NOTE: This draft dated 14 July 2026 prepared by U.S. Army DEVCOM C5ISR Center has not been approved and is subject to modification. DO NOT USE FOR ACQUISITION PURPOSES.

METRIC

MIL-STD-3071
w/CHANGE 2
DRAFT

SUPERSEDING
MIL-STD-3071
w/CHANGE 1
08 July 2024

DEPARTMENT OF DEFENSE INTERFACE STANDARD

TACTICAL MICROGRID COMMUNICATIONS AND CONTROL

AMSC N/A **FSC 6115**

FOREWORD

This standard is approved for use by all Departments and Agencies of the Department of Defense (DoD)

This standard establishes criteria to enable the interoperability of hardware and software necessary to operate a tactical microgrid on the battlefield with respect to the design, intelligent control, stability and performance, security, safety of personnel, and the protection of the tactical microgrid system and equipment.

This standard defines the tactical microgrid architecture using open standards to support a modular, highly cohesive system structure in order to leverage the collaborative innovation of industry, academia, and government participants along with stakeholders.

This document is intended to be free of Essential Patent Claims. Although development of this standard was made with the active participation of the Private Sector, non-government contributors have certified that they have declared all essential patent claims found in this standard.

Comments, suggestions, or questions on this document should be emailed to usarmy.apg.devcom-c5isr.mbx.standardization-crx@army.mil. Since contact information can change, you may want to verify the currency of this address information using the ASSIST Online database at https://assist.dla.mil.

A TMS website connects the Government and the public for information sharing/dissemination on MIL-STD-3071. It will host the current published MIL-STD, upcoming approved changes to the standard, support resources, and provides announcement space for important TMS topics. The website can be found at https://go.mil/tms.

SUMMARY OF CHANGE 2 MODIFICATIONS

1. All devices must implement new requirements for Object Management Group Data Distribution Service (OMG DDS) network interoperability. See section D.1 Network Interoperability for details.

2. Introduced data model version 2.0.3. All changes are backwards compatible with versions 2.0.1 and 2.0.2.

3. Added machine readable files for topic definition figures and tables.

4. Added optional specifications for DDS Security integration.

5. Added an improved Change History at the end of the Appendix.

Table of Contents

1. SCOPE. 1
1.1. Scope. 1
1.2. Purpose. 1
1.3. Application. 1
1.4. Classification. 1
1.5. Limitations. 1
2. APPLICABLE DOCUMENTS 1
2.1. General. 1
2.2. Government documents. 2
2.2.1. Specifications, standards, and handbooks. 2
2.2.2. Other Government documents, drawings, and publications. 2
2.3. Non-Government Publications. 3
2.4. Order of Precedence. 3
3. DEFINITIONS. 4
3.1. Acronyms. 4
3.2. Definitions 5
3.2.1. Automatic Voltage Regulator (AVR). 5
3.2.2. Backwards compatibility. 6
3.2.3. Command. 6
3.2.4. Configuration. 6
3.2.5. Device. 6
3.2.6. Distribution. 6
3.2.7. Essential patent. 6
3.2.8. Exciter. 6
3.2.9. Fingerprint. 6
3.2.10. Generating source. 6
3.2.11. Grid. 6
3.2.12. Interconnection. 6
3.2.13. Interface. 6
3.2.14. Interoperability. 6
3.2.15. Inverter. 6
3.2.16. Legacy equipment. 6
3.2.17. Load. 7

3.2.18. Load shedding. 7
3.2.19. Operator. 7
3.2.20. Power Device. 7
3.2.21. Open standard. 7
3.2.22. Performance. 7
3.2.23. Point of Connection (POC). 7
3.2.24. Ring-bus off-line. 7
3.2.25. Setting. 7
3.2.26. Source (SRC). 7
3.2.27. Storage device. 7
3.2.28. Tactical. 7
3.2.29. Tactical Microgrid (TMG). 7
3.2.30. Tactical Microgrid Controller (TMC). 7
3.2.31. User. 8
3.2.32. Warfighter. 8
4. GENERAL REQUIREMENTS 8
4.1. General. 8
4.2. Types of Tactical Microgrid Equipment. 8
4.2.1. Type I – Power Generation Sources. 8
4.2.2. Type II – Energy Storage Systems. 8
4.2.3. Type III – Feeder Systems. 8
4.2.4. Type IV – Distribution Systems. 8
4.2.5. Type V – Control Systems. 9
4.2.6. Type VI – Loads. 9
4.2.7. Type VII – Power Conversion. 9
4.3. Interoperability. 9
4.4. Standalone Use. 9
4.4.1. Type I Power Generation Sources and Type II Energy Storage Equipment. 9
4.4.2. Type III Feeder and Type IV Distribution Systems Equipment. 9
4.5. Degraded Operation. 9
4.6. Monitoring and Control. 10
4.7. Cybersecurity. 10
4.7.1. Risk Management Framework (RMF). 10
4.7.2. Operational Resilience. 10

4.7.3. Access to firmware/software. 10
4.8. Tactical Microgrid Architecture. 11
4.8.1. Common Microgrid Control System. 11
4.8.2. Common Microgrid Monitoring System. 11
4.8.3. Levels of Control. 11
4.8.4. Tactical Microgrid Control (TMC) Redundancy. 12
4.8.5. Communications Architecture. 12
4.9. Data Model. 12
4.9.1. Device Identity. 13
4.9.2. Device Roles. 13
4.9.3. Platforms. 14
4.9.4. Data flows. 14
4.9.5. Publishers and Subscribers. 15
4.9.6. Topic and Message Flow. 15
4.9.7. Data Types. 15
4.9.8. Data Flow Patterns. 16
4.9.9. Instances. 17
4.9.10. Timing 17
5. DETAILED REQUIREMENTS. 18
5.1. Safety 18
5.2. Human Engineering. 18
5.3. TMG Equipment Roles and Responsibilities 18
5.3.1. Type I – Power Sources 18
5.3.2. Type II – Energy Storage 20
5.3.3. Type III – Feeder Systems 21
5.3.4. Type IV – Distribution Systems 22
5.3.5. Type V – Control Systems 23
5.3.6. Type VI – Loads 25
5.3.7. Type VII – Power Conversion 26
5.4. Communications Interface 28
5.4.1. Application Layer. 28
5.4.2. Presentation Layer. 28
5.4.3. Session Layer. 28
5.4.4. Transport Layer. 28

5.4.5. Network Layer. 29
5.4.6. Physical Layer. 29
6. NOTES 29
6.1. Intended Use. 29
6.2. Acquisition requirements. 29
6.3. Tailoring guidance 29
6.4. Subject term (key word) Listing 29
6.5. Changes from previous issue 29
APPENDIX (DATA MODEL) 30
CONCLUDING MATERIAL 422

1. SCOPE.

1.1.Scope. This standard establishes the communication and control interface requirements for Tactical Microgrids (TMGs) to enable TMG components to operate as a single entity to provide electricity in a tactical environment. This standard defines the component interfaces of the hardware and software necessary to provide intelligent control, power quality, electrical stability, electrical performance, safety of personnel, security, and protection.

A tactical microgrid is a warfighter-operated and maintained power system consisting of a mobile, flexible group of interconnected power generation sources, distribution, energy storage, and load devices that act as a single, controllable system to provide electricity on the battlefield.

1.2.Purpose. The purpose of this DOD standard is to define an open communication and control architecture for TMGs to facilitate:

a. Interoperability between manufacturers of tactical microgrid components

Safety of personnel and equipment

Backwards compatibility with legacy equipment wherever possible

Security of the tactical microgrid communication network and controllers

Reduction of procurement, operational, sustainment, and maintenance costs

1.3.Application. This standard is applicable to tactical microgrid systems, subsystems, and individual equipment. While this standard provides interoperability and design criteria guidance, it is not intended to limit innovation in the design or selection of specific hardware, software, materials, and processes.

1.4.Classification. The tactical microgrids are made up of the following types of equipment:

Table 1-1 Types of Equipment in a Tactical Microgrid

Types of Equipment in a Tactical Microgrid	
Type I	Power Generation Sources (e.g. Engine-Generator sets, Solar, Wind, etc.)
Type II	Energy Storage Systems, (e.g. batteries, capacitors, etc.)
Type III	Feeder Systems (e.g. 3 phase inputs with 3 phase outputs)
Type IV	Distribution Systems, (e.g. 1 or 3 phase inputs with 1 phase outputs)
Type V	Control Systems w/ Communications and cabling
Type VI	Loads (e.g. Improved Environmental Control Units)
Type VII	Power Conversion

1.5.Limitations. This document does not define test procedures that verify compliance with this standard. This document is not to be referenced as the sole source for cybersecurity requirements.

2. APPLICABLE DOCUMENTS

2.1.General. The documents listed in this section are specified in sections 3, 4, or 5 of this standard. This section does not include documents cited in other sections of this standard or

recommended for additional information or as examples. While every effort has been made to ensure the completeness of this list, document users are cautioned that they must meet all specified requirements of documents cited in sections 3, 4, or 5 of this standard, whether or not they are listed.

2.2.Government documents.

2.2.1. Specifications, standards, and handbooks. The following specifications, standards, and handbooks form a part of this document to the extent specified herein. Unless otherwise specified, the issues of these documents are those cited in the solicitation or contract.

DEPARTMENT OF DEFENSE STANDARDS

MIL-STD-882	System Safety
MIL-STD-1332	Definitions of Tactical, Prime, Precise, and Utility Terminologies for Classification of the DOD Mobile Electric Power Engine Generator Set Family
MIL-STD-1472	Human Engineering

(Copies of these documents are available online at https://quicksearch.dla.mil.)

2.2.2. Other Government documents, drawings, and publications. The following other Government documents, drawings, and publications form a part of this document to the extent specified herein. Unless otherwise specified, the issues of these documents are those cited in the solicitation or contract.

ARMY REGULATIONS

AR 385-10	The Army Safety Program

(Copies can be obtained at https://armypubs.army.mil/ProductMaps/PubForm/AR.aspx.)

DOD ISSUANCES

DODI 8500.01	Cybersecurity

(Copies can be obtained at https://www.esd.whs.mil/DD/DoD-Issuances/.)

NATIONAL INSTITUTES OF SCIENCE AND TECHNOLOGY

NIST SP 800-39	Managing Information Security Risk: Organization, Mission, and Information System View
NIST SP 800-53 Rev. 5	Security and Privacy Controls for Information Systems and Organizations

NIST SP 800-82 Rev. 2 Guide to Industrial Control Systems (ICS) Security

(Copies can be obtained at https://www.nist.gov/.)

2.3. Non-Government Publications. The following documents form a part of this document to the extent specified herein. Unless otherwise specified, the issues of these documents are those cited in the solicitation or contract.

INSTITUTE OF ELECTRICAL AND ELECTRONIC ENGINEERS (IEEE)

IEEE 802.3 Standard for Ethernet

(Copies can be obtained at https://www.ieee.org/)

INTERNATIONAL ORGANIZATION FOR STANDARDIZATION

ISO/IEC 7498-1	Information technology – Open Systems Interconnection – Basic Reference Model: The Basic Model
ISO/IEC 7498-3	Information technology – Open Systems Interconnection – Basic Reference Model: The Basic Model Naming and Addressing
ISO/IEC 7498-4	Information Processing Systems- Open Systems Interconnection – Basic Reference Model: Part 4 Management Framework

(Copies can be obtained at https://www.iso.org)

INTERNET ENGINEERING TASK FORCE

RFC-8200 Internet Protocol, Version 6 (IPv6) Specification

(Copies can be obtained at https://www.ietf.org/)

NATIONAL FIRE PROTECTION ASSOCIATION (NFPA)

NFPA-70	National Electrical Code
NFPA-70E	Standard for Electrical Safety in the Workplace

(Copies can be obtained at https://nfpa.org/)

2.4. Order of Precedence. Unless otherwise noted herein or in the contract, in the event of a conflict between the text of this document and the references cited herein, the text of this

document takes precedence. Nothing in this document, however, supersedes applicable laws and regulations unless a specific exemption has been obtained.

3. DEFINITIONS.

3.1. Acronyms.

ACRONYM	DEFINITION
AC	Alternating Current
AN	Application Note
ANSI/NEMA	American National Standards Institute / National Electrical Manufacturers Association
API	Application Programming Interface
AR	Army Regulation
DC	Direct Current
DDS	Data Distribution Service
DER	Distributed Energy Resource
DOD	Department of Defense
DODI	Department of Defense Instruction
ECU	Environmental Control Unit
EIO	Energy Informed Operations
GCRA	Generic Cell Rate Algorithm
IEC	International Electrotechnical Commission
IEEE	Institute of Electrical and Electronics Engineers
IETF	Internet Engineering Task Force
IoT	Internet of Things
IP	Internet Protocol
IPv6	Internet Protocol Version 6
ISO	International Organization for Standardization
IT	Information Technology
JSON	JavaScript Object Notation
MBS	Maximum Burst Size
MC	Microgrid Controller
MIL-STD-XXX	Defense Standard
MMC	Master Microgrid Controller

ACRONYM	DEFINITION
NEC	National Electrical Code
NFPA	National Fire Protection Association
NIST	National Institute of Standards and Technology
NSN	National Stock Number
OMG	Object Management Group
OpenFMB™	Open Field Message Bus
OSI	Open Systems Interconnection
OT	Operational Technology
PCR	Peak Cell Rate
PTP	Precision Time Protocol
QoS	Quality of Service
RFC	Request For Comment
RMF	Risk Management Framework
RMS	Root Mean Squared
ROM	Read Only Memory
RoT	Root of Trust
SDLC	System Development Lifecycle
SP	Special Publication
TM	Technical Manual
TLS	Transport Layer Security
TMG	Tactical Microgrid
TMC	Tactical Microgrid Controller
TMS	Tactical Microgrid Standard
TPM	Trusted Platform Module
UDP	User Datagram Protocol
UI	User Interface
VAC	Volts AC
XML	Extensible Markup Language

3.2. Definitions

3.2.1. Automatic Voltage Regulator (AVR). A device that automatically adjusts the synchronous generator's excitation to regulate the voltage output to either a predetermined level

or based on a stable reference, for example, the sinusoidal output Root Mean Squared (RMS) voltage.

3.2.2. Backwards compatibility. The ability of a system or component to operate with previously-fielded systems or components of the system.

3.2.3. Command. An action with an immediate effect that the system must attempt to execute. A failure results in an error.

3.2.4. Configuration. A collection of devices and connections that complete a TMG system.

3.2.5. Device. A standalone component that is part of the TMG system, including generating sources, Tactical Microgrid Controller (TMC), distribution, storage, and loads.

3.2.6. Distribution. All the equipment necessary to distribute power from the sources to the loads, including protective devices, boxes, and cabling. Distribution equipment include both Type III Feeder Systems and Type IV Distribution Systems.

3.2.7. Essential patent. A patent required to be obtained in order to produce a standard compliant product.

3.2.8. Exciter. A device that provides electric current to produce the rotor's magnetic field in a generator, for example, a battery or small generator.

3.2.9. Fingerprint. A data type that assigns a unique identity to each TMS device, associated with a single specific device role.

3.2.10. Generating source. Equipment which consumes fuel to provide power to the TMG, typically synchronous diesel generators.

3.2.11. Grid. General term for a system of components connected together. In this standard, grid refers to either a Utility Power Grid or a Microgrid.

3.2.12. Interconnection. Process of connecting military generation and distribution equipment to a powered or un-powered Power System.

3.2.13. Interface. The point where two systems meet and interact.

3.2.14. Interoperability. The ability to act coherently, effectively, and efficiently to achieve tactical, operational, and strategic objectives. The devices must communicate, provide, and use services and information between systems, regardless of manufacturer or version, without the need for operator intervention.

3.2.15. Inverter. Device that changes a direct current power source into an alternating current or voltage source.

3.2.16. Legacy equipment. In this standard, refers to any and all military generation, distribution, storage, and load equipment deployed before the release of this standard.

3.2.17. Load. A user device that is powered by the TMG.

3.2.18. Load shedding. Distribution disconnects one or more loads when the generators are overloaded until the overloaded condition is removed.

3.2.19. Operator. A warfighter tasked to run, and maintain the tactical microgrid.

3.2.20. Power Device. A standalone component that is part of the TMG system, including generating sources, distribution, storage, and loads. A power device does not include the TMC.

3.2.21. Open standard. Standards made available to the general public that are developed (or approved) and maintained via a collaborative and consensus driven process.

3.2.22. Performance. The capability of a device when observed under particular conditions.

3.2.23. Point of Connection (POC). The power connection between any power device and the TMG.

3.2.24. Ring-bus off-line. A ring-bus microgrid configuration where one bus segment is connected but unpowered and kept as a spare. In the event a bus segment is cut, the spare bus segment would be energized.

3.2.25. Setting. Modifies either the software or hardware behavior of the device.

3.2.26. Source (SRC). A device that provides energy to the microgrid, including both generating and renewable sources.

3.2.27. Storage device. A system that saves and stores energy to be used at a later time.

3.2.28. Tactical. A type of generator sets, IAW MIL-STD-1332, designed for high mobility in direct support of military forces where output of generator sets is normally, but not exclusively, used at generated voltages without the necessity of transformation or extensive distribution systems. Life Characteristics are considered secondary to lightweight, small size, and a high degree of mobility. In the context of this document, Tactical applies to all types of tactical microgrid equipment specified in Table 1-1 Types of Equipment in a Tactical Microgrid.

3.2.29. Tactical Microgrid (TMG). A tactical microgrid is a warfighter-operated and maintained power system consisting of a mobile, flexible group of interconnected power generation sources, distribution, storage, and load devices that act as a single, controllable system to provide electricity on the battlefield. Intended to be self-contained and readily deployable, the TMG may utilize alternative energy resources, energy storage, and be capable of interfacing with other grids.

3.2.30. Tactical Microgrid Controller (TMC). A device, which an operator can access to set up, run, or maintain the system, which can communicate with other devices and automatically change their settings.

3.2.31. User. A warfighter allowed to view the current state and make minor changes to the tactical microgrid.

3.2.32. Warfighter. Military personnel with primary skills in combat that may have limited staff with generator experience. Warfighters can be both users and operators.

4. GENERAL REQUIREMENTS

4.1.General. The Tactical Microgrid (TMG) is the essential tool needed to reduce fuel consumption while producing and distributing mission critical electricity on the battlefield. The TMG must have the ability to produce and distribute electricity in a tactical environment effectively, and efficiently to enable the Warfighter to meet mission needs. This section provides general requirements for tactical microgrid interface and design to meet this need.

4.2.Types of Tactical Microgrid Equipment. There are basic components necessary for a tactical microgrid. Each component or end item has specific functions to enable the microgrid to function. These components are classified in para 1.4 and Table 1-1. The basic functions and requirements are summarized in the following paragraphs.

4.2.1. Type I – Power Generation Sources. Type I - Power Generation Sources are end items designed to produce electricity for the TMG. Each item has the capability to operate as an individual power source or can be synchronized to provide power as a TMG. As an individual power source, they can be used to provide power to a single load such as a shelter or radar system. As an element of the TMG, multiple power sources can be synchronized and controlled to turn on and off based on the load on the TMG. Type I equipment includes, but is not limited to, typical diesel engine driven generator sets, alternative energy power sources such as solar, wind, fuel cells, etc. and hybrids.

4.2.2. Type II – Energy Storage Systems. Type II – Energy Storage Systems provide stability to the TMG during transitions and peak shaving. Energy storage systems shall provide power during the period when additional power is necessary but the power source has not been added to the grid. In addition, energy storage systems shall function as an uninterruptable power supply (UPS) or as a hybrid when paired with a power source for individual loads. Energy storage systems shall include energy conversion to enable individual operation or connection to the TMG. Individual operation means not functioning as part of the TMG but can function when connected to a load.

4.2.3. Type III – Feeder Systems. Type III – Feeder Systems are a type of distribution equipment which provide the interface or connection points to the TMG for the other types TMG equipment. Feeder systems define the TMG grid itself. They can be added or subtracted to define the electrical bus necessary to provide the power to the loads directly or through the distribution systems. They can also provide energy conversion of power sources into the TMG or loads out of the TMG. Feeder systems shall include the ability to manage the circuits through independent control or through the use of Type V – Control Systems equipment. Feeder systems shall have metering and monitoring systems incorporated to enable the Warfighter to manually control the system and to automatically control the system using Type V Control Systems

4.2.4. Type IV – Distribution Systems. Type IV – Distribution Systems provide the power from the TMGs through the Type III – Feeder Systems to loads which require individual circuits for standard lighting and standard duplex receptacles. Type II equipment can be individual items or integrated into various shelters. Distribution systems shall include the ability to manage the

circuits through independent control or through the use of Type V – Control Systems equipment. Distribution systems shall have metering and monitoring systems incorporated to enable the Warfighter to manually control the system and to automatically control the system using the Type V Control Systems.

4.2.5. Type V – Control Systems. Type V – Control Systems provide power management for the TMG. Control systems monitor various parameters of the TMG, such as current, and add or subtract power generation sources and/or add or subtract load to ensure power is available. Control systems shall monitor and protect the system in the event of overload or faults. Control systems shall reside as an individual item or be built into the various types of TMG equipment.

4.2.6. Type VI – Loads. Type VI – Loads consume the power provided by the TMG. Standard loads consume power but do not communicate with the TMG. Smart loads also consume power but can communicate with the TMG to protect the TMG. Smart loads allow the TMG controller to control their loads to prevent the grid from being overloaded. In addition, smart loads can notify the TMG controller about their pending activation. This notification will allow the TMG controller to ensure adequate power is available on the TMG for the additional load.

4.2.7. Type VII – Power Conversion. Type VII – Power Conversion systems enable Type I Power sources and Type II Energy Storage systems which produce different voltages and/or are Direct Current (DC), to connect to the TMG feeder bus. These systems convert the source/storage power to the common TMG bus AC voltage. These systems can be separate items or can be integrated into the source/storage systems they support.

4.3.Interoperability. Individual equipment types specified above, shall communicate, and interact with one another in order to interoperate as a TMG, regardless of manufacturer or version, and without the need for operator intervention. Interactions include but are not limited to starting and shutdown, synchronization, grid connection/disconnection, parallel operation to include proper load sharing, load management, etc. Each item shall have standard power and communication connections to enable this interoperability.

