the STATUS | 2578 | 10.3p1b | r | r |
|| clause shall not be updated to || || |
|| restore an items status (e.g., do || || |
|| not change "deprecated" to || || |
|| "current") || || |
+-----+
| d. | All normative text about a | 2578 | 8.3p1 | m | m |
|| notification type shall be included || || |
|| within its DESCRIPTION clause (but || || |
|| this may include references to || || |
|| other text) || || |
+-----+
| e. | The DESCRIPTION clause shall | 2578 | 8.1p1 | m | m |
|| specify the meaning conveyed by || || |
|| each occurrence of each object type || || |
|| in the OBJECTS clause || || |
+-----+
| f. | The DESCRIPTION clause shall | 2578 | 8.1p1 | m | m |
|| specify which instance should be || || |
|| included for each occurrence of || || |
|| each object type in the OBJECTS || || |
|| clause || || |
+-----+
| g. | The DESCRIPTION clause shall | 4181 | 4.7p3 | m | m |
|| specify or otherwise reference the || || |
|| throttling technique to be used for || || |
|| the notification[^53] || || |
+-----+
| h. | Clarifications and additional | 2578 | 10.3p1c | o | o |
|| explanations shall be added to the || || |
|| DESCRIPTION clause as needed || || |
+-----+
| i. | The next-to-last sub-identifier of | 2578 | 8.5p1 | m | m |
|| the OID for any newly defined || || |
|| notification type shall have a || || |
|| value of zero || || |
+-----+
| j. | An OBJECT IDENTIFIER that is | 4181 | 4.6.5p2 | r | m |
|| assigned to notification type shall || || |
|| not have any sub-identifiers || || |
|| defined. || || |
+-----+

```

Mapping of the NOTIFICATION-TYPE macro

Mapping of the Textual Convention Macro

Section 3 and 4 of [RFC 2579](#) defines requirements related developing a new textual convention macro,

```

+-----+
| Item | Requirement | RFC | Clause | RFC | NTCIP\ |
| || || | Status | Status |

```

```

+====+=====+====+:====+:====+:====+:====+:
| a. | Textual conventions shall be defined ||| m |
|| for any SYNTAX that is used ||||
|| repeatedly with similar semantics ||||
+-----+-----+-----+-----+
| b. | Textual conventions shall be defined ||| m |
|| for any SYNTAX that would benefit ||||
|| from being associated with a ||||
|| DISPLAY-HINT ||||
+-----+-----+-----+-----+
| c. | The descriptor for the textual | 2579 | 3p2 | m | m |
|| convention shall not conflict with ||||
|| reserved words ||||
+-----+-----+-----+-----+
| d. | The descriptor shall not be in all | 2579 | 3p2 | m | m |
|| upper case ||||
+-----+-----+-----+-----+
| e. | The descriptor shall start with ||| m |
|| \"Ntcip\" to ensure uniqueness from ||||
|| any existing or future textual ||||
|| conventions from RFCs or other ||||
|| sources ||||
+-----+-----+-----+-----+
| f. | The DISPLAY-HINT clause shall not be | 2579 | 3.1p1 | m | m |
|| defined for any textual convention ||||
|| with a SYNTAX clause value of OBJECT ||||
|| IDENTIFIER, IpAddress, Counter32, ||||
|| Counter64, or any enumerated integer ||||
|| or BITS construct syntax ||||
+-----+-----+-----+-----+
| g. | The DISPLAY-HINT clause shall be | 4181 | 4.6.3p2 | r | r |
|| defined for all other values of the ||||
|| SYNTAX clause ||||
+-----+-----+-----+-----+
| h. | The value assigned in the STATUS | 2579 | 5p2 | o | o |
|| clause shall not be updated to ||||
|| restore an item's status (e.g., do ||||
|| not change "deprecated" to "current") ||||
+-----+-----+-----+-----+
| i. | The SYNTAX clause of a textual | 2579 | 3.5p1 | m | m |
|| convention shall not refer to another ||||
|| textual convention ||||
+-----+-----+-----+-----+
| j. | The SYNTAX clause of a textual | 2579 | 4p1 | o | o |
|| convention shall be constrained ||||
|| (i.e., sub-typed) ||||
+-----+-----+-----+-----+
| k. | ASN.1 type assignments (e.g., | 2579 | 5p2 | m | m |
|| \<MyType>::=\<Type>) shall not be ||||
|| used[^54] ||||

```

: Mapping the Textual Convention macro

Mapping of the OBJECT-GROUP macro

Section 3 of [RFC 2580](#) defines requirements related to the OBJECT-GROUP macro.