4.4.Standalone Use.

4.4.1. Type I Power Generation Sources and Type II Energy Storage Equipment. Type I Power Generation Sources and Type II Energy Storage equipment shall be capable of operating independently to provide power in a standalone or spot generation configuration. Power generation sources and energy storage systems will be operated independent of the TMG for loads not on the TMG.

4.4.2. Type III Feeder and Type IV Distribution Systems Equipment. Type III Feeder and IV Distribution Systems equipment shall be capable of operation in an independent mode when they are independent of a TMG. Feeder and Distribution systems will be operated independent of the TMG if separated by a fault in the TMG or when providing power distribution for a standalone power source. In addition, Type III and Type IV equipment shall be capable or operating in a manual mode if automatic control is lost.

4.5.Degraded Operation. The TMG shall have the ability to degrade operations when experiencing grid instability, faults, or failures. The TMG shall be capable of autonomous and operator-implemented degraded operation during loss of communication events, equipment failures, and emergency conditions. These capabilities include selective isolation of equipment and

segmentation of the grid into two or more smaller microgrids. The minimum degradation will be to disable the TMG controls and operate components in spot generation configurations.

4.6.Monitoring and Control. All types of TMG equipment shall provide status, configuration data, and control for individual equipment and the overall TMG to system operators. Access to this data and information shall be provided through human machine interfaces on the individual equipment and remote interfaces such as the Energy Informed Operations (EIO) dashboard or a standard notebook computer.

4.7.Cybersecurity. Cybersecurity is essential to the operation of the TMG. Although the ability to apply firmware/software upgrades is necessary, individual equipment shall be protected from the introduction of malicious firmware/software into the systems directly or through remote interfaces.

All types of TMG equipment shall conform to DOD Cybersecurity policy as specified in DODI 8500.01. Specific policy areas include: Risk Management, Operational Resilience, Integration and Interoperability, Cyberspace Defense, Performance, Identity Assurance, and Information Technology.

4.7.1. Risk Management Framework (RMF). Individual types of TMG equipment as well as the overall TMG, shall comply with DOD Risk Management Framework policy defined in DODI 8510.01. The RMF shall be based on the National Institute of Standards and Technology (NIST), RMF including Special Publication (SP) 800-39 Managing Information Security Risk: Organization, Mission, and Information System View, SP 800-53 Rev. 5 Security and Privacy Controls for Information Systems and Organizations, and SP 800-82 Rev. 2 Guide to Industrial Control Systems (ICS) Security. The RMF applies across the System Development Lifecycle (SDLC) of the individual items and how they communicate within the TMG. Applying standards-based cybersecurity as early as possible in the SDLC can also help improve its effectiveness at a reduced cost compared to waiting until later phases in the lifecycle.

4.7.2. Operational Resilience. Individual types of TMG equipment as well as the overall TMG, shall ensure information, data, and services are available to other elements of the TMG. Technology components (e.g., hardware and software) shall have the ability to reconfigure, optimize, and recover with defense in depth. Attempts made to reconfigure, and recover shall be logged in an audit file.

4.7.3. Access to firmware/software. Access control methods and procedures shall be implemented to mitigate cyber threats to individual equipment. Strong identities, assured access, trustworthy logging, and integrity-protected operations form the core list identified and defined within this document:

> Strong identities help maintain a trustworthy sense of what people, processes, and devices are taking action within the system.
>
> Assured access helps guarantee that authorized identities have appropriate access to system resources in order to, for example, initiate commands or collect telemetry.
>
> Trustworthy logging allows TMG Operators to operate and troubleshoot a running system and cyber analysts to perform forensic analysis after a cyber-event has been detected.

Integrity-protected operations ensure the trustworthiness of data and operations within the system and help assure that accidental and malicious changes to the system are detectable.

4.8. Tactical Microgrid Architecture. The components of the Tactical Microgrid are required to interface with one another through a common power bus and microgrid control system communications bus. The basic architecture is shown in Figure 1 Tactical Microgrid Schematic. All types of TMG equipment shall conform to the following architecture requirements.

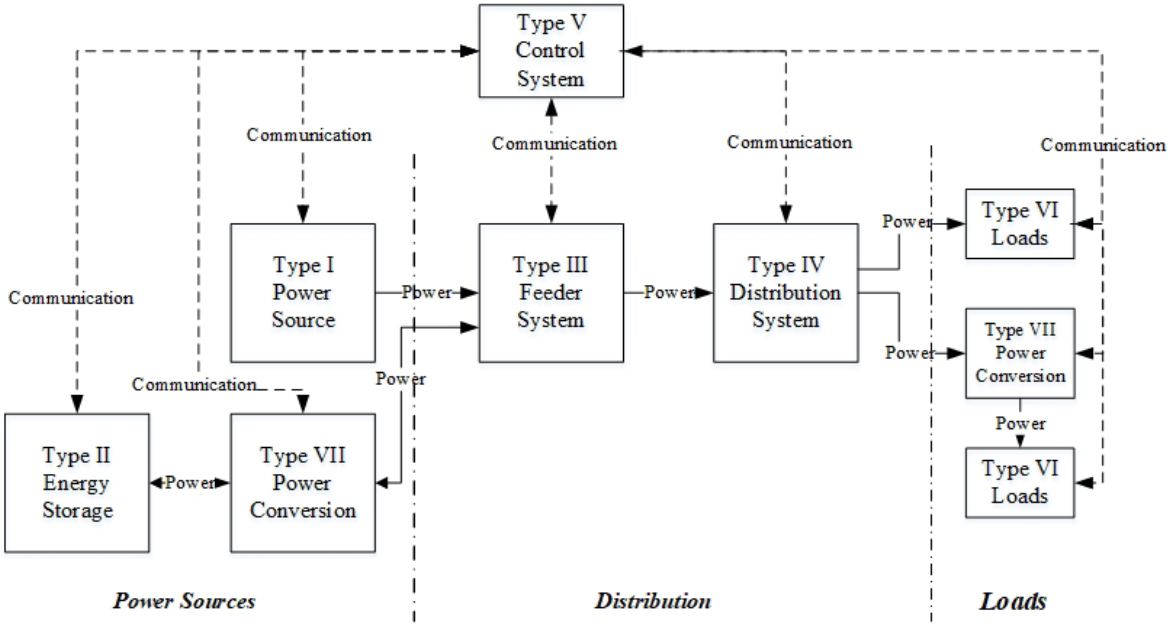


Figure 1 - Tactical Microgrid Schematic

4.8.1. Common Microgrid Control System. Type V Control Systems shall interface with other types of TMG equipment to provide automatic microgrid control of those devices. The control communications shall be accomplished over an OSI Layer 1 medium that is capable of Ethernet transmission. Standard connectors and cabling shall withstand environmental characteristics specified herein. The communication bus shall be capable of being disconnected without interrupting the operation of the TMG.

4.8.2. Common Microgrid Monitoring System. All types of TMG equipment shall have the capability to provide operational data to a central monitoring system using an OSI Layer 1 medium that is capable of Ethernet transmission. Standard connectors and cabling shall withstand environmental characteristics specified herein. The communication bus shall be capable of being disconnected without interrupting the operation of the TMG.

4.8.3. Levels of Control. The Tactical Microgrid equipment shall have internal controls and be capable of accepting TMG control from outside sources. The internal equipment controls enable the individual equipment to perform their functions independent from a TMG. The TMG control enables the primary functions of the TMG to be accomplished while connected to the individual components. In order to prevent control issues, the TMG level of control shall not override the

internal control functions of the equipment. TMG level control shall activate/deactivate local equipment controls to meet TMG functionality.

4.8.4. Tactical Microgrid Control (TMC) Redundancy. Electricity is mission critical in many TMG applications. The TMG must provide continuous, reliable and stable electricity to the user. Type V Control Systems equipment provide the TMC. To ensure TMG reliability and stability, there is a need to have redundant TMCs. Redundancy is accomplished by incorporating Master and Standby modes into the TMC. This redundancy enables a second TMCs, operating in standby mode, to maintain TMG function when the TMC in the master mode fails. The relationship between TMCs is:

- All TMCs receive data from all devices.
- Only the TMC in the master mode can send commands or setting changes.
- Devices reject commands from other TMCs.
- Failure of the TMC in the master mode shall trigger the automatic selection of a different TMC to operate in the master mode. TMCs in standby mode shall monitor the TMC in the master mode commands and setting changes in order to maintain a stable grid when they become active. The TMCs in standby mode will automatically take control using these commands and settings until new commands and settings become necessary based on operational data.

4.8.4.1. Protection coordination. All microgrid devices have some level of automated, threshold-based fault protection. Protection coordination sequences contactors and engages loads/sources for power system black-start, circuit partitioning, and circuit re-closing. Protection coordination also monitors power flows to prevent an over-current condition on a Type III feeder or Type IV distribution bus.

4.8.5. Communications Architecture. All devices operating on the Tactical Microgrid shall operate on a common Ethernet communications network. The selected communication protocols are shown in Table 4.1.

Table 4-1: Tactical Microgrid Communications Stack (TMCS)

Layer	TMS Implementation
Applications	Microgrid Control
Data Objects	TMCS Data Objects
Middleware	Data Distribution Service (DDS)
Transport	User Datagram Protocol (UDP)
Network	Internet Protocol Version 6 (IPv6)
Physical	Ethernet (802.3)

4.9.Data Model. The data model is defined in terms of data flows and data types. Data flows define the motion of messages while data types define the contents of messages, as some tactical microgrid functions require coordination between devices.

The TMS data model includes all messages sent among TMS devices. These messages are used to communicate all system health, status, command, and control information. They are also recommended for communications with external networks.

The data model provides a common language used by all TMS devices. It is platform independent to allow flexibility in device design and implementation. It contains detailed data flow and data type definitions to guarantee interoperability between devices, regardless of make or mode.

The data model allows for design and implementation flexibility while still taking into account such considerations as security, performance, simplicity, interoperability, and extensibility.

Tactical microgrid devices shall use this data model to implement all functions defined in this standard. Having a common data model is essential for interoperability between devices. Tactical microgrid devices may additionally use other communications to implement functions not defined in this standard. If such additional communications add functionality that is within the scope of this standard, then they should be proposed for inclusion in future revisions of this standard.

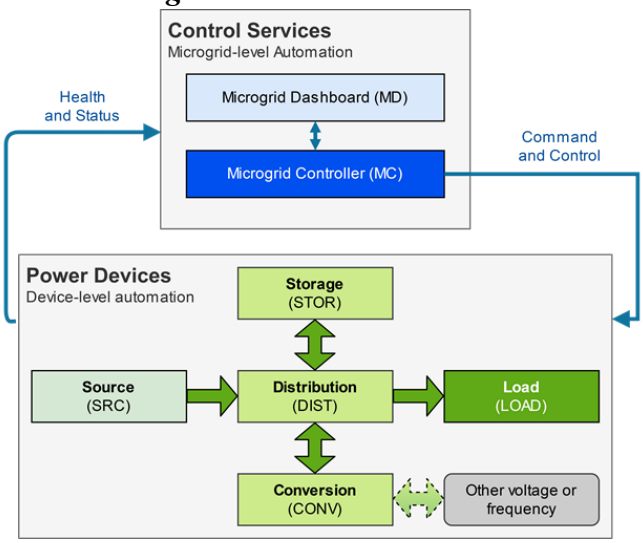
The roles and responsibilities for the TMG elements are defined in Section 5.3, with technical requirements defined in the appendix.

4.9.1. Device Identity. In the TMS data model, every device shall have a unique identity, assigned by an Identity, which is bound to a single, specific device role that has a number of built-in requirements and permissions. This approach simplifies the communications by preventing any ambiguity about how roles may interact. Under all circumstances, every TMS device role shall have its own unique Fingerprint. Since there will likely be multiple devices assigned to the same role, such as generators, the Fingerprint data type assigns a unique identity to each device, regardless of the device role. Data Types, such as Fingerprint, are discussed in Appendix Chapter A and Appendix Chapter B.

4.9.2. Device Roles. Device roles are defined building blocks within the TMS that provide modularized functionality. Each role is assigned based upon the device's capabilities in the TMS. Each device role is pre-defined in both functionality and power/communication interfaces to provide modular flexibility. Pre-defined components and interfaces allow customizable modularity in each TMS system to meet the unique needs of each microgrid installation. By using the device roles to define specific functionalities in the microgrid, any number of devices, regardless of capacity size, can be procured and assembled into a single TMS that meets the

mission needs. There are two types of device roles in TMS: Control Services and Power Devices. Figure 2 below, shows the relationship between these roles.

Figure 2 TMS Device Roles



4.9.3. Platforms. A platform is a group of TMS devices assembled into a larger physical package. Platforms may integrate multiple device roles. Each device on a platform shall have a separate identity and role. Multi-role products shall have separate identities and device roles for each type of functionality. Multi-role products shall communicate as if they were multiple physically-separate devices. It is recommended that each device role have its own data connection and run on a separate processor so each role can be physically isolated. If multiple devices are packaged into a single platform or multi-role product, they may share hardware or software components. If such sharing does occur, then all messages shall be sent separately for each contained device identity and role. Messages may be received singly and re-shared using internal mechanisms.

4.9.4. Data flows. TMS devices shall communicate through pre-defined data flows to share all operating information such as power commands and status. For simplicity, the device roles may be shown in the data flows throughout this document with abbreviated names - either P* for the Power Device role or C* for the Control Service device role - to represent any device in that category. Every data flow contains one or more topics. Each topic is defined with a name, publishers that send data, subscribers that receive data, a data type that specifies what values can be transferred, and timing that specifies when data is transferred. Data flows shall be defined using a publisher/subscriber (pub/sub) paradigm that defines the data flows in terms of logical connections. The underlying software and hardware may use buffers, filters, and a mix of broadcast, multicast, and point-to-point connections to implement the flow. These details are specified by the OMG DDS standard as used by TMS. A data flow containing a single topic is provided in Figure 3.

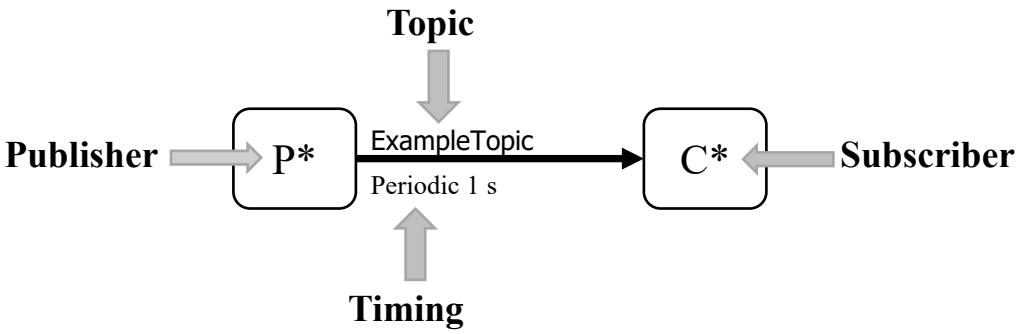


Figure 3 - Data Flow Notation

Each topic has the following key features:

Publisher is the Device Role(s) that sends these messages.

Subscriber is the Device Role(s) that receives these messages.

Topic is the name/identifier for these messages and the message flow.

Timing is the condition that causes messages to be sent and the reliability of the message flow.

4.9.5. Publishers and Subscribers. Device Roles shall be used to specify the source of the message, which is the publisher, and the receiver of the message, which is the subscriber, for each data flow. While a device role can be both a publisher and subscriber, typically in an individual flow it is one or the other. Conceptually, all publishers send all messages to all subscribers on a topic.

4.9.6. Topic and Message Flow. Topics shall be used to identify the data flows. Topics define a small group of TMS data flow message topics that have closely related values and match the publishers to the subscribers for each message chain. Multiple data flow topics are used together in larger patterns to achieve system-level TMS behaviors. Each data flow topic shall have a single unique data type with defined event trigger and reliability metrics, which are discussed below. The same data type shall be used by multiple data flow topics, providing a common connection between similar data flows.

The arrows in the data flow diagram indicate the flow of the message stream for the messages sent and received. The arrowhead indicates the direction that messages are being sent. The message stream for the data flows is published following specific data flow patterns that are pre-defined in the TMS. Arrows indicate messages that can be sent from both the initiating publisher and the subscriber in response to the publisher. Data flow patterns, including the subscriber response patterns, are discussed in further detail below.

4.9.7. Data Types. Devices shall use the TMS-defined data type that is assigned to each topic. The type identifies possible values for the messages where each message may have a different value. Platform-independent logical types are defined in Appendix Chapter B, Detailed Data Model Requirements. Platform-specific details such as data APIs and network serialization are defined in Appendix Chapter D, DDS Requirements.

4.9.8. Data Flow Patterns. The data flows identify and describe the system-level behaviors of TMS communication between devices through a set of pre-defined messages. There are three basic patterns of data flows that messages between devices follow: Status, Request/Response, and Custom.

4.9.8.1. Status data flow pattern. The status data flow messages shall provide status of the device that publishes the message. Figure 4 provides an example of a status data flow pattern. These messages do not require a reply or response from any subscribing devices. The timing of these messages shall occur when an event triggers the message to be sent. Examples of event triggers include when a device starts up or if an error condition occurs.

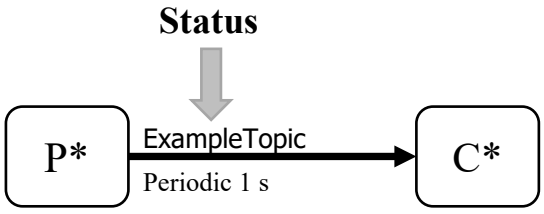


Figure 4 - Sample Status Data Flow Pattern

4.9.8.2. Request/Response data flow pattern. Request/Response messages shall require interaction between devices in the form of command and acknowledgment. Figure 5 provides an example of a request/response data flow pattern, where MC is the Microgrid Controller, P* is the Power Device role, and. MD is the Microgrid Dashboard.

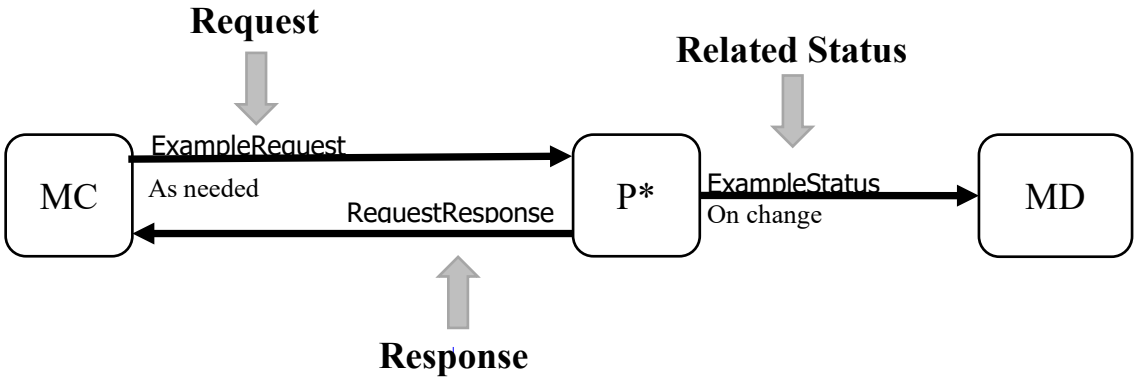


Figure 5 - Sample Request/Response Data Flow Pattern

A command is a request having immediate effect. Other requests may be deferred for later execution. A reply is a data sample update sent in response to a request.

4.9.8.3. Other responses include actions taken to complete a request. A Request is sent from one device to another, which responds to the request. Related status may be sent to a third device. An example of this data flow pattern is a health status request sent to a TMS device from the user interface to which the device responds and also sends additional related status. Custom data flow pattern. Custom messages shall have a unique interaction pattern that doesn't match the Status or Request/Response patterns. Each custom data flow pattern shall have

message interactions that are unique and specific to that one condition. Typically custom patterns define a TMS system-level behavior.

4.9.9. Instances. Some data flows have a single value that applies to the entire microgrid. Other data flows have different values for each device in the microgrid or even for each power connector in a single device. These different values are called sample instances. Editorial note: Working on a summary of the various key values and where they are used. This is being revised in the tech refresh.

4.9.10. Timing Effective communication requires a careful balance between sending messages frequently, so important information is not delayed, and limiting the frequency and size of messages, so resources are not overloaded. Each topic shall have timing requirements that define when new messages are sent, how frequently new messages can be sent, and when old messages are re-sent. These timing requirements shall be used to define behaviors and to derive minimum resource allocations for software and hardware in TMS devices. Topic data values change over time. Each new value is called a sample. When a publisher updates a sample, one or more messages will be sent to transmit the new value. There is not a direct, one-to-one mapping between sample updates and messages. For example, the communications software may bundle multiple samples into a single message, split a single sample across multiple messages, or re-transmit samples when messages are lost. Any such mapping is compliant as long as the overall timing requirements are met. The OMG DDS middleware allows the re-use of timing standards and software implementation. Timing requirements apply to the business logic in a TMS application, the OMG DDS middleware software library, operating system, and network hardware. There shall be four major timing requirements: data trigger, rate class, burst size, and quality of service (QoS). The first three requirements shall be defined in terms of sample updates. The QoS defines how these translate into messages. See Appendix Chapter A.8.5.

4.9.10.1. Event trigger. An event trigger is a pre-defined condition that causes sample updates. Fixed rate samples shall be triggered by a periodic timer set to a nominal rate. Variable rate samples shall be triggered by a given type of event. Details such as the event trigger for each topic type, rate limits, and timing parameters are defined in Appendix Chapter A, Data Model.

4.9.10.2. Rate limit The rate limit shall provide minimum separation time interval between sample updates. Upper bound on the rate of transmission and hence resource utilization. The publisher shall not send messages until after this minimum separation time interval. The network or subscriber may drop messages sent at a higher rate

4.9.10.4. Deadline. Deadline is the maximum separation time interval between sample updates. Lower bound on the rate of transmission and hence resource utilization. The publisher shall send updates before this maximum separation time interval. The subscriber may trigger an error-handling behavior if updates are not received on time.

4.9.10.5. Quality of Service (QoS). Quality of service is the policy defining how data samples are prioritized, buffered, sent, and re-sent as messages. Accounts for constraints

in network connectivity. A fully specified policy is called a profile. The QoS profile may include information from the rate limit and deadline.

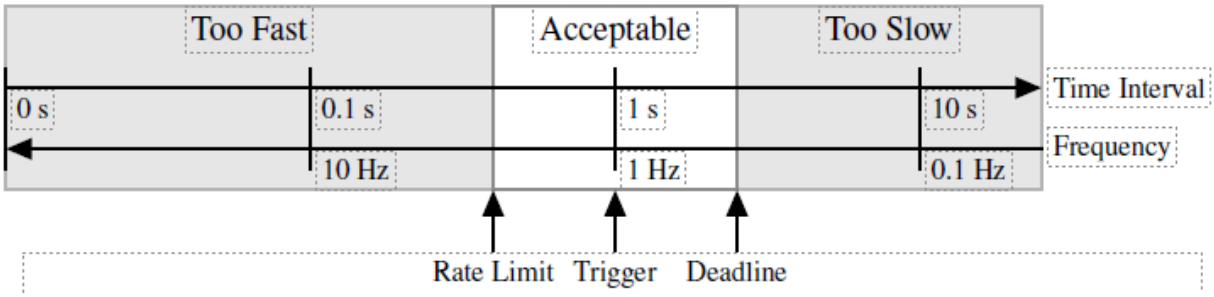


Figure 7 Example Timing for a Periodic 1s topic

5. DETAILED REQUIREMENTS

5.1.Safety The TMG interfaces required in this standard shall not cause harm to the Warfighter or damage equipment. Equipment developed in accordance with this standard shall comply with Army Regulation 385-10, The Army Safety Program. As a minimum, all interfaces shall meet various safety requirements as specified in MIL-STD-882 System Safety; NFPA-70 National Electric Code and NFPA-70E Standard for Electrical Safety in the Workplace.

5.2.Human Engineering The human interfaces required to implement the requirements specified herein, shall meet the design criteria requirements of MIL-STD-1472 Human Engineering.

5.3.TMG Equipment Roles and Responsibilities Each type of TMG equipment has specific roles and responsibilities when connected to the TMG. The roles and responsibilities define the functions of the item. There are related data flows in DDS which enable the functions to be performed. Detailed Data flows used to enable the equipment to meet their roles and responsibilities are defined in Appendix Chapter B. The roles and responsibilities for each type of TMG equipment are as follows:

5.3.1. Type I – Power Sources The role of the power source is to provide power to the grid. Power sources are responsible for producing quality electricity for the TM. Power sources shall accept commands from and provide operational parameters to Type V Control Systems in order to optimize microgrid operation. The power source shall provide power in accordance with its individual specification and shall be capable of accepting and implementing commands from Type V equipment either internal or external to the item.

5.3.1.1. Standard Power Source Functions The following standard power source functions and related commands shall be used to enable the power source role to be met:

Table 5-1: Type I Standard Power Source Functions

	Function	Reporting	Data Flows
a.	Discovery	Publish/Subscribe	B.5 Device Announcement B.20 Power Topology B.4 Heartbeat
b.	Start	Publish/Subscribe	B.16 Remote Start/Stop

Table 5-1: Type I Standard Power Source Functions

	Function	Reporting	Data Flows
c.	Feeder System Contactor Control	Publish/Subscribe	Special signal, see text.
d.	Parallel Operation	Publish/Subscribe	B.16 Remote Start/Stop B.21 AC Load Sharing B.22 DC Load Sharing B.14 Device Parameters
e.	Monitor Operational Parameters	Publish/Subscribe	B.8 Control Hardware B.9 Power Hardware B.12 Diagnostics Reporting B.18 AC Measurements B.19 DC Measurements
f.	Drop Load	Publish/Subscribe	B.21 AC Load Sharing B.22 DC Load Sharing
g.	Shutdown	Publish/Subscribe	B.16 Remote Start/Stop

a. Discovery. The discovery function enables the device to be recognized as a member of the TMG. This function initiates communication from the device to other members and determines acceptance and identification.

b. Start. The start function enables the TMG controller to direct the power source device to prepare to provide power to the TMG bus. The start function initiates the operation of a power source prior to connection to the TMG. This function varies with the type of power source. For example, with a standard engine driven generator set, this function starts the engine and ensures frequency and voltage are produced. Whereas, with a wind turbine, this function enables the wind turbine to turn and ensures frequency and voltage are produced.

c. Feeder System Contactor Control. This function is a safety mechanism used to prevent power from being fed back to the load terminals of a non-operating power source. It enables the power source to send activation/deactivation direction to Type III Feeder System contactor to connect the power system to the TMG bus. This is a special signal, with a dedicated direct connection between devices, and thus not part of the data model.

d. Parallel Operation. The parallel operation function enables the TMG controller to direct the power source to connect power to the TMG bus. The parallel operation function synchronizes the power source to the TMG, safely connects it to the TMG, and enables proper load sharing between power sources and the TMG.

e. Monitor Operational Parameters. The monitor operational parameters function enables the TMG controller to monitor the operation of the power source. The monitor operational parameters function provides the operational conditions of the power source to the Type V control systems. These parameters include but are not limited to:

voltage, frequency, current, total power, etc. In addition, any onboard health/diagnostics/prognostics or protective system indications shall also be reported.

f. Drop Load. The drop load function enables the TMG controller to direct the power source to disconnect power from the TMG bus as needed. Dropping load disconnects power source from the bus, but does not shutdown the power source. This function prevents the need to restart the power source if the power is needed on the grid.

g. Shutdown. The shutdown function enables the TMG controller to direct the power source to disconnect the load and shutdown the device. Shutdown shall stop an engine from operating or a wind turbine from rotating. The control systems for the power sources shall go into a standby mode where the TMG controller can maintain communication and send control commands.

5.3.2. Type II – Energy Storage The role of the energy storage systems (ESS) is to temporarily provide power during peak load demands or during operations when acoustic signature needs to be reduced. ESSs are responsible for storing excess energy and providing it back to the TMG as needed. ESSs shall provide power when the load exceeds current power production and additional power sources are needed to meet the load demand. ESSs shall accept commands from and provide operational parameters to Type V Control Systems in order to optimize microgrid operation. The ESS shall provide power in accordance with its individual specification and shall be capable of accepting and implementing commands from Type V equipment either internal or external to the item.

5.3.2.1. Standard Energy Storage System Functions The following standard energy storage system functions and related commands shall be used to enable the ESS role to be met:

Table 5-2: Type II Standard Energy Storage System Functions

	Function	Reporting	Data Flows
a.	Discovery	Publish/Subscribe	B.5 Device Announcement B.20 Power Topology B.4 Heartbeat
b.	Feeder System Contactor Control	Publish/Subscribe	Special signal, see text.
c.	Parallel Operation	Publish/Subscribe	B.16 Remote Start/Stop B.21 AC Load Sharing B.22 DC Load Sharing B.14 Device Parameters
d.	Monitor Operational Parameters	Publish/Subscribe	B.8 Control Hardware B.9 Power Hardware B.12 Diagnostics Reporting B.18 AC Measurements B.19 DC Measurements
e.	Drop Load	Publish/Subscribe	B.21 AC Load Sharing B.22 DC Load Sharing

a. Discovery. The discovery function enables the device to be recognized as a member of the TMG. This function initiates communication from the device to other members and determines acceptance and identification.

b. Feeder System Contactor Control. This function is a safety mechanism used to prevent power from being fed back to the load terminals of a non-operating ESS. It enables the ESS to send activation/deactivation direction to Type III Feeder System contactor to connect the ESS to the TMG bus.

c. Parallel Operation. The parallel operation function enables the TMG controller to direct the ESS to connect power to the TMG bus. The parallel operation function synchronizes the ESS to the TMG, safely connects it to the TMG, and enables proper load sharing between ESSs and the TMG bus.

d. Monitor Operational Parameters. The monitor operational parameters function enables the TMG controller to monitor the operation of the ESS. The monitor operational parameters function provides the operational conditions of the ESS to the Type V control systems. These parameters include but are not limited to: voltage, frequency, current, total power, etc. In addition, any onboard health/diagnostics/prognostics or protective system indications shall also be reported.

e. Drop Load. The drop load function enables the TMG controller to direct the power source to disconnect power from the TMG bus as needed. Dropping load disconnects the ESS from the bus, but does not shut down the ESS. This function prevents the need to restart the ESS if the power is needed on the grid.

5.3.3. Type III – Feeder Systems The role of feeder system is to establish the interconnected power and communications bus for the TMG. Feeder systems are responsible for providing electricity from the source to the load in a safe manner. Feeder systems shall accept power from the Type I Power Sources and Type II Energy Storage Systems and distribute it to the Type IV Distribution Systems and various standard and Type VI loads. Feeder systems shall accept commands from and provide operational parameters to Type V Control Systems in order to optimize microgrid operation. Feeder systems shall accept commands from Type I and Type II equipment to control power isolation contactors. The feeder system shall distribute power in accordance with its individual specification and shall be capable of accepting and implementing commands from Type V equipment either internal or external to the item.

In addition to the power interface, the Type III feeders systems will function as the communications hub for the TMG. Each power connection will have a matching communications port to enable all of the TMG elements to communicate with each other.

5.3.3.1. Standard Feeder System Functions The following standard feeder system functions and related commands shall be used to enable the feeder system role to be met:

Table 5-3: Type III Feeder System Functions

	Function	Reporting	**Data Flows**
a.	Discovery	Publish/Subscribe	B.5 Device Announcement B.20 Power Topology B.4 Heartbeat

Table 5-3: Type III Feeder System Functions

	Function	Reporting	**Data Flows**
b.	Feeder System Contactor Control	Publish/Subscribe	Special signal, see text.
c.	Close Circuit	Publish/Subscribe	B.17 Power Switches and Connectors
d.	Monitor Operational Parameters	Publish/Subscribe	B.8 Control Hardware B.9 Power Hardware B.12 Diagnostics Reporting B.18 AC Measurements B.19 DC Measurements
e.	Open Circuit	Publish/Subscribe	B.17 Power Switches and Connectors

a. Discovery. The discovery function enables the device to be recognized as a member of the TMG. This function initiates communication from the device to other members and determines acceptance and identification.

b. Feeder System Contactor Control. This function is a safety mechanism used to prevent power from being fed back to the load terminals of a non-operating power source or ESS. It enables the feeder system to receive contactor activation/deactivation direction from the Type I and Type II equipment to connect power to the TMG bus.

c. Close Circuit. The close circuit function enables the TMG controller to direct the feeder system to connect power from the power bus to the individual power circuits.

d. Monitor Operational Parameters. The monitor operational parameters function enables the TMG controller to monitor the operation of the feeder system. The monitor operational parameters function provides the operational conditions of the feeder system to the Type V control systems. These parameters include but are not limited to: voltage, frequency, current, total power, state of circuit (open/closed/tripped), etc. In addition, any onboard health/diagnostics/prognostics or protective system indications shall also be reported.

e. Open Circuit. The open circuit function enables the TMG controller to direct the feeder system to disconnect power from the individual power circuits.

5.3.4. Type IV – Distribution Systems The role of distribution systems is to provide the power interface from the Type III feeder system to the load and function as the communications hub for smart load to communicate with the other elements of the TMG. Distribution systems are responsible of distributing electricity from the TMG bus, through individual branch circuits, to the loads. Distribution systems shall divide the TMG power bus into individual branch circuits and shall provide the final overcurrent protection of those circuits to meet load requirements. Distribution Systems shall accept commands from and provide operational parameters to Type V Control Systems in order to optimize microgrid operation. The distribution system shall

distribute power in accordance with its individual specification and shall be capable of accepting and implementing commands from Type V equipment either internal or external to the item.

In addition to the power interface, the Type IV distribution systems will function as the communications hub for the smart loads and of elements of the TMG. Each power connection will have a matching communications port to enable all of the smart loads to communicate with the other elements of the TMG.

5.3.4.1. Standard Distribution System Functions The following standard distribution system functions and related commands shall be used to enable the distribution system role to be met:

Table 5-4: Type IV Standard Distribution System Functions

	Function	Reporting	Data Flow
a.	Discovery	Publish/Subscribe	B.5 Device Announcement
b.	Close Circuit	Publish/Subscribe	B.17 Power Switches and Connectors
c.	Monitor Operational Parameters	Publish/Subscribe	B.8 Control Hardware B.9 Power Hardware B.12 Diagnostics Reporting B.18 AC Measurements B.19 DC Measurements
d.	Open Circuit	Publish/Subscribe	B.17 Power Switches and Connectors

a. Discovery. The discovery function enables the device to be recognized as a member of the TMG. This function initiates communication from the device to other members and determines acceptance and identification.

b. Close Circuit. The close circuit function enables the TMG controller to direct the feeder system to connect power from the power bus to the individual branch circuits.

c. Monitor Operational Parameters. The monitor operational parameters function enables the TMG controller to monitor the operation of the distribution system. The monitor operational parameters function provides the operational conditions of the distribution system to the Type V control systems. These parameters include but are not limited to: voltage, frequency, current, total power, state of circuit (open/closed/tripped), etc. In addition, any onboard health/diagnostics/prognostics or protective system indications shall also be reported.

d. Open Circuit. The open circuit function enables the TMG controller to direct the feeder system to disconnect power from the individual branch circuits.

5.3.5. Type V – Control Systems The role of the control system is to communicate with all of the TMG equipment in order to optimize TMG operation. The control system is responsible for controlling the TMG membership; identifying each member; receiving operational data from each member; analyzing the data; and providing commands back to the members in order to meet optimization goals. In addition, the control system is responsible for providing the user interface in order to control the TMG. The control system shall utilize the commands specified

herein to control the functions of the individual equipment of the TMG. The control system shall provide monitoring and controls in accordance with its individual specification and be capable of accepting data and initiating commands to all TMG equipment. Standard Control System Source Functions The following standard control system functions and related commands shall be used to enable the TMG optimization role to be met:

Table 5-5: Type V Standard Control System Functions

	Function	Reporting	Data Flows
a.	Discovery	Publish/Subscribe	B.5 Device Announcement B.20 Power Topology B.4 Heartbeat
b.	Start	Publish/Subscribe	B.16 Remote Start/Stop
c.	Parallel Operation	Publish/Subscribe	B.16 Remote Start/Stop B.21 AC Load Sharing B.22 DC Load Sharing B.14 Device Parameters
d.	Close Circuit	Publish/Subscribe	B.17 Power Switches and Connectors
e.	Monitor Operational Parameters	Publish/Subscribe	B.12 Diagnostics Reporting B.18 AC Measurements B.19 DC Measurements B.8 Control Hardware B.9 Power Hardware
f.	Auto Restart	Publish/Subscribe	B.16 Remote Start/Stop B.17 Power Switches and Connectors
g.	Open Circuit	Publish/Subscribe	B.17 Power Switches and Connectors B.21 AC Load Sharing B.22 DC Load Sharing
h.	Drop Load	Publish/Subscribe	B.21 AC Load Sharing B.22 DC Load Sharing
i.	Shutdown	Publish/Subscribe	B.16 Remote Start/Stop

a. Discovery. The discovery function enables the device to be recognized as a member of the TMG. This function initiates communication from the device to other members and determines acceptance and identification.

b. Start. The start function enables the TMG controller to direct the power source device to prepare to provide power to the TMG bus. The start function initiates the operation of a power source prior to connection to the TMG. This function varies with the type of power source. For example, with a standard engine driven generator set, this function turns starts the engine and ensures frequency and voltage are produced. Whereas, with a wind turbine, this function enables the wind turbine to turn and ensures frequency and voltage are produced.

c. Parallel Operation. The parallel operation function enables the TMG controller to direct the power source to connect power to the TMG bus. The parallel operation

function synchronizes the power source to the TMG, safely connects power source to the TMG, and enables proper load sharing between power sources.

d. Close Circuit. The close circuit function enables the TMG controller to direct the feeder system to connect power from the power bus to the individual branch circuits.

e. Monitor Operational Parameters. The monitor operational parameters function enables the TMG controller to monitor the operation of the power source. The monitor operational parameters function provides the operational conditions of the power source to the Type V control systems. These parameters include but are not limited to voltage, frequency, current, total power, etc. In addition, any onboard health/diagnostics/prognostics or protective system indications shall also be reported.

f. Auto Restart. The auto restart function provides for a controlled restart of the TMG if the entire grid goes down. The controller shall attempt to restart the grid in an orderly fashion unless it cannot be restored within a fixed amount of time.

g. Open Circuit. The open circuit function enables the TMG controller to direct the feeder system to disconnect power from the individual branch circuits.

h. Drop Load. The drop load function enables the TMG controller to direct the power source to disconnect power from the TMG bus as needed. Dropping load disconnects the load from the bus, but does not shutdown the power source. This function prevents the need to restart the power source if the power is needed on the grid.

i. Shutdown. The shutdown function enables the TMG controller to direct the power source to disconnect the load and shutdown the device. Shutdown shall stop an engine from operating or a wind turbine from rotating. The control systems for the power sources shall go into a standby mode where the TMG controller can maintain communication and send control commands.

5.3.5.2. Operator Interface Function Type V Control systems shall provide the operator interface for the TMG. The operator shall have the ability to monitor the TMG, define the control parameters, and take action to correct various issues with the TMG through the control system. This interface shall be accomplish as specified in Appendix Chapter B.

5.3.6. Type VI – Loads Type VI loads are power consumers which comply with the communications requirements of this standard. The role of Type VI loads is to use the power from the TMGs to meet mission needs. The responsibility of the TMS compliant loads is to inform the Type V control system of intent to activate and use electricity or deactivate if already operating. These loads shall communicate with the Type V control systems to announce their intent to activate or deactivate to allow the Type V control system to ensure enough power is available on the grid. Loads accept commands from the Type V Control Systems in order to optimize microgrid operation. The load shall consume power in accordance with its individual specification and be capable of accepting commands from Type V equipment either internal or external to the item.

Non TMS compliant loads can be connected to the TMG, but the control functions are implemented through the Type III Feeder and Type IV Distribution systems. The control systems shall connect and disconnect these loads by activating/deactivating the circuits where they are connected. No direct communication with these loads is necessary.

5.3.6.1. Standard Load Functions The following standard load functions and related commands shall be used to enable the load role to be met:

Table 5-6: Type VI Standard Load System Functions

	Function	Reporting	Data Flows
a.	Discovery	Publish/Subscribe	B.5 Device Announcement B.20 Power Topology B.4 Heartbeat
b.	Close Circuit	Publish/Subscribe	B.17 Power Switches and Connectors B.21 AC Load Sharing B.22 DC Load Sharing
c.	Monitor Operational Parameters	Publish/Subscribe	B.12 Diagnostics Reporting B.18 AC Measurements B.19 DC Measurements B.8 Control Hardware B.9 Power Hardware
d.	Open Circuit	Publish/Subscribe	B.17 Power Switches and Connectors B.21 AC Load Sharing B.22 DC Load Sharing

a. Discovery. The discovery function enables the device to be recognized as a member of the TMG. This function initiates communication from the device to other members and determines acceptance and identification.

b. Close Circuit. The close circuit function enables the TMG controller to direct the load to connect to the TMG.

c. Monitor Operational Parameters. The monitor operational parameters function enables the load to provide various operational parameters and equipment health status to the TMG controller.

d. Open Circuit. The open circuit function enables the TMG controller to direct the feeder system to disconnect power from the individual branch circuits.

5.3.7. Type VII – Power Conversion The role of Power Conversion equipment is the enable Type I power sources or Type II energy storage systems with different voltages and currents to connect to the TMG. Power conversion equipment shall be required to provide operational parameters and accept commands from the Type V Control Systems in order to optimize microgrid operation. The power conversion systems provide power in accordance with its individual specification and be capable of accepting commands from Type V equipment either internal or external to the item.

5.3.7.1. Standard Power Conversion Functions Since power conversion equipment must function with Type I power sources or Type II energy storage systems, the standard functions shall be the same standard functions as the Type I and Type II equipment. The following

standard power source functions and related commands shall be used to enable the power conversion role to be met:

Table 5-7: Type VII Standard Power Conversion Functions

	Function	Reporting	Data Flows
a.	Discovery	Publish/Subscribe	B.5 Device Announcement B.20 Power Topology B.4 Heartbeat
b.	Start	Publish/Subscribe	B.17 Power Switches and Connectors
c.	Feeder System Contactor Control	Publish/Subscribe	B.17 Power Switches and Connectors
d.	Parallel Operation	Publish/Subscribe	B.16 Remote Start/Stop B.21 AC Load Sharing B.22 DC Load Sharing B.14 Device Parameters
e.	Monitor Operational Parameters	Publish/Subscribe	B.8 Control Hardware B.9 Power Hardware B.12 Diagnostics Reporting B.18 AC Measurements B.19 DC Measurements
f.	Drop Load	Publish/Subscribe	B.21 AC Load Sharing B.22 DC Load Sharing
g.	Shutdown	Publish/Subscribe	B.16 Remote Start/Stop

a. Discovery. The discovery function enables the device to be recognized as a member of the TMG. This function initiates communication from the device to other members and determines acceptance and identification.

b. Start. The start function enables the TMG controller to direct the power source device to prepare to provide power to the TMG bus. The start function initiates the operation of a power source prior to connection to the TMG. This function varies with the type of power source. For example, with a standard engine driven generator set, this function turns starts the engine and ensures frequency and voltage are produced. Whereas, with a wind turbine, this function enables the wind turbine to turn and ensures frequency and voltage are produced.

c. Feeder System Contactor Control. This function is a safety mechanism used to prevent power from being fed back to the load terminals of a non-operating power source. It enables the power source to send activation/deactivation direction to Type III Feeder System contactor to connect the power system to the TMG bus.

d. Parallel Operation. The parallel operation function enables the TMG controller to direct the power source to connect power to the TMG bus. The parallel operation function synchronizes the power source to the TMG, safely connects it to the TMG, and enables proper load sharing between power sources and the TMG.

e. Monitor Operational Parameters. The monitor operational parameters function enables the TMG controller to monitor the operation of the power source. The monitor

operational parameters function provides the operational conditions of the power source to the Type V control systems. These parameters include but are not limited to: voltage, frequency, current, total power, etc. In addition, any onboard health/diagnostics/prognostics or protective system indications shall also be reported.

f. Drop Load. The drop load function enables the TMG controller to direct the power source to disconnect power from the TMG bus as needed. Dropping load disconnects power source from the bus, but does not shutdown the power source. This function prevents the need to restart the power source if the power is needed on the grid.

g. Shutdown. The shutdown function enables the TMG controller to direct the power source to disconnect the load and shutdown the device. Shutdown shall stop an engine from operating or a wind turbine from rotating. The control systems for the power sources shall go into a standby mode where the TMG controller can maintain communication and send control commands.