```

+-----+-----+-----+-----+
| Item | Requirement | RFC | Clause | RFC | NTCIP\ |
| | | | Status | Status |
+=====+=====+=====+=====+=====+=====+
| a. | The descriptor for an OBJECT-GROUP | | | r |
| | shall include a revision number | | | |
| | ("R#") | | | |
+-----+-----+-----+-----+
| b. | The OBJECTS clause shall not contain | 2580 | 3.1p1a | m | m |
| | object types from other modules | | | |
+-----+-----+-----+-----+
| c. | The OBJECTS clause shall not contain | 2580 | 3.1p1b | m | m |
| | object types with MAX-ACCESS of | | | |
| | "not-accessible" | | | |
+-----+-----+-----+-----+
| d. | A MIB module shall not define any | 2580 | 3.1p2 | m | m |
| | accessible object types that are not | | | |
| | referenced by the OBJECTS clause of | | | |
| | at least one object group | | | |
+-----+-----+-----+-----+
| e. | A revision to a standardized | 4181 | 4.9p6a | m | m |
| | OBJECT-GROUP macro shall not add or | | | |
| | delete an object from the OBJECTS | | | |
| | clause | | | |
+-----+-----+-----+-----+
| f. | The value assigned in the STATUS | 2580 | 7.1p2a | o | o |
| | clause shall not be updated to | | | |
| | restore an items status (e.g., do | | | |
| | not change "deprecated" to | | | |
| | "current") | | | |
+-----+-----+-----+-----+
| g. | The STATUS clause of the | 2580 | 7.1p2f | o[^55] | o |
| | OBJECT-GROUP shall change when the | | | |
| | status of a referenced object type | | | |
| | changes | | | |
+-----+-----+-----+-----+
| h. | The DESCRIPTION clause shall | | | r |
| | identify the standard and version | | | |
| | where the group was originally | | | |
| | defined[^56] | | | |

```

: OBJECT-GROUP mapping

Mapping of the NOTIFICATION-GROUP macro

Section 3 of [RFC 2580](#) defines requirements related to the NOTIFICATION -GROUP macro.

Item	Requirement	RFC	Clause	RFC	NTCIP\	Status	Status
a.	The NOTIFICATIONS clause shall not contain notification types from other modules	2580	4.1p1	m	m		
b.	A MIB module shall not define notification types that are not referenced by the NOTIFICATIONS clause of at least one notification group	2580	4.1p2	m	m		
c.	A revision to a standardized NOTIFICATION-GROUP macro shall not add or delete a NOTIFICATION from the NOTIFICATIONS clause	4181	4.9p6a	x	x		
d.							
e.	The value assigned in the STATUS clause shall not be updated to restore an items status (e.g., do not change "deprecated" to "current")	2580	7.2p2a	o	o		
f.	The STATUS clause of the NOTIFICATION-GROUP shall change when the status of a referenced notification type changes	2580	7.2p2f	o	o		

: NOTIFICATION-GROUP mapping

Mapping of the MODULE-COMPLIANCE macro

Section 3 of [RFC 2580](#) defines requirements related to the MODULE-COMPLIANCE macro. Until a practical benefit can be demonstrated in using such statements, the NTCIP community has decided to document conformance with PRLs, RTMs, MIB compliance tables, and supported range tables. PRLs and RTMs are defined in NTCIP 8002; MIB compliance tables are intended to document how the objects within a MIB have evolved over time; supported range tables are intended to precisely define the minimum required ranges for each [object](#) for each version of the

standard in a relatively concise manner. The MIB compliance and supported range tables should be presented as defined in this clause.

MIB Compliance Table

The top row of the MIB Compliance Table shall provide the column names. The first column shall be labeled \"Group\" and subsequent columns shall indicate a compliance identifier, which is typically the version number of the standard (e.g., \"v03\")[^57]. The first column of second row of the table shall contain the label \"Status\"; subsequent columns of the second row show indicate the status of each version of the MIB as follows:

- 1. \"c\" indicates current
- 2. \"d\" indicates deprecated
- 3. \"o\" indicates obsolete

Typically, only the latest version would be shown as current, but a standard could use multiple current columns to define different profiles (e.g., NTCIP 1203 could define a vXX profile for CMS and a vXX profile VMS, both of which are current simultaneously). Columns should only be shown as obsolete once all implementations are believed to be removed from service.

Subsequent rows of the table shall list each object and notification group defined by the MIB, with the descriptor of the group provided in the left-hand column of the table. The subsequent columns shall show the conformance status of the group as defined for the indicated version of the standard; conformance shall be shown using the conformance symbols as defined for use within PRLs. Finally, any cell within the table can be associated with a note via a reference symbol (e.g., \"[1]\"). The text of the notes shall appear immediately after the table.

Values within a column shall not be changed, except to correct typos or to change the status of the MIB version (i.e., as recorded in row 2). MIB versions that were previously documented as obsolete in a previous version of the standard can be omitted from the table. Each version of the standard should add at least one new column.

Table 28 provides an example of a MIB compliance table.