5.4.Communications Interface The communications interface (CI) is how the communications and controls will be transmitted from one device to another within the TMG. The CI is described in terms of the Information technology - Open Systems Interconnection (OSI) model as defined in ISO/IEC 7498 series. The model contains various layers that define the communications. Table 5-8 Tactical Microgrid Communications Stack defines the various layers and protocols within the layer used in the TMG.

Table 5-8: Tactical Microgrid Communications Stack

OSI Layer	TMS Implementation
Application	Device Controller
Presentation	TMS Data Model
Session	OMG Data Distribution Service (DDS)
Transport	User Datagram Protocol (UDP)
Network	Internet Protocol Version 6 (IPv6)
Physical	Ethernet (802.3)

5.4.1. Application Layer. The application layer enables each device controller to share information with other devices in the TMG to make decisions and control the TMG. Each type of device may have its own implementation.

5.4.2. Presentation Layer. The data model provides a common language for communication between TMG devices. Data flows define the movement of information and data types define the range of possible values.

5.4.3. Session Layer. The OMG Data Distribution Service (OMG DDS) provides portable APIs for sending and receiving information. The DDS-RTPS provides interoperable quality of service over transport layer.

5.4.4. Transport Layer. The User Datagram Protocol (UDP) provides efficient communications over the network. Reliability is handled by OMG DDS as required.

5.4.5. Network Layer. The network layer provides the functional and procedural means for connectionless-mode or connection mode transmission among transport-entities and, therefore, provides to the transport-entities independence of routing and relay considerations. The network layer shall use Internet Protocol Version 6 (IPv6) IAW RFC8200 (STD86).

5.4.6. Physical Layer. The physical layer provides the mechanical, electrical, functional, and procedural means to activate, maintain, and de-activate physical-connections for bit transmission between data-link-entities. The physical layer shall be Ethernet (802.3) based IAW IEEE 802.3. It is recommended that each TMG item have two connections point to allow for multiple items to be daisy chained.

6. NOTES

(This section contains information of a general or explanatory nature that may be helpful but is not mandatory.)

6.1. Intended Use. Standards covered by this standard are intended for use in acquisition to establish requirements for military-unique communication interfaces for Tactical Microgrid equipment.

6.2. Acquisition requirements. Acquisition documents should specify the title, number, and date of this standard.

6.3. Tailoring guidance To ensure proper application of this standard, invitation for bids, request for proposals, and contractual statements of work should tailor the requirements in sections 4 and 5 of this standard to exclude any unnecessary requirements. For example, if the statement of work requires the development of a revised standard, then all material related to notices should be excluded.

6.4. Subject term (key word) Listing.

Tactical Microgrid Standard
Energy Storage
Generator
Feeder System
Load

6.5. Changes from previous issue.

Marginal notations are not used in this change to identify changes with respect to the previous issue due to the extent of the changes.

The Data Model for the Tactical Microgrid Standard

This Appendix is a mandatory part of the standard. The information contained herein is intended for compliance.

CONTENTS

PARAGRAPH PAGE

A General Data Model Requirements 49
A.1 Introduction . . . 49
A.2 Scope and Purpose . . . 49
A.3 Implementation Considerations . . . 50
A.4 General Requirements . . . 51
A.5 Device Identity . . . 51
A.6 Device Roles . . . 51
A.6.1 Control Devices . . . 52
A.6.2 Power Devices . . . 52
A.6.3 Platforms and Multi-Role Products . . . 53
A.7 Microgrid Functions . . . 54
A.7.1 Power Balance . . . 55
A.7.2 Energy Dispatch . . . 56
A.7.3 Operator Control . . . 56
A.7.4 Consistency . . . 57
A.7.5 Microgrid Controller Selection . . . 59
A.8 Data Flows . . . 61
A.8.1 Publishers and Subscribers . . . 62
A.8.2 Topics, Data Samples, and Messages . . . 62
A.8.3 Data Flow Patterns . . . 63
A.8.3.1 State Pattern . . . 63
A.8.3.2 Request/Reply Pattern . . . 63
A.8.3.3 Request/Reply with State Pattern . . . 63
A.8.3.4 Custom Pattern . . . 64
A.8.4 Instances and Key Values . . . 64
A.8.5 Timing . . . 65
A.8.5.1 Data Triggers . . . 67
A.8.5.2 Rate Class . . . 68
A.8.5.3 Burst Size . . . 70
A.8.5.4 Quality of Service (QoS) . . . 71
A.8.5.5 Order of Delivery . . . 73
A.8.6 Topic Naming Conventions . . . 74
A.8.7 Topic Usage . . . 75
A.9 Data Types . . . 76
A.9.1 Type Hierarchy . . . 76

CONTENTS

PARAGRAPH		PAGE
A.9.2	Type Names	77
A.9.3	Type Namespaces	77
A.9.4	Units of Measure	79
A.9.5	Power Measurement Sign Conventions	80
A.9.6	Primitive Data Types	82
A.9.6.1	Boolean	82
A.9.6.2	Character	82
A.9.6.3	Octet	83
A.9.6.4	Integer Numbers	83
A.9.6.5	Floating-Point Numbers	84
A.9.6.6	Enumeration	84
A.9.7	Constructed Data Types	85
A.9.7.1	Structure	85
A.9.7.2	Array	85
A.9.7.3	Sequence	85
A.9.7.4	String	86
A.9.7.5	Optional	86
A.9.7.6	Typedef	86
A.9.8	Domain-Specific Data Types	87
A.9.9	Constants	87
A.9.10	Data Accuracy and Completeness	87
A.9.10.1	Derived Values	87
A.9.10.2	Default Values	88
A.9.11	Annotations	88
A.9.11.1	Extensibility	88
A.9.11.2	Topic Usage	89
A.9.11.3	Key Values	89
A.9.11.4	Bounded Values	89
A.9.11.5	Units of Measure	90
A.10	Resource Allocation	90
A.11	Data Model Extensions	91
A.12	Change History	92
B	**Detailed Data Model Requirements**	**93**
B.1	Device Role Details	96
B.1.1	Control Device Roles	96
B.1.1.1	Microgrid Controller (MC) Logic	96
B.1.1.2	Microgrid Dashboard (MD) Logic	98
B.1.1.3	Monitor (MON) Logic	100
B.1.2	Power Device Roles	101
B.1.2.1	Device Controller Logic	101
B.1.2.2	Power Port Components	103

CONTENTS

PARAGRAPH PAGE
B.1.2.3 Source (SRC) Components . . . 104
B.1.2.4 Storage (STOR) Components . . . 107
B.1.2.5 Load (LOAD) Components . . . 109
B.1.2.6 Distribution (DIST) Components . . . 111
B.1.2.7 Conversion (CONV) Components . . . 113
B.2 Request/Response Pattern . . . 116
B.2.1 Data Flows . . . 117
B.2.2 Data Types . . . 118
B.2.2.1 `tms::Reply` . . . 118
B.2.2.2 `tms::Identity` . . . 120
B.2.2.3 `tms::ConfigId` . . . 120
B.2.2.4 `tms::PowerPortNumber` . . . 121
B.2.2.5 `tms::RequestSequence` . . . 121
B.2.2.6 `tms::ReplyStatus` . . . 122
B.2.2.7 `tms::ReplyCode` . . . 122
B.2.2.8 `tms::String32` . . . 124
B.3 Identity . . . 124
B.3.1 Data Types . . . 124
B.3.1.1 `tms::Identity` . . . 124
B.4 Heartbeat . . . 124
B.4.1 Data Flows . . . 125
B.4.2 Data Types . . . 126
B.4.2.1 `tms::Heartbeat` . . . 126
B.4.2.2 `tms::Identity` . . . 126
B.5 Device Announcement . . . 126
B.5.1 Data Flows . . . 127
B.5.2 Data Types . . . 128
B.5.2.1 `tms::DeviceInfo` . . . 129
B.5.2.2 `tms::Identity` . . . 130
B.5.2.3 `tms::DeviceRole` . . . 130
B.5.2.4 `tms::ProductInfo` . . . 131
B.5.2.5 `tms::NatoStockNumber` . . . 132
B.5.2.6 `tms::GlobalTradeItemNumber` . . . 132
B.5.2.7 `tms::String32` . . . 133
B.5.2.8 `tms::TopicInfo` . . . 133
B.5.2.9 `tms::TopicList` . . . 134
B.5.2.10 `tms::TopicName` . . . 134
B.5.2.11 `tms::ControlHardwareInfo` . . . 134
B.5.2.12 `tms::PowerHardwareInfo` . . . 135
B.5.2.13 `tms::EngineInfo` . . . 136
B.5.2.14 `tms::FuelInfo` . . . 138
B.5.2.15 `tms::GeneratorInfo` . . . 138

CONTENTS

PARAGRAPH		PAGE
B.5.2.16	tms::EnergyStorageInfo	139
B.5.2.17	tms::PowerElectronicsInfo	140
B.5.2.18	tms::ThermalInfo	141
B.5.2.19	tms::ThermalZoneSequence	141
B.5.2.20	tms::String1-32	141
B.5.2.21	tms::SynchronousMachineCoefficients	142
B.5.2.22	tms::ParameterMetadataSequence	144
B.5.2.23	tms::ParameterMetadata	145
B.5.2.24	tms::String16	146
B.5.2.25	tms::EnumLabelSequence	147
B.5.2.26	tms::ControlServiceInfo	147
B.5.2.27	tms::MicrogridControllerInfo	147
B.5.2.28	tms::MicrogridControllerFeatureSequence	148
B.5.2.29	tms::MicrogridControllerFeature	148
B.5.2.30	tms::MicrogridDashboardInfo	149
B.5.2.31	tms::MicrogridDashboardFeatureSequence	149
B.5.2.32	tms::MicrogridDashboardFeature	149
B.5.2.33	tms::PowerDeviceInfo	150
B.5.2.34	tms::PowerPortInfoSequence	151
B.5.2.35	tms::PowerPortInfo	151
B.5.2.36	tms::PowerPortNumber	153
B.5.2.37	tms::CircuitWiring	153
B.5.2.38	tms::PowerPortDirectionality	154
B.5.2.39	tms::PowerConnectorInfo	154
B.5.2.40	tms::PowerConnectorFeatureSequence	155
B.5.2.41	tms::PowerConnectorFeature	155
B.5.2.42	tms::PowerConnectorType	156
B.5.2.43	tms::PowerConnectorPolarity	157
B.5.2.44	tms::PowerSwitchInfo	157
B.5.2.45	tms::PowerSwitchFeatureSequence	158
B.5.2.46	tms::PowerSwitchFeature	158
B.5.2.47	tms::GroundingInfoSequence	159
B.5.2.48	tms::GroundingInfo	160
B.5.2.49	tms::GroundingCircuitNumber	161
B.5.2.50	tms::GroundingDesignType	161
B.5.2.51	tms::PowerPortNumberSequence	162
B.5.2.52	tms::ConversionInfo	162
B.5.2.53	tms::ConversionFeatureSequence	162
B.5.2.54	tms::ConversionFeature	163
B.5.2.55	tms::ActiveConversionInfo	163
B.5.2.56	tms::LoadSharingInfoSequence	164
B.5.2.57	tms::LoadSharingInfo	164

CONTENTS

PARAGRAPH			PAGE
B.5.2.58		tms::Curve2D	165
B.5.2.59		tms::Point2DSequence	167
B.5.2.60		tms::Point2D	167
B.5.2.61		tms::LoadStepResponseSequence	167
B.5.2.62		tms::LoadStepResponse	168
B.5.2.63		tms::PassiveConversionInfo	169
B.5.2.64		tms::PowerPortConversionInfoSequence	170
B.5.2.65		tms::PowerPortConversionInfo	170
B.5.2.66		tms::ConversionTapInfoSequence	170
B.5.2.67		tms::ConversionTapInfo	171
B.5.2.68		tms::TapNumber	171
B.5.2.69		tms::DistributionInfo	172
B.5.2.70		tms::DistributionFeatureSequence	172
B.5.2.71		tms::DistributionFeature	172
B.5.2.72		tms::SourceInfo	173
B.5.2.73		tms::SourceFeatureSequence	173
B.5.2.74		tms::SourceFeature	174
B.5.2.75		tms::EnergyStartStopLevelSequence	174
B.5.2.76		tms::EnergyStartStopLevel	175
B.5.2.77		tms::StorageInfo	176
B.5.2.78		tms::StorageFeatureSequence	176
B.5.2.79		tms::StorageFeature	177
B.5.2.80		tms::LoadInfo	177
B.5.2.81		tms::LoadFeatureSequence	178
B.5.2.82		tms::LoadFeature	178
B.6	Nickname		178
B.6.1	Data Flows		178
B.6.2	Data Types		180
B.6.2.1		tms::IdentityNicknameState	181
B.6.2.2		tms::IdentityNicknameRequest	181
B.6.2.3		tms::Reply	182
B.6.2.4		tms::Identity	182
B.6.2.5		tms::String32	182
B.6.2.6		tms::DeviceRequest	182
B.6.2.7		tms::RequestSequence	182
B.6.2.8		tms::ConfigId	182
B.6.2.9		tms::PowerPortNumber	182
B.6.2.10		tms::ReplyStatus	182
B.6.2.11		tms::ReplyCode	183
B.7	Device Icon		183
B.7.1	Data Flows		183
B.7.2	Data Types		184

CONTENTS

PARAGRAPH			PAGE
B.7.2.1		tms::DeviceIcon	184
B.7.2.2		tms::Identity	185
B.7.2.3		tms::String32	185
B.7.2.4		tms::OctetSequence	185
B.8	Control Hardware		185
B.8.1	Data Flows		185
B.8.2	Data Types		186
B.8.2.1		tms::ControlHardwareUpdate	187
B.8.2.2		tms::Identity	188
B.8.2.3		tms::ClockMonotonic	188
B.9	Power Hardware		189
B.9.1	Data Flows		189
B.9.2	Data Types		190
B.9.2.1		tms::PowerHardwareUpdate	190
B.9.2.2		tms::Identity	191
B.9.2.3		tms::ClockMonotonic	191
B.9.2.4		tms::EngineState	191
B.9.2.5		tms::EngineOperatingState	192
B.9.2.6		tms::FuelState	193
B.9.2.7		tms::GeneratorState	194
B.9.2.8		tms::EnergyStorageState	194
B.9.2.9		tms::StorageCellStateSequence	195
B.9.2.10		tms::StorageCellState	196
B.9.2.11		tms::StorageCellNumber	197
B.9.2.12		tms::PowerElectronicsState	197
B.9.2.13		tms::ThermalState	198
B.9.2.14		tms::ThermalLoadSequence	198
B.10	Energy Storage		198
B.10.1	Data Flows		198
B.10.2	Data Types		199
B.10.2.1		tms::StorageUpdate	200
B.10.2.2		tms::Identity	201
B.10.2.3		tms::Curve2D	201
B.10.2.4		tms::Point2DSequence	201
B.10.2.5		tms::Point2D	201
B.11	Operator Input		202
B.11.1	Data Flows		202
B.11.2	Data Types		203
B.11.2.1		tms::OperatorIntentState	204
B.11.2.2		tms::OperatorIntentRequest	205
B.11.2.3		tms::Identity	205
B.11.2.4		tms::OperatorIntent	206

CONTENTS

PARAGRAPH PAGE
B.11.2.5 tms::GridRequest . . . 206
B.11.2.6 tms::OperatorIntentType . . . 207
B.11.2.7 tms::MicrogridIntent . . . 207
B.11.2.8 tms::OperatingMode . . . 208
B.11.2.9 tms::LoadSharingIntent . . . 209
B.11.2.10 tms::OperatorPriority . . . 210
B.11.2.11 tms::OperatorPriorityType . . . 210
B.11.2.12 tms::StorageIntent . . . 211
B.11.2.13 tms::ClockMonotonic . . . 211
B.11.2.14 tms::IdentitySequence . . . 212
B.11.2.15 tms::DeviceIntentSequence . . . 212
B.11.2.16 tms::DeviceIntent . . . 212
B.11.2.17 tms::PowerPortIntentSequence . . . 213
B.11.2.18 tms::PowerPortIntent . . . 213
B.11.2.19 tms::PowerPortNumber . . . 213
B.11.2.20 tms::RequestSequence . . . 213
B.12 Diagnostics Reporting . . . 213
B.12.1 Data Flows . . . 214
B.12.2 Data Types . . . 215
B.12.2.1 tms::ActiveDiagnosticState . . . 216
B.12.2.2 tms::Identity . . . 216
B.12.2.3 tms::DiagnosticTroubleCodeSequence . . . 216
B.12.2.4 tms::DiagnosticTroubleCode . . . 217
B.12.2.5 tms::PowerPortNumber . . . 218
B.12.2.6 tms::SuspectParameterNumber . . . 219
B.12.2.7 tms::FailureModeIndicator . . . 221
B.12.2.8 tms::ClockMonotonic . . . 222
B.12.2.9 tms::DtcSeverity . . . 223
B.12.2.10 tms::String32 . . . 223
B.13 Microgrid Controller Selection . . . 223
B.13.1 Data Flows . . . 224
B.13.2 Data Types . . . 225
B.13.2.1 tms::ActiveMicrogridControllerState . . . 225
B.13.2.2 tms::Identity . . . 225
B.14 Device Parameters . . . 225
B.14.1 Data Flows . . . 226
B.14.2 Data Types . . . 228
B.14.2.1 tms::ControlParameterState . . . 229
B.14.2.2 tms::ControlParameterRequest . . . 229
B.14.2.3 tms::Reply . . . 230
B.14.2.4 tms::MetricParameterState . . . 230
B.14.2.5 tms::Identity . . . 230

CONTENTS

PARAGRAPH PAGE
B.14.2.6 tms::ConfigId . . . 230
B.14.2.7 tms::ParameterValueSequence . . . 231
B.14.2.8 tms::ParameterValue . . . 231
B.14.2.9 tms::String1-32 . . . 231
B.14.2.10 tms::DeviceConfigRequest . . . 232
B.14.2.11 tms::RequestSequence . . . 232
B.14.2.12 tms::PowerPortNumber . . . 232
B.14.2.13 tms::ReplyStatus . . . 232
B.14.2.14 tms::ReplyCode . . . 232
B.14.2.15 tms::String32 . . . 232
B.15 Grounding Circuits . . . 232
B.15.1 Data Flows . . . 233
B.15.2 Data Types . . . 234
B.15.2.1 tms::GroundingCircuitState . . . 235
B.15.2.2 tms::GroundingCircuitRequest . . . 236
B.15.2.3 tms::Reply . . . 236
B.15.2.4 tms::Identity . . . 236
B.15.2.5 tms::GroundingCircuitStateSequence . . . 237
B.15.2.6 tms::GroundingCircuitStatus . . . 237
B.15.2.7 tms::GroundingCircuitNumber . . . 237
B.15.2.8 tms::PowerSwitchStatus . . . 238
B.15.2.9 tms::CircuitContinuity . . . 239
B.15.2.10 tms::PowerSwitchReason . . . 239
B.15.2.11 tms::PowerSwitchTransitionCode . . . 240
B.15.2.12 tms::GroundFaultDetection . . . 241
B.15.2.13 tms::DeviceConfigRequest . . . 241
B.15.2.14 tms::ConfigId . . . 241
B.15.2.15 tms::RequestSequence . . . 242
B.15.2.16 tms::DesiredCircuitContinuity . . . 242
B.15.2.17 tms::PowerPortNumber . . . 242
B.15.2.18 tms::ReplyStatus . . . 242
B.15.2.19 tms::ReplyCode . . . 242
B.15.2.20 tms::String32 . . . 242
B.16 Source and Storage Start/Stop . . . 243
B.16.1 Data Flows . . . 244
B.16.2 Data Types . . . 246
B.16.2.1 tms::EnergyStartStopState . . . 246
B.16.2.2 tms::EnergyStartStopRequest . . . 247
B.16.2.3 tms::Reply . . . 248
B.16.2.4 tms::Identity . . . 248
B.16.2.5 tms::EnergyStartStopLevel . . . 248
B.16.2.6 tms::PowerSwitchReason . . . 248

CONTENTS

PARAGRAPH PAGE
B.16.2.7 tms::PowerSwitchTransitionCode . . . 248
B.16.2.8 tms::DeviceConfigRequest . . . 248
B.16.2.9 tms::ConfigId . . . 248
B.16.2.10 tms::RequestSequence . . . 248
B.16.2.11 tms::PowerSwitchCloseConditions . . . 249
B.16.2.12 tms::PowerPortNumber . . . 249
B.16.2.13 tms::ReplyStatus . . . 249
B.16.2.14 tms::ReplyCode . . . 249
B.16.2.15 tms::String32 . . . 249
B.17 Power Switches and Connectors . . . 250
B.17.1 Data Flows . . . 250
B.17.2 Data Types . . . 251
B.17.2.1 tms::PowerPortState . . . 252
B.17.2.2 tms::PowerSwitchRequest . . . 252
B.17.2.3 tms::Reply . . . 253
B.17.2.4 tms::Identity . . . 253
B.17.2.5 tms::PowerPortStatusSequence . . . 253
B.17.2.6 tms::PowerPortStatus . . . 253
B.17.2.7 tms::PowerPortNumber . . . 254
B.17.2.8 tms::CableSenseStatus . . . 254
B.17.2.9 tms::PowerPortId . . . 254
B.17.2.10 tms::PowerSwitchStatus . . . 254
B.17.2.11 tms::CircuitContinuity . . . 255
B.17.2.12 tms::PowerSwitchReason . . . 255
B.17.2.13 tms::PowerSwitchTransitionCode . . . 255
B.17.2.14 tms::PowerPortConfigRequest . . . 255
B.17.2.15 tms::ConfigId . . . 255
B.17.2.16 tms::RequestSequence . . . 255
B.17.2.17 tms::DesiredCircuitContinuity . . . 255
B.17.2.18 tms::PowerSwitchCloseConditions . . . 256
B.17.2.19 tms::ReplyStatus . . . 256
B.17.2.20 tms::ReplyCode . . . 256
B.17.2.21 tms::String32 . . . 256
B.18 AC Measurements . . . 256
B.18.1 Data Flows . . . 256
B.18.2 Data Types . . . 258
B.18.2.1 tms::ac::MeasurementUpdate . . . 258
B.18.2.2 tms::Identity . . . 259
B.18.2.3 tms::ClockMonotonic . . . 259
B.18.2.4 tms::ac::PowerPortMeasurementSequence . . . 259
B.18.2.5 tms::ac::PowerPortMeasurement . . . 260
B.18.2.6 tms::PowerPortNumber . . . 262

CONTENTS

PARAGRAPH PAGE
B.18.2.7 tms::ac::PowerLineMeasurementSequence 262
B.18.2.8 tms::ac::PowerLineMeasurement 263
B.18.2.9 tms::ac::PhaseAngleSequence 264
B.19 DC Measurements 264
B.19.1 Data Flows 265
B.19.2 Data Types 266
B.19.2.1 tms::dc::MeasurementUpdate 267
B.19.2.2 tms::Identity 267
B.19.2.3 tms::ClockMonotonic 267
B.19.2.4 tms::dc::PowerPortMeasurementSequence 268
B.19.2.5 tms::dc::PowerPortMeasurement 268
B.19.2.6 tms::PowerPortNumber 269
B.19.2.7 tms::dc::PowerLineMeasurementSequence 270
B.19.2.8 tms::dc::PowerLineMeasurement 270
B.20 Power Topology 271
B.20.1 Data Flows 271
B.20.2 Data Types 273
B.20.2.1 tms::PowerConnectionState 274
B.20.2.2 tms::Identity 274
B.20.2.3 tms::PowerConnectionSequence 274
B.20.2.4 tms::PowerConnection 275
B.20.2.5 tms::PowerConnectionId 276
B.20.2.6 tms::PowerPortId 276
B.20.2.7 tms::PowerPortNumber 276
B.20.2.8 tms::PowerConnectionDetectionType 276
B.21 AC Load Sharing 278
B.21.1 Data Flows 280
B.21.2 Data Types 281
B.21.2.1 tms::ac::LoadSharingState 282
B.21.2.2 tms::ac::LoadSharingRequest 283
B.21.2.3 tms::Reply 283
B.21.2.4 tms::Identity 283
B.21.2.5 tms::ConfigId 283
B.21.2.6 tms::ac::LoadSharingParameterSequence 284
B.21.2.7 tms::ac::LoadSharingParameters 285
B.21.2.8 tms::PowerPortNumber 286
B.21.2.9 tms::ControlCurve 287
B.21.2.10 tms::Curve2D 288
B.21.2.11 tms::Point2DSequence 288
B.21.2.12 tms::Point2D 288
B.21.2.13 tms::PowerPortConfigRequest 288
B.21.2.14 tms::RequestSequence 288

CONTENTS

PARAGRAPH PAGE
B.21.2.15 tms::ReplyStatus . . . 288
B.21.2.16 tms::ReplyCode . . . 288
B.21.2.17 tms::String32 . . . 288
B.22 DC Load Sharing . . . 288
B.22.1 Data Flows . . . 291
B.22.2 Data Types . . . 292
B.22.2.1 tms::dc::LoadSharingState . . . 293
B.22.2.2 tms::dc::LoadSharingRequest . . . 293
B.22.2.3 tms::Reply . . . 294
B.22.2.4 tms::Identity . . . 294
B.22.2.5 tms::ConfigId . . . 294
B.22.2.6 tms::dc::LoadSharingParameterSequence . . . 294
B.22.2.7 tms::dc::LoadSharingParameters . . . 295
B.22.2.8 tms::PowerPortNumber . . . 295
B.22.2.9 tms::ControlCurve . . . 296
B.22.2.10 tms::Curve2D . . . 296
B.22.2.11 tms::Point2DSequence . . . 296
B.22.2.12 tms::Point2D . . . 296
B.22.2.13 tms::PowerPortConfigRequest . . . 296
B.22.2.14 tms::RequestSequence . . . 296
B.22.2.15 tms::ReplyStatus . . . 296
B.22.2.16 tms::ReplyCode . . . 296
B.22.2.17 tms::String32 . . . 296
B.23 Clock Status . . . 296
B.23.1 Data Flows . . . 297
B.23.2 Data Types . . . 298
B.23.2.1 tms::ClockState . . . 298
B.23.2.2 tms::Identity . . . 299
B.23.2.3 tms::ClockMonotonic . . . 299
B.23.2.4 tms::ClockMonotonicPersistenceType . . . 299
B.23.2.5 tms::ClockRealtime . . . 300
B.23.2.6 tms::ClockSource . . . 300
B.24 Authorization To Energize . . . 302
B.24.1 Data Flows . . . 305
B.24.2 Data Types . . . 306
B.24.2.1 tms::AuthorizationToEnergizeRequest . . . 307
B.24.2.2 tms::AuthorizationToEnergizeReply . . . 308
B.24.2.3 tms::AuthorizationToEnergizeResult . . . 309
B.24.2.4 tms::GridRequest . . . 310
B.24.2.5 tms::Identity . . . 310
B.24.2.6 tms::RequestSequence . . . 311
B.24.2.7 tms::PowerPortNumberSequence . . . 311

CONTENTS

PARAGRAPH PAGE
B.24.2.8 tms::PowerPortNumber . . . 311
B.24.2.9 tms::ClockRealtime . . . 311
B.24.2.10 tms::ClockMonotonic . . . 311
B.100 Data Type Index . . . 313
B.100.1 Data Types in All Namespaces . . . 313
B.100.2 Data Types in Namespace tms . . . 315
B.100.3 Data Types in Namespace tms::ac . . . 318
B.100.4 Data Types in Namespace tms::dc . . . 318
B.100.5 Defined Constants . . . 319

C Cybersecurity Requirements 320

D OMG DDS Requirements 321
D.1 Network Interoperability . . . 321
D.2 Domains . . . 322
D.3 Topics . . . 323
D.4 IDL . . . 323
D.4.1 Data Types . . . 323
D.4.2 Annotations . . . 324
D.4.2.1 Extensibility . . . 324
D.4.2.2 Topic Usage . . . 324
D.4.2.3 Key Values . . . 324
D.4.2.4 Bounded Values . . . 325
D.4.2.5 Units of Measure . . . 325
D.5 Quality of Service . . . 326
D.5.1 PublishLast Profile . . . 327
D.5.2 Rare Profile . . . 329
D.5.3 Slow Profile . . . 331
D.5.4 Medium Profile . . . 333
D.5.5 Continuous Profile . . . 335
D.5.6 Command Profile . . . 337
D.5.7 Response Profile . . . 339
D.5.8 Reply Profile . . . 341
D.6 DDS Security . . . 342
D.6.1 Data Model Integration . . . 342
D.6.2 External Interfaces . . . 343
D.6.3 DDS Security Configuration . . . 343
D.6.4 Microgrid Governance . . . 344
D.6.5 Device Permissions . . . 345

CONTENTS

PARAGRAPH PAGE

E Machine Readable Files **347**
E.1 IDL 347
E.2 Quality of Service XML 409
E.3 Topics Table 415
E.4 Data Flows Table 418

Change History **419**
Version 2.0.3 419
Version 2.0.2 419
Version 2.0.1 420

CONTENTS

FIGURE PAGE

A.1 TMS Device Roles . . . 52
A.2 Example Power Balance Automation . . . 55
A.3 Overview of communications for consistent operation. . . . 58
A.4 Microgrid controller selection state diagram. . . . 59
A.5 Data Flow Notation . . . 61
A.6 Data samples in the business logic become messages in the network logic. . . . 62
A.7 Example State Data Flow Pattern . . . 63
A.8 Example Request/Reply Data Flow Pattern . . . 63
A.9 Request/Reply with State Data Flow Pattern Example . . . 64
A.10 Illustration of Sample Update and Message Timing . . . 66
A.11 Example timing for a periodic 1 s topic. . . . 68
A.12 Example publisher send windows. . . . 69
A.13 Example subscriber receive windows. . . . 69
A.14 Order of Delivery Example . . . 74
A.15 Power measurement sign conventions. . . . 80
A.16 Example measurements of a 0.8 power factor inductive load. . . . 81

B.1 Conceptual logic of the Microgrid Controller device role. . . . 97
B.2 Conceptual logic of the Microgrid Dashboard device role. . . . 99
B.3 Conceptual logic of the Monitor device role. . . . 101
B.4 Conceptual logic of all power device roles. . . . 102
B.5 Components of a power port. . . . 103
B.6 Schematic diagram of the Source device role. . . . 104
B.7 Schematic diagram of the Storage device role. . . . 107
B.8 Schematic diagram of the Load device role. . . . 109
B.9 Schematic diagram of the Distribution device role. . . . 111
B.10 Schematic diagram of the Conversion device role. . . . 113
B.11 Request/Response Pattern data flow . . . 117
B.12 Heartbeat data flow . . . 125
B.13 Device Announcement data flow . . . 127
B.14 Example `Curve2D` with three points. . . . 166
B.15 Nickname data flow . . . 179
B.16 Device Icon data flow . . . 183
B.17 Control Hardware data flow . . . 185
B.18 Power Hardware data flow . . . 189
B.19 Energy Storage data flow . . . 198
B.20 Operator Input data flow . . . 202
B.21 Diagnostics Reporting data flow . . . 214
B.22 Microgrid Controller Selection data flow . . . 224
B.23 Device Parameters data flow . . . 226
B.24 Device Parameters data flow . . . 226

CONTENTS

FIGURE PAGE

B.25 Grounding Circuits data flow . . . 233
B.26 Source and Storage Start/Stop data flow . . . 244
B.27 Power Switches and Connectors data flow . . . 250
B.28 AC Measurements data flow . . . 256
B.29 AC Measurements data flow . . . 256
B.30 Phase Angles for a 208/120V Wye System . . . 261
B.31 DC Measurements data flow . . . 265
B.32 DC Measurements data flow . . . 265
B.33 Power Topology data flow . . . 271
B.34 Power Topology data flow . . . 271
B.35 Power Topology data flow . . . 272
B.36 Example AC Load Sharing Control References . . . 278
B.37 Example AC Load Sharing Operating Bounds . . . 278
B.38 Example AC Load Sharing Types . . . 279
B.39 Example Multi-Segment AC Load Sharing . . . 279
B.40 Example AC Load Sharing Region . . . 280
B.41 AC Load Sharing data flow . . . 280
B.42 Interpreting ControlCurve minimum, nominal, and maximum. . . . 287
B.43 Example DC Load Sharing Control References . . . 289
B.44 Example DC Load Sharing Operating Bounds . . . 289
B.45 Example DC Load Sharing Types . . . 290
B.46 Example Multi-Segment DC Load Sharing . . . 290
B.47 Example DC Load Sharing Region . . . 290
B.48 DC Load Sharing data flow . . . 291
B.49 Clock Status data flow . . . 297
B.50 Authorization To Energize Timing . . . 302
B.51 Authorization To Energize Protocol . . . 303
B.52 Energizing Action State Diagram . . . 304
B.53 Safety Indicator State Diagram . . . 304
B.54 Authorization To Energize data flow . . . 305

D.1 Checklist of OMG DDS interoperability settings . . . 322

E.1 Example Topic Definition Tables . . . 416
E.2 Example Data Flow . . . 418

CONTENTS

TABLE PAGE

A.1 Times for microgrid controller selection. . . . 59
A.2 Overview of all data types used as key values. . . . 64
A.3 Relationship between Timing Parameters. . . . 66
A.4 Specifications for each rate class. . . . 70
A.5 High-level definitions of QoS Profiles. . . . 72
A.6 Topic Suffixes 74
A.7 Namespaces defined by TMS. . . . 78
A.8 Units of Measure 79
A.9 Overview of Primitive Data Types and their General Usage 82
A.10 UCS Portable Character Set 83
A.11 Required Integer Types 84
A.12 Naming convention for pre-defined length constants. . . . 87
A.13 Default Values for all Data Types 88
A.14 Appendable Extensibility for all Data Types 89
A.15 Example quantities for a small microgrid. . . . 90

B.1 Overview of Topic Participants. P=publisher, S=subscriber. . . . 94
B.2 Overview of Topic Usage. . . . 95
B.3 Data Model Sections for Source Device Role Components 106
B.4 Data Model Sections for Storage Device Role Components 108
B.5 Data Model Sections for Load Device Role Components 110
B.6 Data Model Sections for Distribution Device Role Components 112
B.7 Data Model Sections for Conversion Device Role Components 115
B.8 Description of the Request/Response Pattern topics. . . . 117
B.9 Participants of the Request/Response Pattern topics. . . . 117
B.10 Timing of the Request/Response Pattern topics. . . . 118
B.11 Data types of the Request/Response Pattern topics. . . . 118
B.17 Description of the Heartbeat topics. . . . 125
B.18 Participants of the Heartbeat topics. . . . 125
B.19 Timing of the Heartbeat topics. . . . 125
B.20 Data types of the Heartbeat topics. . . . 126
B.22 Description of the Device Announcement topics. . . . 127
B.23 Participants of the Device Announcement topics. . . . 127
B.24 Timing of the Device Announcement topics. . . . 128
B.25 Data types of the Device Announcement topics. . . . 128
B.41 ParameterMetadata values for primitive data types. . . . 146
B.49 Usual relationship between `PowerPortInfo` directionality and amperage limits. 153
B.81 Description of the Nickname topics. . . . 179
B.82 Participants of the Nickname topics. . . . 179
B.83 Timing of the Nickname topics. . . . 180
B.84 Data types of the Nickname topics. . . . 180

CONTENTS

TABLE PAGE
B.88 Description of the Device Icon topics. . . . 183
B.89 Participants of the Device Icon topics. . . . 183
B.90 Timing of the Device Icon topics. . . . 184
B.91 Data types of the Device Icon topics. . . . 184
B.93 Description of the Control Hardware topics. . . . 186
B.94 Participants of the Control Hardware topics. . . . 186
B.95 Timing of the Control Hardware topics. . . . 186
B.96 Data types of the Control Hardware topics. . . . 186
B.99 Description of the Power Hardware topics. . . . 189
B.100 Participants of the Power Hardware topics. . . . 189
B.101 Timing of the Power Hardware topics. . . . 189
B.102 Data types of the Power Hardware topics. . . . 190
B.113 Description of the Energy Storage topics. . . . 198
B.114 Participants of the Energy Storage topics. . . . 199
B.115 Timing of the Energy Storage topics. . . . 199
B.116 Data types of the Energy Storage topics. . . . 199
B.118 Description of the Operator Input topics. . . . 202
B.119 Participants of the Operator Input topics. . . . 203
B.120 Timing of the Operator Input topics. . . . 203
B.121 Data types of the Operator Input topics. . . . 204
B.135 Description of the Diagnostics Reporting topics. . . . 214
B.136 Participants of the Diagnostics Reporting topics. . . . 215
B.137 Timing of the Diagnostics Reporting topics. . . . 215
B.138 Data types of the Diagnostics Reporting topics. . . . 215
B.142 Suspect Parameter Number value ranges 221
B.145 Description of the Microgrid Controller Selection topics. . . . 224
B.146 Participants of the Microgrid Controller Selection topics. . . . 224
B.147 Timing of the Microgrid Controller Selection topics. . . . 224
B.148 Data types of the Microgrid Controller Selection topics. . . . 225
B.150 Device parameters defined by TMS. . . . 226
B.151 Description of the Device Parameters topics. . . . 227
B.152 Participants of the Device Parameters topics. . . . 227
B.153 Timing of the Device Parameters topics. . . . 228
B.154 Data types of the Device Parameters topics. . . . 228
B.160 Description of the Grounding Circuits topics. . . . 233
B.161 Participants of the Grounding Circuits topics. . . . 233
B.162 Timing of the Grounding Circuits topics. . . . 234
B.163 Data types of the Grounding Circuits topics. . . . 234
B.173 Definition of the `StartStopLevel`s. . . . 243
B.174 Example operating levels for a diesel genset SRC. . . . 244
B.175 Example operating levels for a battery STOR. . . . 244
B.176 Description of the Source and Storage Start/Stop topics. . . . 245

CONTENTS

TABLE PAGE
B.177 Participants of the Source and Storage Start/Stop topics. 245
B.178 Timing of the Source and Storage Start/Stop topics. 245
B.179 Data types of the Source and Storage Start/Stop topics. 246
B.183 Description of the Power Switches and Connectors topics. 250
B.184 Participants of the Power Switches and Connectors topics. 250
B.185 Timing of the Power Switches and Connectors topics. 251
B.186 Data types of the Power Switches and Connectors topics. 251
B.193 Description of the AC Measurements topics. 257
B.194 Participants of the AC Measurements topics. 257
B.195 Timing of the AC Measurements topics. 257
B.196 Data types of the AC Measurements topics. 258
B.199 AcMeasurementUpdate line order for WIRING_AC_SINGLE. 261
B.200 AcMeasurementUpdate line order for WIRING_AC_SPLIT. 261
B.201 AcMeasurementUpdate line order for WIRING_AC_3DELTA. 261
B.202 AcMeasurementUpdate line order for WIRING_AC_3WYE. 261
B.203 AcSummaryMeasurementUpdate line order. 262
B.204 PowerLineMeasurement values for AcSummaryMeasurementUpdate. 262
B.206 Description of the DC Measurements topics. 265
B.207 Participants of the DC Measurements topics. 265
B.208 Timing of the DC Measurements topics. 266
B.209 Data types of the DC Measurements topics. 266
B.212 DcMeasurementUpdate line order for WIRING_DC. 269
B.213 DcMeasurementUpdate line order for WIRING_DC_3WIRE. 269
B.214 DcSummaryMeasurementUpdate line order. 269
B.215 PowerLineMeasurement values for DcSummaryMeasurementUpdate. 269
B.217 Description of the Power Topology topics. 272
B.218 Participants of the Power Topology topics. 272
B.219 Timing of the Power Topology topics. 273
B.220 Data types of the Power Topology topics. 273
B.225 Description of the AC Load Sharing topics. 280
B.226 Participants of the AC Load Sharing topics. 281
B.227 Timing of the AC Load Sharing topics. 281
B.228 Data types of the AC Load Sharing topics. 282
B.233 Description of the DC Load Sharing topics. 291
B.234 Participants of the DC Load Sharing topics. 291
B.235 Timing of the DC Load Sharing topics. 292
B.236 Data types of the DC Load Sharing topics. 292
B.240 Description of the Clock Status topics. 297
B.241 Participants of the Clock Status topics. 297
B.242 Timing of the Clock Status topics. 297
B.243 Data types of the Clock Status topics. 298
B.248 Description of the Authorization To Energize topics. 305

CONTENTS

TABLE PAGE

B.249 Participants of the Authorization To Energize topics. . . . 305
B.250 Timing of the Authorization To Energize topics. . . . 306
B.251 Data types of the Authorization To Energize topics. . . . 306

D.1 Referenced OMG DDS Standards . . . 321
D.2 TypeConsistencyEnforcementQosPolicy Settings . . . 321
D.3 IDL Type Mapping . . . 323
D.4 IDL Annodations for Extensibility . . . 324
D.5 IDL Annotations for Topic Usage . . . 324
D.6 IDL Annotations for Bounded Values . . . 325
D.7 OMG DDS QoS Policies for the PublishLast Profile . . . 327
D.8 OMG DDS QoS Policies for the Rare Profile . . . 329
D.9 OMG DDS QoS Policies for the Slow Profile . . . 331
D.10 OMG DDS QoS Policies for the Medium Profile . . . 333
D.11 OMG DDS QoS Policies for the Continuous Profile . . . 335
D.12 OMG DDS QoS Policies for the Command Profile . . . 337
D.13 OMG DDS QoS Policies for the Response Profile . . . 339
D.14 OMG DDS QoS Policies for the Reply Profile . . . 341
D.15 Microgrid Governance Settings . . . 344
D.16 Device Permissions Settings . . . 346

E.1 Description of the Device Announcement topics. . . . 416
E.2 Participants of the Device Announcement topics. . . . 416
E.3 Timing of the Device Announcement topics. . . . 416
E.4 Data types of the Device Announcement topics. . . . 416

A. General Data Model Requirements

A.1 Introduction

The TMS defines an *architecture* for building microgrid *systems*. Each microgrid system defines one or more *devices* that connect together. Each device implements a single *role* that provides a modular unit of functionality. Each role has defined interfaces that support both standalone and coordinated behaviors. Multiple devices, implementing the same or different roles, may be packaged together into a single *platform*. This architecture is designed to enable interoperability between devices and platforms from all systems, subject to power constraints.

Individual devices perform essential TMS functions using only internal inputs and outputs. System-level TMS performance may be enhanced by sending coordination messages between devices. The TMS *data model* defines the language for these messages.

The data model is defined in terms of *data flows* and *data types*. Data flows define the motion of messages while data types define the contents of messages.

The TMS data model is documented in several chapters.

- Chapter A defines requirements for the general-purpose communications infrastructure.
- Chapter B defines requirements for domain-specific messages.
- Chapter C defines requirements for cybersecurity.
- Chapter D defines how OMG DDS is used to meet these requirements. TMS device software may be roughly categorized into business logic and network logic. The OMG DDS middleware encapsulates most of the network logic, thus allowing software re-use and simplifying the business logic.
- Chapter E provides machine-readable definitions of the data model.

A.2 Scope and Purpose

This document defines the TMS data model, including all messages sent between TMS devices. These messages are used to communicate all system health, status, command, and control information. They are also recommended for communications with external networks.

The data model provides a common language used by all TMS devices. It is platform-independent to allow flexibility in device design and implementation. It contains detailed data flow and data type definitions to guarantee interoperability between devices, regardless of make or mode.

A.3 Implementation Considerations

The data model allows for design and implementation flexibility while still providing interoperability. This section identifies high-level objectives for any TMS design and implementation plan. These points should be carefully considered in all decisions.

Security The design should ensure safe and trustworthy operations of the TMS by promoting the availability, integrity, and positive control of microgrid resources. It should remain resilient in the face of device compromise and hamper the spread of malware infection.