Group	v01	v02	v03	v04-VMS	v04-CMS
Version Status	o	d	d	c	c
dmsTypeGroupR1	M	M	M	M	M

dmsTechnologyGroupR1	0.1(2)	M	M	M	M
dmsDisplayCapabilitiesGroupR1	0.1(2)	Capabilities:M	Capabilities:M	Capabilities:M	Capabilities:M
dmsVmsCapabilitiesGroupR1		VMS:M	VMS:M	M	
dmsFontGroupR1	0.2	Fonts:M	Fonts:M	Fonts:M	Fonts:M
dmsFontGroupR1Ext		Fonts:M	Fonts:M	Fonts:M	Fonts:M

Table 28: Example MIB Compliance Table (based on a draft of NTCIP 1203)

Supported Range Table

MIB modules define the standardized allowed syntax for each object type; however, there are cases where the intent is to allow implementations to subrange the standardized syntax. Within prior versions of NTCIP (i.e., those based on SMIv1), the default policy was that implementations were allowed to subrange any object unless otherwise indicated. In general, the NTCIP standards were rather lax in stating these exceptions. Within SMIv2 (and specifically RFC 2580), the default is that implementations are required to support the full standardized range unless otherwise indicated.

To conform to the SMIv2 format, all NTCIP standards should include a supported range table to indicate the intent of supported ranges so that implementations (especially manger implementations) can manage expectations of what values might be supported. Specifically, these tables shall identify the minimum requirements for claiming support for an object and shall describe any conditional support rules.

The supported range table shall indicate the following columns in the header row:

- ¹. An "\Obj\" column that lists the descriptor of each object type in the MIB module for which implementations are not required to support the full range of values.
- ². A "\Versions\" column that identifies the compliance statements to which the refinement applies (this is typically just a version number but could include a version number coupled with a profile name).
- ³. A "\Refinement\" column that identifies the allowed variations in the object type that implementations are allowed to impose.

The first row of the table shall list the column headings. Subsequent rows are presented in groups, with each group consisting of one row that displays the object descriptor across an entire row of the table followed by one or more rows where each subsequent row list one or more versions and the refinement specification that applies to the object type for the indicated

versions. The refinement specification shall be documented according to the Object construct defined within the MODULE-COMPLIANCE macro in Section 2 of [RFC 2580](#), but omitting the `\\\"OBJECT\\\" value(ObjectName)\\\"` portion.

Table 29 provides an example of an object refinement table.

Obj	Versions	Refinement
dmsMessageMemoryType		
v01	SYNTAX INTEGER {	currentBuffer (5) }
	DESCRIPTION	\\\"The value of \\\"other\\\" is reserved for manufacturer specific use and is not expected to be widely used. Implementations shall support at least one of \\\"permanent\\\", \\\"changeable\\\", or \\\"volatile\\\" in addition to \\\"currentBuffer\\\". Support for \\\"schedule\\\" is conditional based on support for dmsActionTableGroupR1. The enumeration of \\\"blank\\\" was not defined in v01.\\\"
v02, v03	SYNTAX INTEGER {	currentBuffer (5) }
	DESCRIPTION	\\\"The value of \\\"other\\\" is reserved for manufacturer specific use and is not expected to be widely used. Implementations shall support at least one of \\\"permanent\\\", \\\"changeable\\\", or \\\"volatile\\\" in addition to \\\"currentBuffer\\\" and \\\"blank\\\". Support for \\\"schedule\\\" is conditional based on support for dmsActionTableGroupR1.\\\"
v04-VMS	SYNTAX INTEGER {	currentBuffer (5) }
	DESCRIPTION	\\\"The value of \\\"other\\\" is reserved for manufacturer specific use and is not expected to be widely used. Implementations shall support at least one of \\\"changeable\\\" or \\\"volatile\\\" in addition to \\\"currentBuffer\\\" and \\\"blank\\\". Implementations may support \\\"permanent\\\". Support for \\\"schedule\\\" is conditional based on support for dmsActionTableGroupR1.\\\"
v04-CMS	SYNTAX INTEGER {	permanent (2), currentBuffer (5) }
	DESCRIPTION	\\\"The value of \\\"other\\\" is reserved for manufacturer specific use and is not expected to be widely used. Implementations shall support

\permanent\, \currentBuffer\, and \blank\.
Implementations shall not support \changeable\ or
\volatile\. Support for \schedule\ is conditional
based on support for dmsActionTableGroupR1.\
+-----+
dmsMessageNumber
+-----+
v01 DESCRIPTION \Range is limited based on memory type as
follows:
permanent: per PICS
changeable: per PICS
volatile: per PICS
currentBuffer: (1)
schedule: (1)\
+-----+
v02,v03, v04-VMS, DESCRIPTION \Range is limited based on memory type as
v04-CMS follows:
permanent: per PICS
changeable: per PICS
volatile: per PICS
currentBuffer: (1)
schedule: (1)
blank: (1..255) \
+-----+
dmsMessageOwner
+-----+
v01,v02, v03 DESCRIPTION \Implementations shall support only
NVT-ASCII characters.\
+-----+
v04-VMS, v04-CMS DESCRIPTION \Implementations shall support all
NVT-ASCII characters and any other UTF-8 characters
identified in the PICS\
+-----+

Table 29: Example Object Refinement Table

Summary

The format of the MIB Compliance and Supported Range Tables are designed to facilitate the production of corresponding MODULE-COMPLIANCE statements if necessary, in the future. However, the tabular structure is intended to facilitate identifying changes that have occurred over time in a more concise manner and to reduce redundancy within the text.

Table 30 defines additional rules for populating the MIB compliance and object refinement tables.

Item	Requirement	RFC	Clause	RFC	NTCIP\
	Status[^58]	Status[^59]			
a.	Each version of a MIB shall be associated with a MIB compliance statement	2580	5p2	m	m
b.	The MIB compliance statement shall identify each required and optional <u>object group</u> and <u>notification group</u> for the indicated version	4181	4.8p2a	r	m
c.	The MIB compliance statement shall specify any allowed implementation variances (e.g., in access permissions or in supported <u>object</u> ranges)	4181	4.8p2b	m	m
d.	The groups listed in the MIB compliance statement shall include any and all required <u>object</u> groups and/or types from external MIBs, prefixed with their module descriptor	4181	4.8p5a	r	m
e.					
f.					
g.	When a writeable enumerated integer or a BITS construct that is referenced by the compliance statement has enumerations or named bits added, a note that specifies support for the original set of values shall be added to all previous versions of the respective <u>object</u> group(s) (if not already present)	4181	4.9p7	r	m
g.	The information contained within the MIB compliance statement shall be consistent with the information provided within the respective PRL(s).				

: MIB Compliance Table Rules

Additional NTCIP Requirements

Required Implementation Notes

Every NTCIP standard containing a MIB module shall contain the following normative text prior to its MIB compliance table.

The MIB compliance and object refinement tables may partially duplicate and extend requirements contained within the PRL and RTM. In case of any conflict between these sources, the PRL and RTM take precedence over the MIB compliance and object refinement tables.

To claim support for any object type, implementations shall:

-
- support the defined MAX-ACCESS and SYNTAX of the object type, unless the object refinement table defines a refinement for the object type for the relevant compliance statement;
-
- when the MIB object refinement table defines a refinement for an object type for the relevant compliance statement, support at least the minimum access and values specified.
-

NTCIP-Defined Table Types

NTCIP defines three types of conceptual table types: static, dynamic, and managed.

The number of conceptual rows contained within a static conceptual table shall not change during run time, irrespective of whether they have been initialized or not. As a result, an SNMP manager can successfully query any supported object from any supported conceptual row at any time but should be prepared to receive a data that is not initialized.

The number of conceptual rows contained within a dynamic conceptual table can change during run time, but only through processes internal to the SNMP agent (e.g., an agent creating a new log entry) that the manager does not directly control.