Many security properties must be integrated to provide seamless operation, interoperability, and a high level of resilience. The root of trust must include cryptographic identities for hardware, software, and users. Data flows must be enforced with authentication, authorization, and integrity protection. Logging must provide a high-integrity, trustworthy forensic record of events from production through operation.

Performance The design should support the desired functionality in an easily-proven manner. It should not unduly burden either operator or equipment. It should be robust against potential faults and support graceful degradation. It should support reliability, availability, and maintainability.

Simplicity The design should be simple to understand and implement. Usage patterns should be predictable and consistent. Each part of the data model trades increased functionality for increased overall complexity. Simpler solutions improve this trade-off.

Interoperability The design should be clear and unambiguous. Standard interaction patterns should be well defined. Incompatible dialects should be avoided by reducing options. In the ideal case, any two devices, developed in isolation and only linked by this standard, will work together the first time they are connected.

Extensibility The design should be flexible and support a range of possible solutions. Room should be left for innovation and product differentiation. The design should provide mechanism, without enforcing too much policy. Options may be left in places where there are compelling alternatives. Well-chosen constraints increase extensibility by providing system-level guarantees.

Verification and Validation Validation artifacts should demonstrate that all design thresholds and objectives were met. Verification tests should demonstrate conformance of devices to the TMS data model. The design should support clear and rigorous verification and validation. Both verification and validation results should provide an accurate prediction of operational performance in the field and interoperability with other possible device variations.

A.4 General Requirements

The general requirements discusses the TMS data model architecture and the basic structure of communication between TMS devices. The data flows and data types defined in this standard identify the basic communication requirements for TMS.

This chapter defines how the data model is constructed. Chapter B defines the data model at a conceptual level. Chapter D provides the low-level details of how the data model is implemented.

A.5 Device Identity

Every TMS device has a unique identity which is bound to a single, specific device role that has a defined set of requirements and permissions. The identity value distinguishes between devices, and the bound information establishes how each identity can interact with other devices. For example, three generators may have the same type and role but they would have separate identities, and none of these identities are valid for sending microgrid controller requests.

Identities are used to associate control and monitoring information with a specific device. Identities must be unique for every device in the microgrid. Cryptography should be used to protect the identities from impersonation and their bound information from modification.

The device identity is represented by the `tms::Identity` data type defined in Section B.3.

A.6 Device Roles

Device roles are defined building blocks within the TMS that provide modularized functionality. Each role is assigned based upon the device's capabilities as defined in the TMS. Each device role is pre-defined in both functionality and power/communication interfaces to provide modular flexibility.

The device roles were designed to meet many objectives including the following.

1. Self-sufficient. Every device role provides basic functionality by itself. Communications enhance functionality but are not required for baseline operation.
2. Simple. Understandable to an operator or microgrid controller. Each role provides distinct functionality. Complexity is in the assembly, not the individual devices.
3. Flexible. Devices can be assembled in different ways to meet changing mission needs.
4. Interoperable. Well-defined interfaces with clear interactions between roles. Products from different manufacturers can be connected without integration difficulties.
5. Extensible. There is room for innovation, competition, and growth.
6. Universal. Every type of power hardware fits in a device role.

There are two types of device roles in TMS: Control Devices and Power Devices. Figure A.1 shows the relationship between these roles.

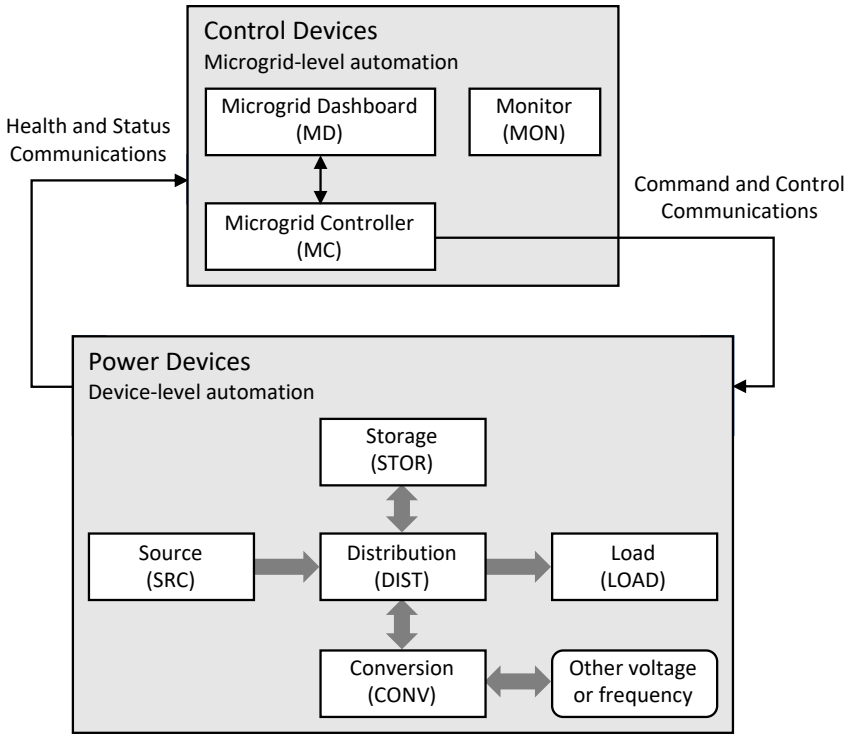


Figure A.1: TMS Device Roles

A.6.1 Control Devices

Control device roles embody functionality that is not specific to a particular type of power hardware. They provide mostly microgrid-level functionality and rely on communications with the power devices.

The Control Device roles include:

- **Microgrid Controller (MC)**. Coordinates automation across the microgrid. Monitors power devices and requests actions or other configuration settings as needed.
- **Microgrid Dashboard (MD)**. Provides a microgrid-level graphical user interface (GUI) to monitor health and status. May allow the operator to send requests to the MC for execution.
- **Monitor (MON)** Provides passive monitoring of all communications. Does not send any information. May be used to record network traffic, create logs, and monitor test capability.

More details on the control device roles, including diagrams of their major logical components, are provided in Section B.1.

A.6.2 Power Devices

Power device roles are specific to particular types of power hardware. They provide mostly device-level functionality and include everything that uses, generates, or stores power.

The Power Device roles include:

- **Source (SRC)** devices provide power to the microgrid and include diesel gensets, solar panels, and wind turbines. Note that while an energy storage unit (ESU) can act as a source, TMS considers ESUs exclusively as Storage device roles.
- **Storage (STOR)** devices charge to store power and discharge to return it. Storage devices include batteries and flywheels.
- **Load (LOAD)** devices consume power from the microgrid. The purpose of the microgrid is to power load devices. While most loads will interoperate without implementing TMS, those that do will have access to features such as health and status monitoring, demand response, and reduced spinning reserve.
- **Distribution (DIST)** devices connect other power devices together and commonly provide metering and switching capabilities.
- **Conversion (CONV)** devices modify the input power to output a different frequency or voltage for other devices. Conversion devices include inverters, rectifiers, and transformers.

The SRC, STOR, and LOAD device roles have a single internal power bus and a single external power connector. The DIST and CONV device roles have a single internal power bus and two or more external power connectors. Thus SRC, STOR, and LOAD are called *single-port* devices, and DIST and CONV are called *multi-port* devices.

More details on the power device roles, including schematics of their major hardware components, are provided in Section B.1.

A.6.3 Platforms and Multi-Role Products

While every device in the TMS data model has a unique identity and fills a single, specific device role to simplify communications and prevent ambiguity about role interactions, there are cases where grouping multiple TMS devices into a single physical or virtual container is desirable. The reasons for such groupings are typically cost, size, shipping, and presentation.

Examples of platforms and multi-role products include:

- Multiple devices that are mounted together on a larger *platform*, such as:
  - A vehicle with separate SRC, DIST, and MC devices mounted on the chassis.
  - A building with LOAD, DIST, and backup SRC equipment.
- Products that package multiple device roles into a single *multi-role product*, such as:
  - A generator with a built-in microgrid controller has both SRC and MC device roles.
  - A control computer incorporating MC and MD device roles.
  - A simulation computer with separate identities for each simulated device role.

- Products that have one device role but may be split into multiple identities of the device role, such as:
  - A large DIST device with the power ports split into multiple DIST device role identities.

Each device on a platform shall have a separate identity and role. Multi-role products shall have separate identities and device roles for each type of functionality. Multi-role products shall communicate as if they were multiple physically-separate devices. Platforms and multi-role products are represented in TMS as a collection of devices sharing a common `tms::ProductInfo.platformId`. The `platformId` is considered static data and shall not be changed. Platform devices generally move together. In other words, they attach and disconnect from the microgrid as a group. Products with a built-in MC shall allow the internal MC to be disabled so the other devices can be used without it.

When operating within a platform, only a single device may act as the platform representative responsible for publishing *platform*-specific information such as the platform `IdentityNicknameState` or `DeviceIcon`, if these features are supported. Each platform shall have at most one platform representative, identified as the only device publishing `tms::ProductInfo.platformRepresentative` as true.

It is recommended that each:

- Device role has its own data connection and runs on a separate processor so each role can be physically isolated.
- TMS power device has a minimum of two data connections, one for networking with other TMS devices and one for maintenance access.
- TMS DIST device has a network switch with one connector for each power port so every TMS device that connects to power can also connect to data.

If multiple devices are packaged into a single platform or multi-role product, they may share hardware or software components. If such sharing does occur, then all messages shall be sent separately for each contained device identity and role. Messages may be received singly and re-shared using internal mechanisms.

Change Note: Improved description of platforms and multi-role products. This version introduces the concept of a platform representative.

A.7 Microgrid Functions

This section describes some high-level functionality that is essential to microgrid operations. Multiple TMS devices and roles need to coordinate in order to perform these functions. Chapter B provides details on how these functions are implemented in TMS.

A.7.1 Power Balance

The fundamental purpose of a microgrid is to meet the power demands of the attached loads. In many situations, power devices automatically keep the frequency and voltage within rated values. The MC monitors the automatic actions of all devices and may request coordination between devices to efficiently meet microgrid power needs.

When the load changes or something else disturbs the power balance, many different actions can increase power output or decrease power consumption to restore the balance. Figure A.2 provides an example of the response times for different device actions. The following responses and times are supported but not required by TMS.

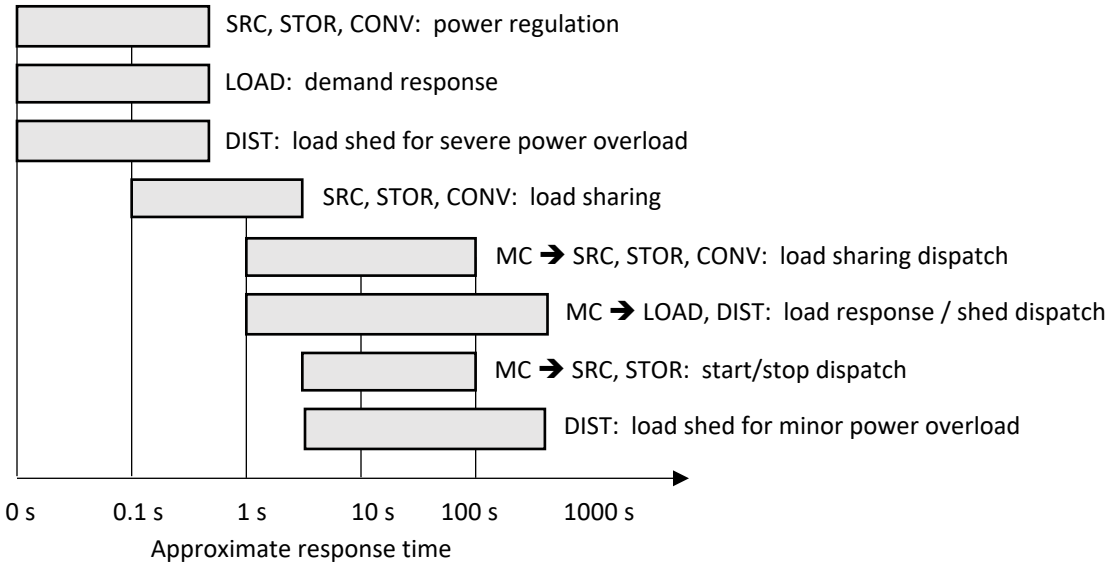


Figure A.2: Example Power Balance Automation

The fastest response can be provided by automatic actions within each power device, using internal inputs and outputs, without depending on external communications. Automatic power regulation by a SRC, STOR, or CONV is most preferred because all loads remain operational. Demand response by a LOAD is acceptable because it is specific to each device, it can reduce power without completely shutting down, and it can be indicated on the device. Load shed by a DIST is better than a potential blackout but not desirable because it completely disables one or more loads, and it may be difficult for the operator to understand where and why the load shed occurred.

After the initial transient response, automatic load sharing between SRC, STOR, and CONV devices re-balances power output and dampens any oscillations.

Any MC response is delayed by communications latency and the sampling rate of periodic measurements. The MC may request SRC, STOR, and CONV devices to adjust their power output. The MC may request LOAD and DIST devices to adjust their power consumption.

If power adjustments are not sufficient, the MC may request additional SRC or STOR devices to start or stop. DIST devices may also have automatic responses to shed loads that are causing long-term problems.

Manual actions are usually slower than any of these automatic actions, and automation can reduce the operator workload.

A.7.2 Energy Dispatch

When the microgrid is operating autonomously, the MC is responsible for selecting which energy resources provide power. This requires communications to collect status on the available devices, share the operator's priorities, and send requests.

Here are example considerations the MC may balance when choosing which energy resources to dispatch.

1. Available power. SRC, STOR, and CONV power capacity.
2. Available energy. SRC fuel level and STOR state of charge.
3. Topology. Electrical path between devices, including cable current limits and voltage drops.
4. Efficiency. Minimize resource consumption.
5. Resiliency. Tolerate configuration changes, load fluctuations, or equipment failures.
6. Noise. Do not be too loud.
7. Equipment wear. Coordinate maintenance activities. Avoid excessive wear leading to premature failure. Avoid multiple devices wearing out at the same time.

The actual energy dispatch considerations and related algorithms are not standardized by TMS. Rather, they are left open to innovation, and TMS simply provides communications that enable an MC to implement a wide range of options.

A.7.3 Operator Control

Usability is very important for microgrid operations. Manual operations need direct, positive control. Automatic operations need to be simple to understand, predict, and adjust. The operator needs the ability to easily switch between manual and automatic operations, so they can choose when to interact with the microgrid and are not interrupted by nuisances. Health and status information needs to be clear and easy to understand. Fault or warning messages need to be useful and prompt relevant actions.

The TMS supports such usability considerations by providing the MD device role with its associated communications and by allowing the operator to directly control devices through their on-device user interface. This standard does not define user interfaces or other human factors considerations.

A.7.4 Consistency

Effective microgrid operations require that all devices agree on the desired state of the microgrid, coordinate their actions, and behave consistently. The microgrid is a distributed system, built of multiple devices that are assembled to meet the operator's mission requirements.

The following factors present challenges to consistent operation:

1. Real time operation. Regulated power must be available whenever possible. The system cannot pause or halt by default.
2. Autonomous operation. The microgrid must not depend on expert configuration or continuous operator supervision.
3. Fault tolerance. A fault in one segment of the microgrid should should not prevent other, functional segments of the microgrid from operating properly.
4. Unreliable network communications. Messages may be delayed or lost. Communications networks may partition (split into separate segments) or join (merge previously separate segments).
5. Unreliable clock synchronization. An authoritative clock such as GPS may not be available, it may take too long to set up, or it may be jammed.
6. Dynamic configuration. Devices may be added, removed, or destroyed while the microgrid is operating.
7. Dynamic power flows. Cables may be connected or disconnected and switches may open or close, thus partitioning or joining power networks.
8. Multiple MDs. Different operators may connect to the microgrid at different locations.
9. Multiple MCs. Redundant MCs may be provided for backup or they may be packaged with other device roles. These MCs may be designed by different manufacturers, feature different capabilities, use different algorithms, and use different software versions.
10. Non-TMS devices. TMS devices must interoperate with non-TMS devices. Also, TMS devices may have TMS functionality disabled by intent or fault.
11. On-device controls. Power devices have internal automation and user interfaces.

The following strategy was adopted to address these challenges:

1. Operational priorities. Prioritize power availability and partition tolerance over consistency.
2. Power physics. Rely on the physics of the power system to resolve most inconsistencies between devices. In-device power measurements observe the resulting behavior.
3. Peer to peer communications. All devices discover and communicate with each other in a peer-to-peer fashion, without relying on a central broker, coordinator, or hub.

4. Internal clocks. All devices can use their own internal clock, without relying on synchronization between devices.

5. Distribute status information. Power devices broadcast their status, including power measurements, to all connected control devices. Thus all control devices will have an up-to-date view of the power devices that are on the same communications network.

6. Distribute power controls. All power safety and most power regulation is handled by inputs and outputs internal to each device. Some power regulation may benefit from peer-to-peer communications. The MC is only responsible for performing "slow" actions such as adjusting the in-device controls. The power devices are capable of operating without an MC connection, unless a large change in coordination is required.

7. Centralize coordination controls. Allow a single MC to make coordination decisions for all connected devices. Prioritize MCs so all power devices follow the same MC. This avoids complex algorithms, communications protocols, and delays when decisions are needed.

8. Separation of mechanism and policy. Standardize the communications necessary to implement microgrid functionality. Allow the operating policy to change over time or according to mission needs.

Figure A.3 illustrates the TMS data flow patterns that support consistent operation.

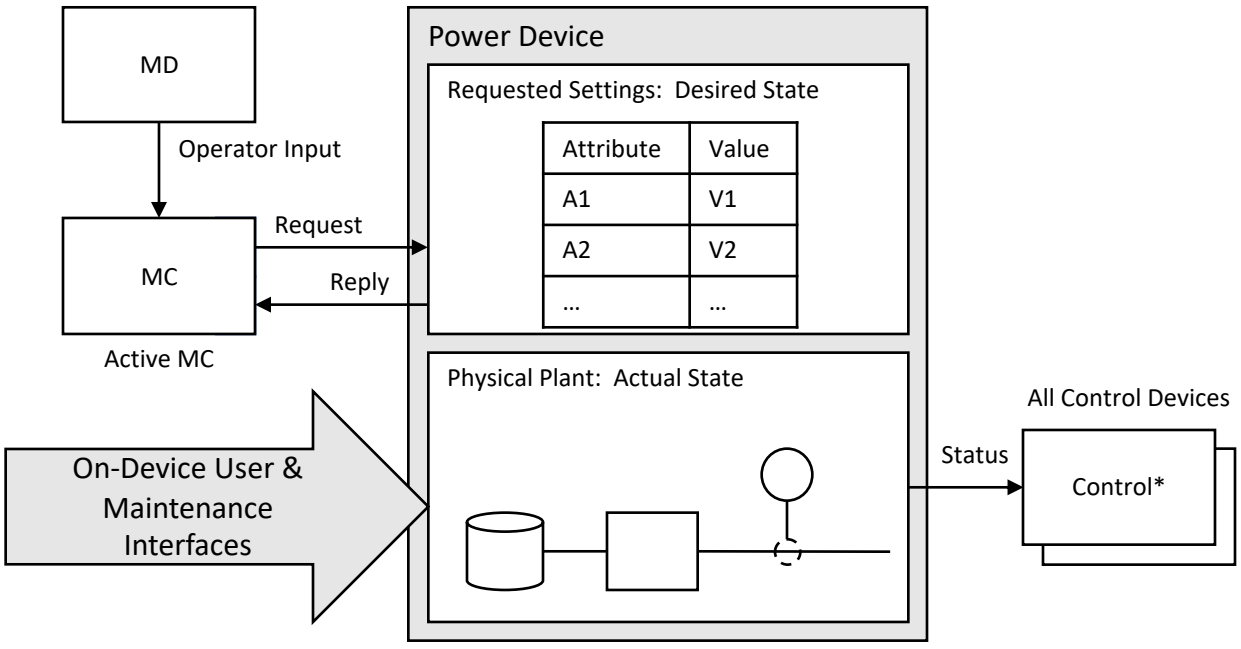


Figure A.3: Overview of communications for consistent operation.

A.7.5 Microgrid Controller Selection

Each power device independently selects which MC has active control over it, since no other device can resolve what to do if conflicting requests are received. All power devices follow the same selection algorithm, so a single MC will gain control over all connected power devices. All power devices report their active MC, so backup MCs do not send requests that will be rejected, and any inconsistent selections can be detected. See Section B.13 for details.

MC selection is triggered by the following three event-based timers. While these timers are running, the device continues accepting any new requests from the most recent active MC. Receiving a new request does not cancel these timers.

1. New MC Timer. This 3-second timer starts when a device receives a Heartbeat from a new MC. This timer continues uninterrupted if a Heartbeat is received from another new MC. When this timer expires, the active MC is selected.

2. Lost Active MC Timer. This 6-second timer starts when a device determines the active MC missed a Heartbeat deadline (3 s since last Heartbeat). This timer does not start when a non-active MC misses a Heartbeat. If a Heartbeat is received from the active MC, then this timer is canceled. When this timer expires, the active MC is cleared and a new active MC is selected.

3. No MC Timer. This 10-second timer starts when a device misses Heartbeats from all MCs. This timer may start simultaneously with the Lost Active MC Timer. If any MC Heartbeat are received, then this timer is canceled. When this timer expires, the active MC is cleared and the device processes `CONFIG_ON_COMMS_LOSS`.

The Heartbeat period and timer delays are summarized in Table A.1.

Table A.1: Times for microgrid controller selection.

Name	Period	Delay
Heartbeat	1 s	-
New MC Timer	-	3 s
Lost Active MC Timer	-	6 s
No MC Timer	-	10 s

The MC selection state machine is shown in Figure A.4.

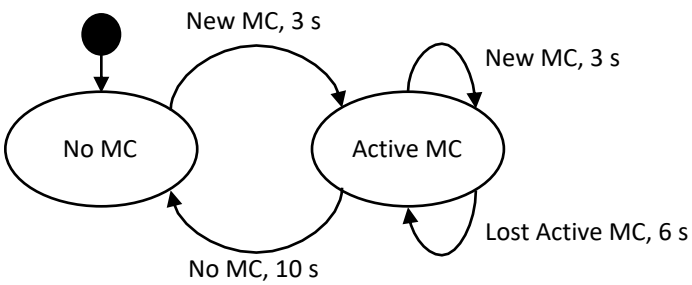


Figure A.4: Microgrid controller selection state diagram.