The number of conceptual rows contained within a managed conceptual table can change during run time due to SNMP management operations (and optionally due to internal operations).

In the case of dynamic and managed tables, an SNMP manager should be prepared to receive an error response if an object instance is requested before its creation or after its deletion.

Bitmap Format

The content of the format subclause is delimited by "<Format>" and shall be defined according to the following:

a) The format clause shall identify the meaning of each bit by its bit number. Bit numbering shall start with zero (0). This may be done in textual form (e.g., "each bit shall represent a sequential sensor number with sensor 1 mapped to Bit 0") or may use a format such as:

Bit Name Description

<x> <bitName> = <description>

...

1 <bitName> = <description>

0 <bitName> = <description> (Least Significant Bit of the Least Significant Byte)

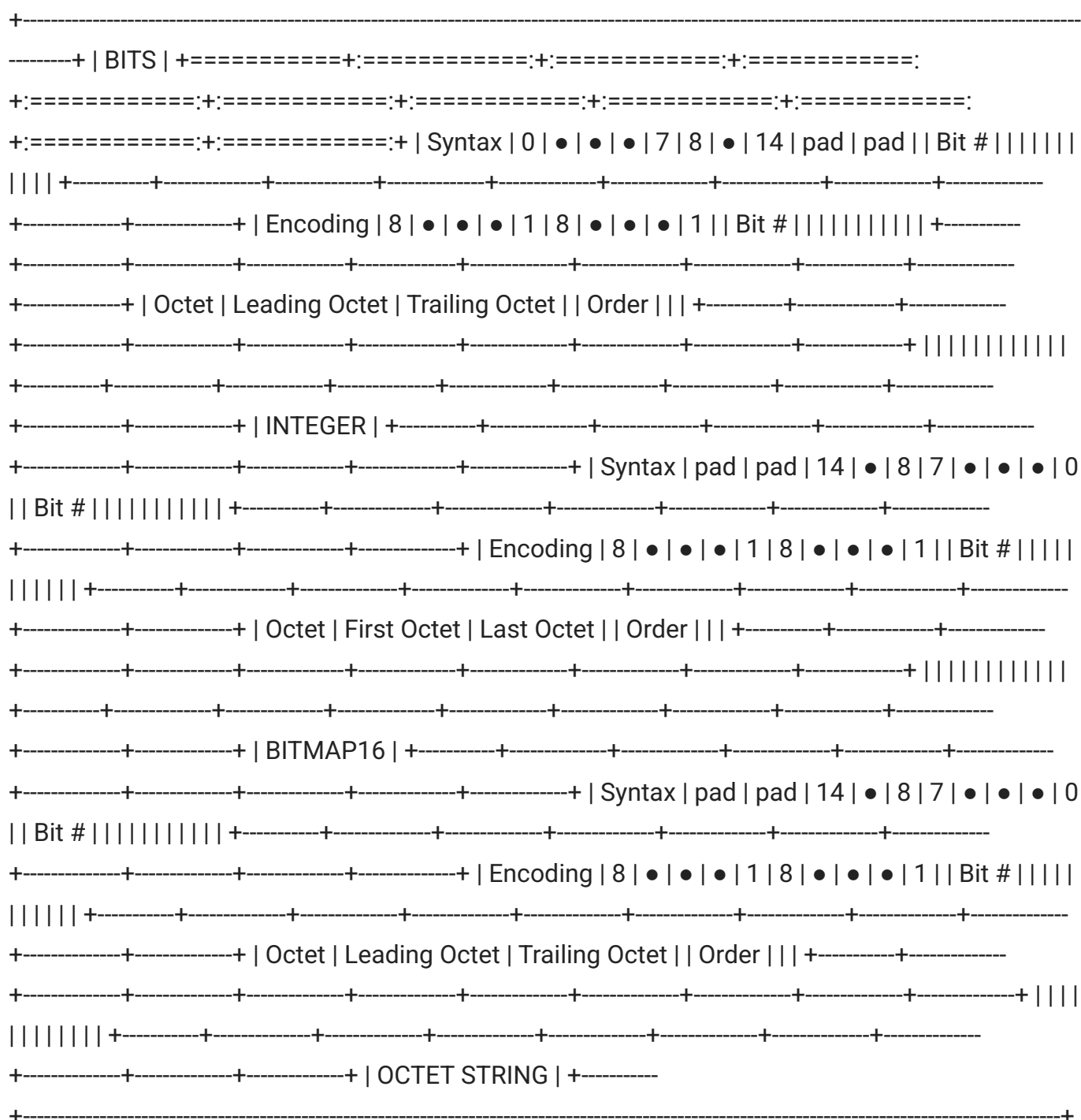
b) If the SYNTAX is based on a "BITS" construct, which is required for all new bitmapped objects, bit numbers are serialized according to the rules defined by RFC 3416 where Bit 0 is the "first bit". This means that Bit 0 is the high-order bit of the first octet in the serialization.

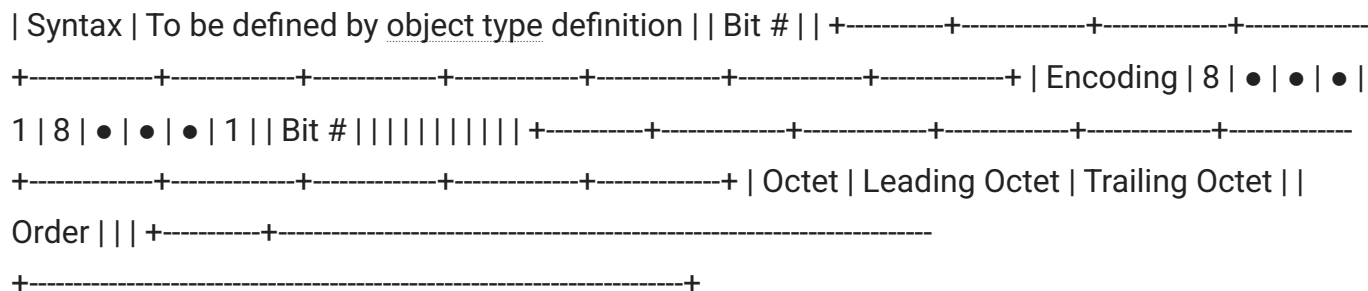
c) If the SYNTAX is based on an integer type (e.g., Integer32), which is only allowed for object types converted from SMIv1, Bit 0 shall represent the low-order bit. This means that Bit 0 is the low-order bit of the last octet in the serialization.

d) If the SYNTAX is based on one of the NTCIP octet-string-based bitmap textual conventions (e.g., NtcipOctetBitmap8, NtcipOctetBitmap16), which are only allowed for object types converted from SMIv1, Bit 0 shall be mapped to the lowest order bit of the final octet. This means that Bit 0 is the low-order bit of the last octet in the serialization.

e) If the SYNTAX is based on an OCTET STRING, which is only allowed for object types converted from SMIv1, the mapping of bit numbers to the bits within the serialization shall be defined within the \<Format> subclause of the DESCRIPTION clause of the object type.

Figure 5 depicts the impact of the above rules for an example object type with 14 bits defined (resulting in two bits of padding that might not be explicitly defined). Although new objects using bitmaps should use the BITS construct standardized by RFC 2578, older objects converted from SMIv1 might still use one of the other constructs.





Bit Ordering of a 14-Bit Value^[60]

ITSOerString Specifications

ITSOerString Syntax

The `"ITSOerString"` textual convention is defined in ISO 20684-1 as an unconstrained OCTET STRING representing value of an X.680 ASN.1 structure encoded using OER. Within SNMP, the size of an OCTET STRING is limited to 65535 octets. An object type that uses the ITSOerString should further constrain this size as tightly as appropriate for its intended use.

Within NTCIP, the definition of the X.680 ASN.1 structure shall be defined or referenced by the `<Definition>` subclause of the OBJECT-TYPE macro according to the rules in Clause 4.5.4.2.

ITSOerString Format

An object with a SYNTAX based on the ITSOerString or similar textual convention shall define the data structure to be encoded using any valid X.680 ASN.1 type; however, the following is substituted for the definition of NamedType to add mechanisms to reference other objects contained in the MIB:

NamedType ::=

identifier Type

| valuereference & `"."` & IndexSuffixes ExternalReference

| `"*"` & valuereference & `"."` & IndexSuffixes ExternalReference

IndexSuffixes ::=

IndexSuffix

| IndexSuffixes & `"."` & IndexSuffix

IndexSuffix ::= number | DefinedValue | \"(\ " ValueRange \")\" | \"*\"

ExternalReference ::=

\"-- @\" modulereference

| empty

where,

a) valuerreference is a descriptor assigned to an invocation of an OBJECT-TYPE macro.

b) IndexSuffixes is a list of the associated indexes as required to explicitly identify the exact instance(s) of a given object-type. Each index may be identified as either a specific value or a variable; if a variable is used, the variable shall be defined within the same module. The IndexSuffixes for a leaf object shall always be "0".

c) A non-empty ExternalReference shall be present at the end of every line containing a valuerreference, if the valuerreference is defined in a different module

d) modulereference, when present, indicates the MIB module where the object type referenced by valuerreference is defined

The first valuerreference option supports the definition of block object types. For example, within a SEQUENCE, instead of providing an "identifier Type" pair, the specification can indicate the name of an object instance that should exist within the agent. The object instance is identified using the descriptor for the defined OBJECT-TYPE along with the appropriate index value(s). The "Type" to be associated with the NamedType shall be the SYNTAX value assigned in the OBJECT-TYPE macro for the associated valuerreference. If an error occurs in trying to produce such an encoding, the entire ITS0erString shall produce a zero-length string.

EXAMPLE SEQUENCE { dmsSignAccess.0 -- \@NTCIP1203-Dms

dmsMessageMultiString.3.2 -- \@NTCIP1203-Dms

}

This would provide a structure containing the scalar value of dmsSignAccess and the dmsMessageMultiString for the second changeable message.

The second valuereference option supports the definition of configurable object types and entails an additional level of dereferencing. For example, within a SEQUENCE, instead of providing an "identifier Type" pair, the specification can indicate the name of an object instance (called the pointer object instance) that should exist within the agent and that is designed to reference an object instance (called the data object instance). The pointer object instance is identified using the descriptor for the defined pointer OBJECT-TYPE along with the appropriate index value(s). However, the actual data being encoded is the value pointed to by the pointer object instance, which is the value of the data object instance. As such, when encoding and decoding, the type to be used shall be derived from the SYNTAX assigned in the OBJECT-TYPE macro for the associated data object instance. If an error occurs in trying to produce such an encoding, the entire ITSOerString shall produce a zero-length string.

EXAMPLE SEQUENCE { *fdObjectGroupFieldObject.x.y.* }

This would provide a structure containing the ordered list of objects referenced by all fdObjectGroupFieldObjects with an initial index of x and a secondary index of y. For example, if fdObjectGroupFieldObject.x.y.1 referenced dmsMessageMultiString.3.2 and fdObjectGroupFieldObject.x.y.2 referenced dmsMessageRunTimePriority.3.2 with no other entries for fdObjectGroupFieldObject.x.y, the structure would equate to

SEQUENCE { dmsMessageMultiString.3.2 -- \@NTCIP1203-Dms

dmsMessageRunTimePriority.3.2 -- \@NTCIP1203-Dms

}

The "\"ValueRange\" notation of IndexSuffix shall be interpreted as a shorthand notation for listing every instance identified within the ValueRange. The "\"*\" notation of IndexSuffix shall be interpreted as a shorthand notation for listing every instance contained in the device in the same order as would appear in a walk of the MIB.

EXAMPLE For example, consider a device that has the following information in a conceptual table:

indexA	indexB	sampleObject
1	1	A

1 1 A

1 2 B

2 1 C

3 1 D

3 2 E

The field:

`sampleObject.(1..2).*`

would evaluate to

`sampleObject.1.1,`

`sampleObject.1.2,`

`sampleObject.2.1`

Annex C provides several examples.

Object Identifier Layout

All items that NTCIP registers on the international object identifier tree shall be registered under the nema node at { iso(1) org(3) dod(6) internet(1) private(4) enterprises(1) nema(1206) }.

NTCIP standards may reference and require support for objects located elsewhere on the international object identifier tree.