When MC selection is triggered, any MC that has not missed its Heartbeat deadline is considered to be available. The previous active MC is also considered to be available until the Lost Active MC Timer expires. The highest priority MC that is available is selected as the active MC. If no MC is available for selection, then the active MC is not changed.

MC priority is determined by selecting the highest priorityRanking in the MicrogridControllerInfo. If multiple MCs have equal priorityRanking, compare the MC identities character by character. The first character with a lower numeric value indicates the highest priority MC.

Change Note: Use of the priorityRanking field in MicrogridControllerInfo is a change from previous versions.

When the active MC is cleared or changes, the power device updates its ActiveMicrogridControllerState.

A.8 Data Flows

TMS devices shall communicate through pre-defined *data flows* to share all operating information such as power commands and status. Every data flow contains one or more *topics*. Each topic is defined with a name, *publishers* that send data, *subscribers* that receive data, a *data type* that specifies what values can be transferred, and *timing* that specifies when data is transferred.

This publish/subscribe (pub/sub) paradigm defines the data flows in terms of logical connections. The underlying software and hardware may use buffers, filters, and a mix of broadcast, multicast, and point-to-point connections to implement the flow. These details are specified by the OMG DDS standard as used by TMS.

Each TMS data flow defines a small group of topics that have closely related values. Multiple data flows are used together in larger patterns to achieve system-level TMS behaviors. For example, the Device Announcement flow provides information necessary for all the other data flows. For another example, power measurements that exceed a threshold may trigger the MC to request a corrective action.

This section defines the generic data flow concepts that are used for the TMS data model. The domain-specific data flows are defined in Chapter B. The OMG DDS data flow settings are defined in Chapter D.

Data flows are illustrated using the notation provided in Figure A.5, with identifiers pointing to the topic name, publisher, subscriber, and timing information. Note that in the data flow diagrams, Power* refers to any Power Device role (SRC, DIST, STOR, LOAD, CONV) and Control* refers to any Control Device role (MC, MD, MON).

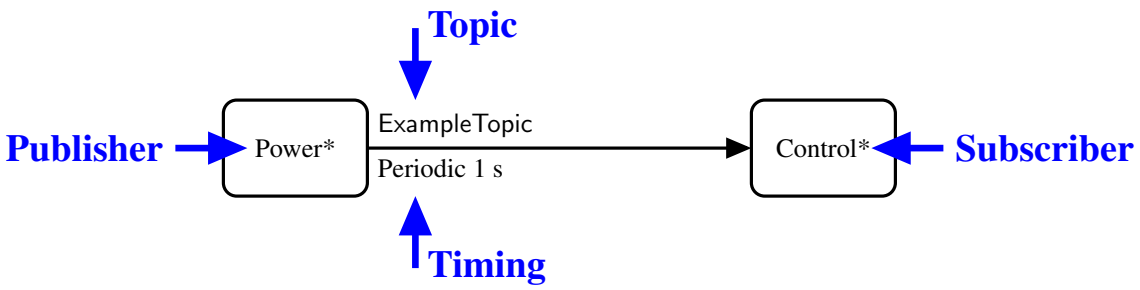


Figure A.5: Data Flow Notation

This figure shows a data flow containing a single topic. Each topic is labeled with the following features.

- **Publisher** is the Device Role(s) that sends these messages.
- **Subscriber** is the Device Role(s) that receives these messages.
- **Topic** is the name/identifier for these messages and the message flow.
- **Timing** is the condition that causes messages to be sent and the reliability of the message flow.

A.8.1 Publishers and Subscribers

Device Roles are used to specify the publisher, or source of the message, and the subscriber, or receiver of the message, for each data flow. While a device role can be both a publisher and a subscriber, typically in an individual flow it is one or the other. Conceptually, all publishers send all messages to all subscribers on a topic.

For simplicity, the device roles may be shown in the data flows throughout this document with abbreviated names - either Power* for the Power Device role or Control* for the Control Device role - to represent any device in that category. Refer to A.6 for more information on device roles.

A.8.2 Topics, Data Samples, and Messages

Topics are used to identify the data flows. Topics define a small group of TMS data flow message topics that have closely related values and match the publishers to the subscribers for each message chain. Multiple data flow topics are used together in larger patterns to achieve system-level TMS behaviors.

Each data flow topic has a single unique data type with defined trigger and reliability metrics, which are discussed in A.8.5. The same data type can be used by multiple data flow topics, providing a common connection between similar data flows.

Topic data values change over time. Each new value is called a *sample*. When a publisher updates a sample, one or more messages will be sent to transmit the new value. There is not a direct, one-to-one mapping between sample updates and messages. For example, the communications software may bundle multiple samples into a single message, split a single sample across multiple messages, or re-transmit samples when messages are lost. Any such mapping is compliant as long as the overall timing requirements are met.

The arrows in the data flow diagram indicate the flow of the message stream. The arrowhead indicates the direction that messages are being sent. The message stream for the data flows is published following specific data flow patterns that are pre-defined in the TMS. Arrows indicate messages that can be sent from both the initiating publisher and the subscriber in response to the publisher. Data flow patterns, including the subscriber response patterns, are discussed in further detail in A.8.3.

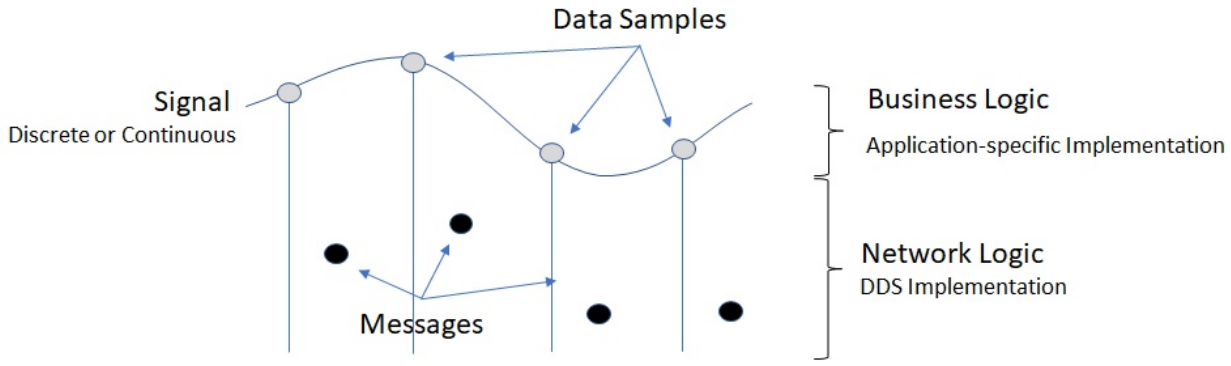


Figure A.6: Data samples in the business logic become messages in the network logic.

A.8.3 Data Flow Patterns

Most data flows use one of the following three interaction patterns: State, Request/Reply, and Request/Reply with State. Special-purpose data flows may use a custom interaction pattern. These patterns are defined below. The topic naming conventions are given in Section A.8.6.

A.8.3.1 State Pattern

The State data flow pattern supports a one-way flow of information, as shown in Figure A.7. A State topic communicates from publisher to subscriber.

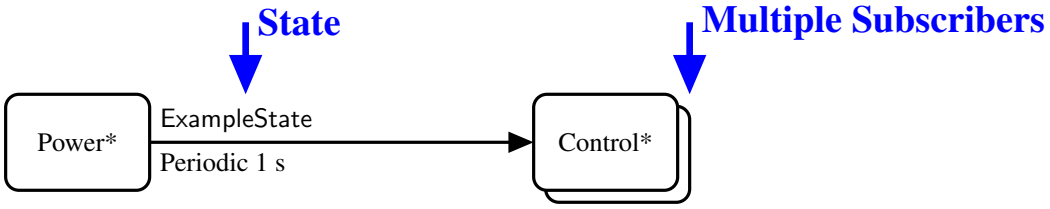


Figure A.7: Example State Data Flow Pattern

A.8.3.2 Request/Reply Pattern

The Request/Reply data flow pattern supports two-way interactions between devices, as shown in Figure A.8. A Request topic communicates to the target device, and a Reply topic returns the result. See Section B.2 for details.

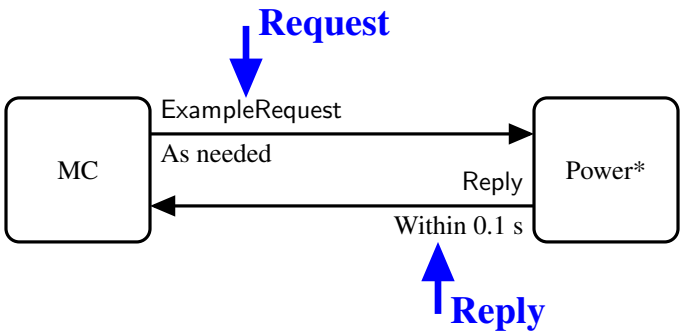


Figure A.8: Example Request/Reply Data Flow Pattern

A.8.3.3 Request/Reply with State Pattern

The Request/Reply with State data flow pattern supports multi-way interactions between devices, as shown in Figure A.9. A Request topic communicates to the target device, a Reply topic returns the result, and a State topic provides related information. The State subscribers usually include the Request publisher and other control devices. The State value may update in response to a request or a manual or automatic action such as device initiation.

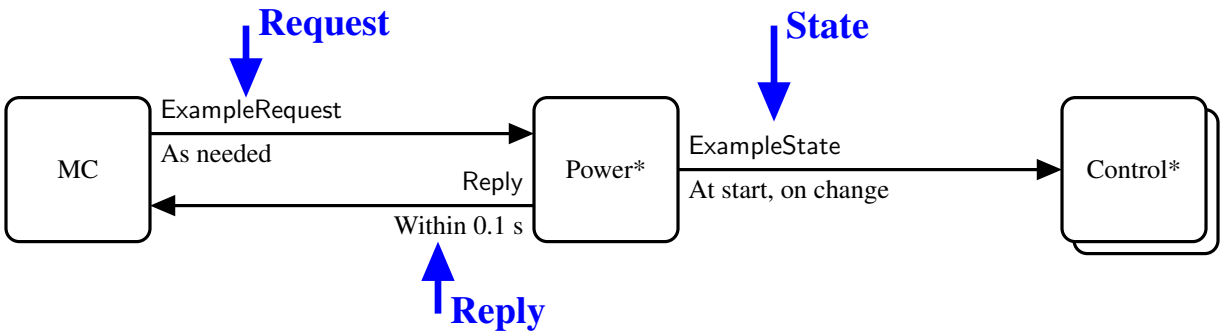


Figure A.9: Request/Reply with State Data Flow Pattern Example

A.8.3.4 Custom Pattern

Some data flows have a unique interaction pattern that does not fit a normal pattern. These data flows define a custom data flow pattern that is specific to their needs. For an example, see Figure B.54.

A.8.4 Instances and Key Values

Some topics have a single value that applies to the entire microgrid. Other topics have different values for each device in the microgrid or even for each power connector in a single device. When a topic has different values at multiple locations, the different values are called sample *instances*. The location is distinguished from the rest of the data by marking it as a *key value*. In effect, key values are used to address, index, or "look up" information at different locations in the microgrid. Key values may also be used to improve performance by filtering messages between devices and by maintaining separate send/receive buffers within each device.

When multiple data samples arrive on a topic, the key values are compared to determine whether each sample represents a new instance or an update to an existing instance. A conservative bound on the number of possible instances can be determined statically by looking at the maximum number of devices, configurations, and ports. A tighter bound can be determined dynamically by looking at the device announcements.

Six data types are used as key values. These are outlined in Table A.2 and explained further in the following paragraphs.

Table A.2: Overview of all data types used as key values.

	Contents				
Data Type	Origin	Target	Config	Port	Definition
`Identity`	Yes	No	No	No	Section B.2.2.2
`GridRequest`	Yes	No	No	No	Section B.11.2.5
`DeviceRequest`	Yes	Yes	No	No	Section B.6.2.6
`DeviceConfigRequest`	Yes	Yes	Yes	No	Section B.14.2.10
`PowerPortConfigRequest`	Yes	Yes	Yes	Yes	Section B.17.2.14
`Reply`	Yes	Yes	Yes	Yes	Section B.2.2.1

1. `Identity`. This key represents the device publishing its status to the rest of the microgrid.

2. `GridRequest`. This key represents the device issuing a request to all devices on the microgrid.

3. `DeviceRequest`. This key represents the device issuing a request and the device that should act on it.

4. `DeviceConfigRequest`. This key represents the device issuing a request, the device that should act on it, and the configuration that should be updated.

5. `PowerPortConfigRequest`. This key represents the device issuing a request, the device that should act on it, the configuration and power port that should be updated.

6. `Reply`. This key contains values that are copied from the related request. Values not present in the request are filled with defaults.

A.8.5 Timing

Effective microgrid communications requires a careful balance of resource utilization. Information must be sent promptly and without delay while keeping message size and frequency within resource constraints.

Four parameters define the timing requirements for each topic:

- **Data Trigger.** Pre-defined conditions that cause sample updates. Fixed rate topics are triggered by a periodic timer set to a nominal rate. Variable rate topics are triggered by a given type of event.

- **Rate Class.** Bounds the time interval between successive sample updates. A *rate limit* prevents excessive resource utilization. A *deadline* ensures regular updates.

- **Burst Size.** Number of successive sample updates that can be sent faster than the rate limit. Allows normally slow rates to occasionally operate at a higher rate.

- **Quality of Service (QoS) Profile.** Policy for how sample updates are prioritized, buffered, sent, and re-sent as messages. Accounts for constraints in network connectivity. The QoS profile may include some trigger, rate class, or burst size information for use by the communications middleware, such as OMG DDS.

These timing requirements shall be used to define behavioral requirements and to derive minimum resource allocations for hardware and software in TMS devices. The application software, middleware, operating system, processor, memory, and network hardware must all work together to meet these timing requirements.

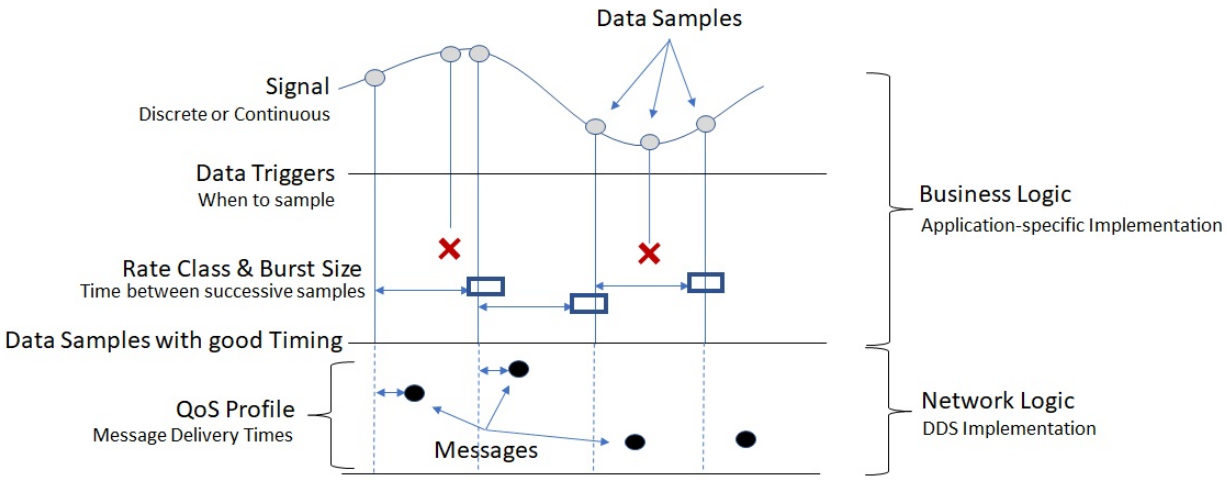


Figure A.10: Illustration of Sample Update and Message Timing

Timing requirements apply separately to each topic, publisher, and sample instance. There are no constraints on timing between topics, except for paired request/reply topics. There are no constraints on timing between publishers, though the subscriber may support a limited number of publishers and discard messages when that number is exceeded. There are no constraints on timing between sample updates, unless otherwise specified.

Some timing requirements are intentionally not specified in this interface standard. For example, there are very few requirements on the timing between different devices or topics so they can be sequenced together in creative ways. Also, all timing requirements are defined in terms of elapsed time so there is no dependency on clock synchronization.

Procurement documents will specify the maximum delay from the time when a device is powered on until the time when it starts sending messages. This maximum delay may depend upon operational requirements and equipment types. Simpler fixed-purpose equipment may have an objective delay of 3 s or less, as larger numbers need to be quickly deployed. Other general-purpose equipment may have an objective delay of a few minutes or less, as smaller numbers are used less often or with less urgency.

The rate classes are consistent with the data triggers. Fixed-rate data triggers are matched to rate classes with the same nominal rate. Variable-rate data triggers are matched to a rate class based on the sustained rate. These relationships are shown in Table A.3.

Table A.3: Relationship between Timing Parameters.

Data Trigger	Rate Class	Deadline	Burst Size	QoS
At Start	Once	No	1	Publish Last
At Start, On Change	*varies*	No	1 or 10	Publish Last
As Needed	*varies*	No	1 or 10	Command profile
Within 0.1 s	*varies*	No	1 or 10	Response profile
Periodic 0.1 s	0.1s	Yes	1	Continuous
Periodic 1 s	1 s	Yes	1	Medium
Periodic 10 s	10 s	Yes	1	Slow
Periodic 900 s	900 s	Yes	1	Rare

This standard does not define any data flows that would require the use of real-time networking protocols. Instead, tasks with hard real-time constraints, such as synchronizing phases or opening a protective breaker, are defined so they can be implemented within a single device.

A.8.5.1 Data Triggers

Every topic has an assigned *data trigger*, a pre-defined condition that causes the publisher to send new data samples and thus new messages. The data trigger is specified in the timing table for each data flow and it is shown underneath each topic arrow in the data flow diagrams.

Publishers should send sample updates when indicated by the data trigger. When the publisher sends sample updates, as indicated or for any other reason, the aggregate traffic shall remain within the specified rate class and burst size.

Eight data triggers, four variable rate and four fixed rate, are defined:

- **At Start.** Values are set during startup. Messages containing the startup value are sent to other devices that connect at a later time. Values are "constant" during normal operation and may only change when the device restarts.
- **At Start, On Change.** Values are set during startup and may update during operation of the device, such as in response to received commands or internal measurements. Each device is responsible for monitoring its internal state and updating these values promptly, subject to the rate class.
- **As Needed.** Values are set and updated in response to manual actions or automatic logic. There is no requirement for a startup value.
- **Within 0.1 s.** Values update immediately after receiving another message with allowance for a 0.1-second response time. Generally used for reply topics.
- **Periodic 0.1 s.** Values update every 0.1 second. If the value hasn't changed, the sample update re-asserts the same value at the present time.
- **Periodic 1 s.** Values update every second. If the value hasn't changed, the sample update re-asserts the same value at the present time.
- **Periodic 10 s.** Values update every 10 seconds. If the value hasn't changed, the sample update re-asserts the same value at the present time.
- **Periodic 900 s.** Values update every 900 seconds, equivalently every 15 minutes. If the value hasn't changed, the sample update re-asserts the same value at the present time.

A.8.5.2 Rate Class

Every topic has an assigned *rate class*, pre-defined limits on the time interval between successive sample updates. The rate class is specified in the timing table for each data flow but not shown in the data flow diagrams.

All topics have a minimum separation time interval called a *rate limit* that prevents excessive resource utilization. Rate limits require some amount of time to pass between successive sample updates thus reserving time for each update to be handled before the next arrives. Topics with a periodic data trigger also have a maximum separation time interval called a *deadline* that ensures regular updates. The rate limit and deadline are enforced at the publisher and may be enforced at the subscriber or elsewhere in the network.

Publishers shall ensure that the time between successive sample updates is greater than or equal to the rate limit for every topic. Publishers should use a timer or rate limit algorithm to ensure adequate time between samples, possibly delaying or merging sample updates as needed. Subscribers or network routers may discard samples that arrive too quickly.

Publishers shall ensure that the time between successive sample updates is less than or equal to the deadline for every topic having a periodic data trigger. Deadlines do not apply to any topic having a non-periodic data trigger. Publishers should re-publish old values as necessary before a timeout occurs. Subscribers or network routers may trigger an alert or fault recovery action when samples arrive too slowly.

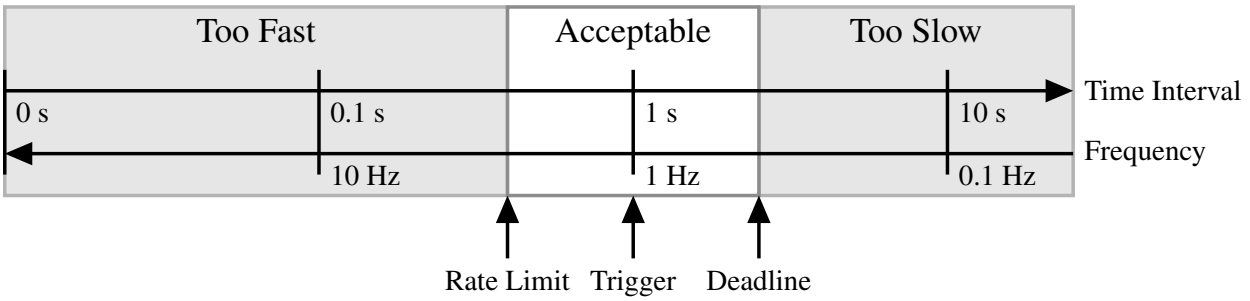


Figure A.11: Example timing for a periodic 1 s topic.

The publisher should allow at least 10 ms margin in the rate limit and deadline for network jitter, changes in the end-to-end timing of successive messages. The subscriber is not required to allow any margins. Example timings are shown in Figure A.12 and Figure A.13.