The NTCIP Coordinator shall ensure the assignment of a unique node(s) to each standard that requires a X.208 module. This assignment(s) shall be documented within an X.208 module contained within the standard. Such assignments are typically used as the name of the MODULE-IDENTITY invocation within the module.

NOTE---Secondary assignments might be used for administrative or other objects that need to exist in other portions of the international object identifier tree.

Nodes under the MODULE-IDENTITY

Exceptions to these rules can be allowed with the approval of the Joint Committee on the NTCIP.

Notification Types

The sub-identifier zero (0) under the MODULE-IDENTITY node shall be assigned or reserved for a descriptor identical to that of the MODULE-IDENTITY with the suffix \"Notifications\" added. Any notification types defined in the module shall be defined directly under this node. This ensures

that each notification type is assigned a name with the next-to-last sub-identifier equal to zero (0), which is required by Clause 4.7 of RFC 4181.

Example --- ntcipSmiNotifications would have the RELATIVE-OID of { transportation(4) protocols(1) ntcipSmi(5) 0 }[^61]

Conformance

The sub-identifier 127 under the MODULE-IDENTITY node shall be assigned or reserved for a descriptor identical to that of the MODULE-IDENTITY with the suffix \"Conformance\" added.

NOTE---RFC 4181 recommends placing all object types under sub-identifier one (1) and placing conformance under sub-identifier two (2). NTCIP standards place object types directly below the MODULE-IDENTITY node and shifts the conformance node to sub-identifier 127 to allow sequential numbering of object types.

Example --- ntcipSmiConformance would have the RELATIVE-OID of { transportation(4) protocols(1) ntcipSmi(5) 127 }[^62]

Sub-identifier one (1) under the conformance node shall be reserved for a compliance node with the descriptor of the MODULE-IDENTITY with the suffix \"Compliances\". As of 2023, NTCIP does not define MODULE-COMPLIANCE statements, but this node is reserved for this purpose if the policy changes.

Example --- ntcipSmiCompliances would have the RELATIVE-OID of { transportation(4) protocols(1) ntcipSmi(5) ntcipSmiConformance(127) 1 }

Sub-identifier two (2) under the conformance node shall be assigned to or reserved for a group node with the descriptor of the MODULE-IDENTITY with the suffix \"Groups\".

Example --- ntcipSmiGroups would have the RELATIVE-OID of { transportation(4) protocols(1) ntcipSmi(5) ntcipSmiConformance(127) 2 }

OBJECT-GROUP and NOTIFICATION-GROUP macros included in the MIB module shall be assigned a sub-identifier under the group node. The descriptor for the group shall include the descriptor of the MODULE-IDENTITY, descriptive terms as needed, and a suffix of \"GroupR#[Ext#]\", where \"[Ext#]\" is an optional extension number used to extend a previously defined group and each \"#\" represents a sequential number of the revision or extension. The \"R#\" should indicate the sequential position of the REVISION as recorded in the MODULE-

IDENTITY\'s revision history, or the DESCRIPTION clauses within the revision history should indicate how the \'R#\''s relate to the revisions.

Example --- dbMgmtV2GroupR1 is assigned to the RELATIVE-OID of { transportation(4) devices(2) global(6) globalV2(9) dbMgmtV2(1) dbMgmtV2Conformance(127) dbMgmtV2Groups(2) 1 }

Object Types

OBJECT-TYPE macros included in the MIB module shall be assigned a sub-identifier under the standard node, perhaps with intermediate nodes to further group object types.

Example --- dbMgmtV2Mode has the RELATIVE-OID of { transportation(4) devices(2) global(6) globalV2(9) dbMgmtV2(1) 1 }

MIB Design Considerations

Maintain Secure Data

Access to data defined within the device needs to be controlled under all scenarios without any possibility of enabling any indirect access by unauthorized users. For example, a configurable event log might monitor data within the device to create a log entry to capture data when a defined event occurs. If a user is able to configure an event to monitor or record data that the user is not authorized to access, a security vulnerability exists.

The designer of the MIB module shall ensure that every object that points to another object within the device is unambiguously associated with appropriate security credentials that are to be used when accessing that data.

Reuse Existing Structures

Developers of NTCIP MIB modules should be familiar existing MIBs published by

- * IETF (i.e., as RFCs), especially
- * RFC 2981, event MIB;
- * RFC 2982, expression MIB; and
- * RFC 3014, notification MIB;
- * RFC 3231, scheduling MIB;
- * RFC 3433, entity sensor MIB;
- * RFC 3877, alarm MIB;
- * RFC 4268, entity state MIB;
- * RFC 6933, entity MIB;
- * ISO (especially the ISO 20684 series)
- * NTCIP

Many times, structures that developers wish to create already have existing MIBs that:

- * Can be used directly
- * Can be used with minor changes
- * Can be used as a basis of a new design, or
- * Simply provide insights into issues that should be considered during a new design

In particular, developers should be aware of the general-purpose input/output capability defined in ISO 20684-2 and how it can be easily extended to support a standardized interface with virtually any generic sensor or actuator.

Use of Table Indexes

This document does not preclude the potential for two or more management applications from simultaneously accessing the same object. However, designers of MIB modules should consider such situations and use designs to minimize conflicts and maximize security. In particular, designers should consider adding an initial owner index to tables when there is a need to prevent different managers from accessing each other's data or configuration. This will allow administrators to configure access to rows within the table by simply specifying the owner identifiers to which they should have access.

Developers should consider using the ISO 20684-7 fdOwnerTable for this purpose. Developers should also be cautious in defining overall management features for the table that might allow users with restricted access to monitor activities by other users. This can be achieved by placing monitoring objects into a table that augments the fdOwnerTable so that each owner can monitor statistics related to their own rows in the table but not universal statistics. Universal statistics, if defined, should be placed in a separate area of the international object identifier tree so that administrators can easily control access.

Consistency with the PRL and RTM

The module compliance statement, as defined by the module compliance table and object refinement table shall be consistent with the Protocol Requirements List (PRL) and Requirements Traceability Matrix (RTM).

Multi-Version Interoperability (MVI, Backward Compatibility)

With the development and publication of newer versions of various NTCIP standards, the need to address and ensure multi-version interoperability (MVI, often referred to as "backward compatibility" with a previous version, or interoperability to a defined extent with future

versions). To promote MVI, the standard shall ensure that standards provide adequate documentation that defines what features are supported by each version.


Row Status in Static Tables

The MIB module for NTCIP 8004 defines the deprecated NtcipRowStatusStatic textual convention; the UML state transition diagrams for this textual convention are provided in Figure 6, Figure 7, and Figure 8.

Row Status Static---Invalid

Row Status Static---Standby

Row Status Static---Available

 June 13, 2025

Requirements for Agent Implementations [Normative]

Agent Capabilities

Manufacturers supplying transportation equipment should make AGENT-CAPABILITIES statements available for their equipment. All manufacturers need to obtain a vendor number from either NEMA or IANA prior to creating MIB modules that have manufacturer-specific data.

Requirements for Non-Standard MIBs

All MIB modules shall conform to the basic compilation rules defined for SMIv2 in addition to the requirements defined within this section. Developers of non-standard MIB modules are encouraged to adopt the requirements defined in this document to the extent to which they might apply.

Default Values

The value specified in a DEFVAL clause should be used when initializing a new object instance (e.g., during reboot or row creation), if feasible unless another value is provided. [^63]

Extensions of Standardized Enumerations

Unless stated otherwise within an object type \<Definition> subclause, a NamedNumber (i.e., an enumerated item in an enumeration) that is assigned an identifier of "other" shall be treated as a read-only value. An attempt to set an object instance to a read-only value shall return an error ("wrongValue" in the case of SNMPv3).

Unless normative text is added to specifically prohibit the use of the "\"other\" state, a user- or manufacturer-specific object shall be permitted to define an object specification that extends the possible states, For example, NTCIP 1202:2005 includes the following object:

coordCorrectionMode OBJECT-TYPE

SYNTAX INTEGER { other (1),

dwell (2),

shortway (3),

addOnly (4) }

ACCESS read-write

STATUS mandatory

DESCRIPTION

\"<Definition> This object defines the Coord Correction

Mode. The possible modes are:

other: the coordinator establishes a new offset by a mechanism not defined in this standard.

dwel: when changing offset, the coordinator shall establish a new offset by dwelling in the coord phase(s) until the desired offset is reached.

shortway (Smooth): when changing offset, the coordinator shall establish a new offset by adding or subtracting to/from the timings in a manner that limits the cycle change. This operation is performed in a device specific manner.

addOnly: when changing offset, the coordinator shall establish a new offset by adding to the timings in a manner that limits the cycle change. This operation is performed in a device specific manner.

...

To define a new correction mode, something like the following proprietary object could be used:

xxxCoordCorrectionModeExt OBJECT-TYPE

SYNTAX INTEGER { other (1),

subOnly (2) }

ACCESS read-write

STATUS mandatory

DESCRIPTION

\<Definition> This object defines an extension to the Coord Correction

Mode as defined in NTCIP 1202. The possible modes are:

other: the coordinator establishes a new offset according to

NTCIP-1202::coordCorrectionMode.

subOnly: when changing offset, the coordinator shall

establish a new offset by subtracting from the timings in such a

manner that limits the cycle change. This operation is performed


in a device specific manner

...

In this case, setting xxxCoordCorrectionModeExt equal to \"subOnly\" forces coordCorrectionMode equal to \"other\". Setting coordCorrectionMode equal to \"shortway\" forces xxxCoordCorrectionModeExt equal to \"other\".

Guidelines for Operating Agencies [Informative]

Operating agencies should be aware that there are cases where two or more management applications are potentially able to simultaneously access the same object. For example, the Global Database Management group does not prevent multiple management applications from accessing the data simultaneously. It is the responsibility of any agencies involved in inter-jurisdictional control to define procedures to ensure against this situation.

 June 13, 2025