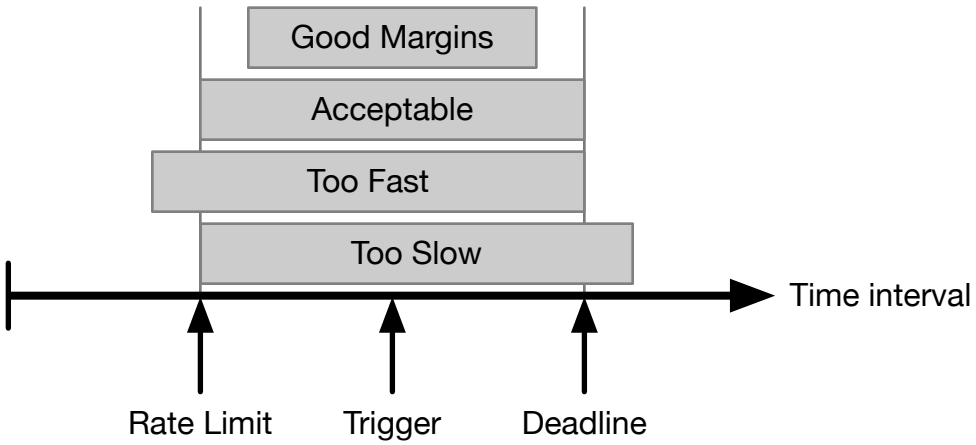


Figure A.12: Example publisher send windows.

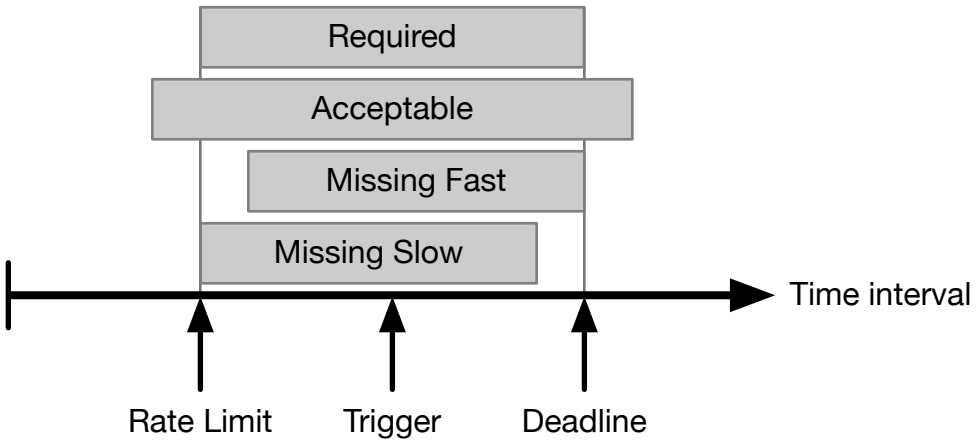


Figure A.13: Example subscriber receive windows.

Six rate classes are defined:

1. **All.** All update rates are acceptable, with no rate limit or deadline.
2. **0.1 s.** Nominal update rate of 0.1 seconds or slower.
3. **1 s.** Nominal update rate of 1 seconds or slower.
4. **10 s.** Nominal update rate of 10 seconds or slower.
5. **900 s.** Nominal update rate of 900 seconds, equivalently 15 minutes, or slower.
6. **Once.** No nominal update rate. Only a single value may be sent.

A small set of pre-defined rates was selected to provide broad coverage and simple implementation. The 0.1 s rate class is fast enough to coordinate load sharing, even with 30% sample loss. Faster rates may require specialized hardware or software. The 900 s rate class is slow enough that network traffic is negligible and long-term logging is feasible. Slower rates can use the 900 s rate class.

Each rate class is defined in terms of four parameters, including two guideline values and two constraints.

1. **Nominal Rate.** Nominal separation time interval between successive sample updates.
2. **Jitter.** Allowed variation in timing between sample updates.
3. **Rate Limit.** Minimum separation time interval between successive sample updates. Applies to all data triggers. Limits the maximum rate.
4. **Deadline.** Maximum separation time interval between successive sample updates. Only applies to periodic data triggers. Limits the minimum rate.

Table A.4: Specifications for each rate class.

	Guidelines		Constraints	
Rate Class	Nominal Rate	Jitter	Rate Limit	Deadline
All	≥0 s	N/A	0 s	Infinite
0.1 s	≥0.1 s	≤0.01 s	0.09 s	2 s
1 s	≥1 s	≤0.1 s	0.9 s	3 s
10 s	≥10 s	≤0.5 s	9.5 s	20 s
900 s	≥900 s	≤50 s	850 s	2000 s
Once	N/A	N/A	Infinite	Infinite

Implementation Note: Many TMS devices will not require any special effort to stay within the rate limits. For those that do, ITU-T Recommendation I.371 provides efficient virtual scheduling algorithms and leaky bucket algorithms that may be used to implement rate limiting. The TMS timing is compatible with the Generic Cell Rate Algorithm (GCRA), Sustained Cell Rate (SCR), and Maximum Burst Size (MBS).

A.8.5.3 Burst Size

Every topic has an assigned *burst size*, allowing sample updates to exceed the rate limit for short periods of time, while still averaging to stay within the rate limit over longer periods of time. The burst size is specified in the timing table for each data flow but not shown in the data flow diagrams.

Some operator inputs, transitions, and alarms have short periods of high activity between long periods of little or no activity. Periodic sampling at a high rate would be inefficient and sampling at a low rate would introduce unacceptable delay. It is generally preferred for sample updates to stay within the rate class but rapid activities may justify faster updates as long as resources are not overwhelmed. For example, an operator may change a configuration setting multiple times before finding the right value or a power device may sequence through multiple states while completing a request or responding to a power event.

Publishers may send sample updates in bursts. Subscribers may respond to bursts by not fully responding to each sample update, effectively treating newer samples as higher priority than older samples. For example, successive requests to stop and then start may result in a

device staying on rather than stopping briefly. Subscribers should reply to every request in a burst, by looking at the aggregate of the burst and doing the most recent request. Further requirements may be added by procurement documents.

Two burst sizes are defined:

1. **One (1).** A single message may exceed the rate limit.
2. **Ten (10).** Ten messages may exceed the rate limit.

The burst size and rate limit interact in the following manner. This description uses a timer and burst count. Other implementations are permitted.

1. Initialize the burst count to the specified burst size.
2. When a sample update would exceed the rate limit, check that the burst count is greater than zero, then count down by one.
3. When a sample update is skipped, two nominal rate time intervals elapse without a sample update in the middle, then count up by one but do not exceed the burst size.

A.8.5.4 Quality of Service (QoS)

Every topic has an assigned *QoS profile* that specifies how data samples are sent as messages. There are many aspects of network communications, with design tradeoffs between performance and resource utilization. Each QoS profile specifies a set of design decisions for these tradeoffs.

Eight QoS profiles are defined:

1. **PublishLast.** Used for topics with an "At Start" or "On Change" data trigger. Provides reliable delivery of the most recent sample, even if the subscriber joins after the data was published.
2. **Command.** Used for topics with an "As Needed" data trigger. Provides reliable delivery of the most recent sample, only to subscribers that join before the data is published.
3. **Response.** Used for topics with a "Within 0.1 s" data trigger. Provides the same performance as the Command profile, so requests and replies have the same behavior.
4. **Reply.** Used for the `Reply` topic. Provides performance the same as the Response profile, except the history is increased to support replies from multiple request topics.
5. **Continuous.** Used for topics with a "Periodic 0.1 s" data trigger. Provides best effort delivery of each sample, with a 2 s delivery deadline.
6. **Medium.** Used for topics with a "Periodic 1 s" data trigger. Provides best effort delivery of each sample, with a 3 s delivery deadline.
7. **Slow.** Used for topics with a "Periodic 10 s" data trigger. Provides best effort delivery of each sample, with a 20 s delivery deadline.

8. **Rare.** Used for topics with a "Periodic 900 s" data trigger. Provides best effort delivery of each sample, with a 2000 s delivery deadline.

Table A.5 and the following paragraphs provide a high-level definition of the QoS profiles. Detailed requirements for each QoS profile are given in Section D.5.

Table A.5: High-level definitions of QoS Profiles.

QoS Profile	Durability	Reliability	History	Deadline	Priority
PublishLast	Transient	Reliable	1	Infinite	Normal
Command	Volatile	Reliable	1	Infinite	Normal
Response	Volatile	Reliable	1	Infinite	Normal
Reply	Volatile	Reliable	128	Infinite	Normal
Continuous	Volatile	Best effort	1	2 s	Normal
Medium	Volatile	Best effort	1	3 s	Normal
Slow	Volatile	Best effort	1	20 s	Normal
Rare	Volatile	Best effort	1	2000 s	Normal

Durability The durability QoS policy controls the delivery of sample updates sent before a subscriber connects to the publisher. *Volatile* delivery sends sample updates to subscribers that join before the data is published, and no attempt is made to send these updates to late-joining subscribers. *Transient* delivery sends sample updates to subscribers that join before the data is published, and the publisher re-sends sample updates to late-joining subscribers. All sample updates are lost when a publisher turns off.

Reliability. The Reliability QoS policy controls whether sample updates are re-sent if messages are lost on the network, thus ensuring continuity of communication and message receipt. *Best effort* means the publisher will attempt to transmit a sample update but not confirm that it was received by the subscriber. *Reliable* means the publisher will maintain a copy of the sample update and periodically resend it until the subscriber acknowledges delivery.

History. The history QoS policy controls how many data samples are buffered on the publisher or subscriber to support other QoS options such as durability or reliability. In general, this is set to a buffer depth of 1 so the most recent sample update is delivered as soon as possible. If the history depth fills faster than messages are delivered, then old data samples will be discarded.

Deadline. The deadline QoS policy controls the maximum latency for message delivery. Missing the deadline may cause a subscriber to signal a communications fault.

Priority. At present, there is no explicit prioritization of topics. The data rates and delivery deadlines may influence prioritization. The following ordering is recommended as a guideline.

1. Command
2. Response
3. Reply
4. Continuous
5. Medium
6. Slow
7. Rare
8. PublishLast

A.8.5.5 Order of Delivery

Order of delivery is the order in which the subscriber receives sample updates from publishers. During delivery, the order the subscriber receives may vary from what the publisher sent. Sample updates are handled in parallel, and messages may get re-ordered due to packet loss or parallel paths in the network. These requirements govern what the publisher and subscriber must do to detect and fix ordering issues. Out of order execution may result in incorrect or unexpected microgrid operation. By defining what order of delivery will or will not be required, all devices can have consistent expectations.

All sample updates from the same publisher and topic shall be received in order. Each publisher guarantees the order of delivery for the sample updates that it sends on a topic-by-topic basis. Each subscriber skips or delays sample updates from a publisher depending on the topic reliability.

This standard does not require specific order of delivery across multiple topics. As a result, a publisher may send sample updates to multiple topics in one order and a subscriber may receive them in another order. Application software shall not rely on ordered delivery of sample updates across topics. When ordered execution of requests is required across topics, application software shall wait for replies to earlier requests before sending later requests.

This standard does not require specific order of delivery across multiple publishers. As a result, multiple publishers may send sample updates to a topic in one order and a subscriber may receive them in another order. Furthermore, different subscribers may see different orders across multiple topics or publishers. Application software shall not rely on ordered delivery of sample updates across publishers.

Figure A.14 illustrates an example of two publishers sending sample updates to two topics, indicating the relative timing that each sample update is sent.

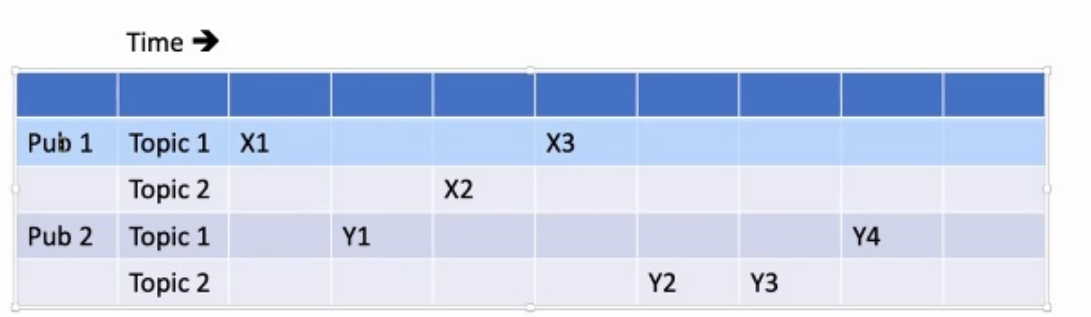


Figure A.14: Order of Delivery Example

In this example, each subscriber:

1. Should always receive X1 before X3 (same publisher and topic)
2. Could receive X2 before X1 or after X3 (same publisher, different topics)
3. Should always receive Y1 before Y4 and Y2 before Y3 (same publisher, topic by topic)
4. Could receive X1 before or after Y1 (different publishers, same topic)

In this example, multiple subscribers:

1. Could see different orders for X1 and X2 (different topics)
2. Could see different orders for X1 and Y1 (different publishers)

A.8.6 Topic Naming Conventions

TMS topics follow a naming convention that indicates the data type and data trigger. Each topic name has three parts:

1. Prefix. Clarify scope when a single data type is used for multiple topics, most TMS topics do not have a prefix.
2. Middle. Identify the data type used by the topic.
3. Suffix. Identify the data trigger according to Table A.6. The data type name usually includes this suffix.

Table A.6: Topic Suffixes

Topic Suffix	Data Trigger
Info	At Start
Reply	Within 0.1 s
Request	As Needed
Result	As Needed
State	At Start, On Change
Update	Periodic

Note that both Reply and Result topics are sent in response to another topic. Reply is used by the normal data flow patterns, and it is sent when a Request is received. Result is used by some custom data flow patterns, and it is sent after another action completes. See Section B.24 for an example Result.

A.8.7 Topic Usage

Devices are not required to publish or subscribe to every topic matching their device role. Some topics may not be implemented in order to reduce cost, simplify operation, reduce cybersecurity risk, or meet some other objective.

There are requirements on when a topic must be implemented, particularly when it must be published. These requirements are intended to reduce integration complexity. In the request/reply pattern, subscribing to the request requires publishing the corresponding reply. In the request/reply with status pattern, subscribing to the request requires publishing both the corresponding reply and status.

All of these requirements can be determined from the information that is published in the device announcement (Section B.5). Conformance testing will verify that publishers and subscribers participate in topics consistent with their `DeviceInfo`. Publishers must update samples according to the specified timing. Subscribers must behave according to the samples received.

Every topic has an assigned *usage*. Three usage categories are defined:

1. **Required.** Implemented by all TMS devices.
2. **Conditional.** Implemented by TMS devices that meet the specified condition.
3. **Optional.** Not required by TMS but may be required by acquisition documents or customers.

Every conditional topic has an assigned condition that identifies when it is required. Nine topic conditions are defined:

1. **HAS_AC_PORTS.** Device has any `tms::PowerDeviceInfo.powerPorts` with at least one `tms::PowerPortInfo.wiring` of AC type.
2. **HAS_AC_METERS.** Device has any `tms::PowerDeviceInfo.powerPorts` with at least one `tms::PowerPortInfo.wiring` of AC type and `hasExternalMeter` or `hasInternalMeter` set true.
3. **HAS_AC_SUMMARY_METERS.** Device has any `tms::PowerDeviceInfo.powerPorts` with at least one `tms::PowerPortInfo.wiring` of AC type and `hasSummaryMeasurementUpdate` set true.
4. **HAS_CONTROL_PARAMETERS.** Device has any `tms::DeviceInfo.controlParameters` with non-zero length.
5. **HAS_DC_PORTS.** Device has any `tms::PowerDeviceInfo.powerPorts` with at least one `tms::PowerPortInfo.wiring` of DC type.

6. **HAS_DC_METERS.** Device has any `tms::PowerDeviceInfo.powerPorts` with at least one `tms::PowerPortInfo.wiring` of DC type and `hasExternalMeter` or `hasInternalMeter` set true.

7. **HAS_DC_SUMMARY_METERS.** Device has any `tms::PowerDeviceInfo.powerPorts` with at least one `tms::PowerPortInfo.wiring` of DC type and `hasSummaryMeasurementUpdate` set true.

8. **HAS_DEVICE_GROUND_PORTS.** Device has `tms::PowerDeviceInfo.grounds` with non-zero length.

9. **SUPPORTS_REQUEST.** Device supports the corresponding request topic.

Every device publishes which conditional and optional topics it implements in the device announcement `TopicInfo`.

A.9 Data Types

The TMS data model standard is designed to be platform-independent, allowing flexibility in system design and implementation.

All data types in the data model represent platform-independent logical types. Platform-specific details such as data APIs and network serialization are defined in the communications profile that is selected to implement the data model. This section describes the generic primitive and constructed data types that are used for the TMS data model. The domain-specific data types are discussed in Chapter B.

The data model messages are pre-defined objects, each having a name, type, and value. The name and type are constant, as defined by TMS. The type identifies possible values for the messages where each message may have a different value.

A.9.1 Type Hierarchy

This section defines the types available for use and the relationship between them. Three categories of types are defined in the data model.

1. **Primitive Types.** Building blocks for individual pieces of information.

2. **Constructed Types.** Building blocks for relationships between information.

3. **Domain-Specific Types.** Primitive or constructed types, used as building blocks or complete data samples, and defined in Chapter B.

A.9.2 Type Names

Every data type is defined with a unique *name*.

When creating data type names for TMS, the names must:

1. Only contain alphabet ('a'–'z' and 'A'–'Z'), digit ('0'–'9'), and underscore ('_') characters
2. Begin with an alphabet character
3. Always be used with the same capitalization (uppercase and lowercase alphabet) as their definition
4. Be unique beyond their capitalization

For example, the names `abc`, `Abc123`, and `a_b_c` are valid. The names `123Abc` and `_abc` are invalid. Both names `abc` and `ABC` are valid, but it is invalid to use both capitalizations within the same type or namespace.

This data model does not define any reserved words. To prevent software development issues, this data model avoids any names that are reserved words in the C++, Java, OMG IDL, or Structured Text programming languages.

A.9.3 Type Namespaces

Every type name exists within a *namespace* that defines the scope of the type definition. Namespace names use the same character rules as type names. Namespaces may contain other, nested namespaces. The top-level, "unnamed" namespace is the *global namespace*. In a single namespace, every definition has a unique name. In multiple namespaces, the same name may be used for different definitions.

When a type name is referenced, the namespace may be explicit (*qualified name*) or implicit (*unqualified name*). A qualified name is written as a concatenation of namespaces, from outer to inner, each separated by a double colon ("::"), and ending with the type name. An unqualified name is simply the type name.

For example, the qualified name `tms::Heartbeat` refers to the `Heartbeat` type within the `tms` namespace. Similarly, the qualified name `tms::dc::MeasurementUpdate` refers to the `MeasurementUpdate` type, within the `dc` namespace, within the `tms` namespace.

Unqualified names are *resolved* by searching for the nearest matching namespace. This search first checks for a match in the namespace where the unqualified name is used, and then checks containing namespaces. If no match is found, then there is an error in the name resolution. Qualified name are resolved by simply traversing the given namespace path, without searching. All qualified names in TMS are fully qualified, and it is an error to only include part of the namespace path.

For example, in both the `tms::ac` and `tms::dc` namespaces, the unqualified name `Heartbeat` resolves to `tms::Heartbeat`. The unqualified name `MeasurementUpdate` resolves to either `tms::ac::MeasurementUpdate` or `tms::dc::MeasurementUpdate`, depending on where it is used. The partially qualified name `dc::MeasurementUpdate` is invalid since it is missing the `tms` namespace.

The notation for qualified names may also be used to reference attributes within data types. In this usage, the attribute name is dropped before performing name resolution for the type.

All data types defined by TMS exist in the `tms` namespace or a nested namespace. All names in the `tms` namespace are reserved for definition by TMS. Table A.7 lists all namespaces defined by TMS.

Table A.7: Namespaces defined by TMS.

Namespace	Purpose
tms	TMS core, including AC power
tms::dc	DC power

TMS uses a naming convention to help distinguish between the different kinds of names. General-purpose data types use short lowercase words. Domain-specific data types use longer "CamelCase" words. Named values use "UPPERCASE_NAMES". Namespaces use lowercase abbreviations.

Change Note: Previous versions used a single dot ('.') instead of the double colon ("::") as the namespace separator. The OMG DDS communications always used the double colon.

A.9.4 Units of Measure

Machine-to-machine communications shall always transmit values using the units of measure specified by the data type. Units of measure are specified by adding a *units* annotation to the data type. All measurements are encoded using metric units. Measurements use the appropriate base or derived unit of measure, without metric prefixes or other scaling.

User interfaces may use more convenient terminology for presentation and operator input, converting to/from the specified units as required. For example, one kilowatt is communicated as 1000 watts, one hour is communicated as 3600 seconds, and one kilowatt-hour is communicated as 3,600,000 joules.

This policy is intended to minimize the risk of conversion errors. It is also in accordance with the guidelines presented in IEEE Std 945. Table A.8 identifies the units of measure used in TMS.

Table A.8: Units of Measure

Abbreviation	**Name**	**Measured Quantity**
A	ampere	Electric current
degC	degree Celsius	Temperature
Hz	hertz	Frequency
J	joule	Energy
L	liter	Volume
ohm	ohm	Electrical resistance
p.u.	per unit	Normalized value
Pa	pascal	Pressure, stress
rad	radian	Plane angle
s	second	Time
V	volt	Voltage
VA	volt ampere	Power (apparent)
var	volt ampere reactive	Power (reactive)
W	watt	Power (real)

In addition to raw units of measure, normalized or "per unit" values are used in situations where the ratio of two values is more important than their magnitudes. For example, a gauge may show when a resource is empty or full, or load sharing may give each source an equal fraction of the load relative to its rated capacity. Per-unit values generally range from 0 to 1, or from -1 to +1 where negative values make sense. While per-unit measurements are effectively unitless, the original unit of measure is often indicated for clarity. For example, "p. u. watts" indicates a power measurement that has been divided by the rated power.

A.9.5 Power Measurement Sign Conventions

The generator sign convention shall be used for all power measurements, regardless of device role or power port directionality. This convention was chosen for consistency with IEEE and IEC standards and with uses such as load sharing and circuit protections.

1. Positive current, both DC and RMS AC, represents energy flowing out of a device and into the microgrid, as produced by a source. Negative current flows into a device, as consumed by a load.

2. Positive real power represents energy flowing out of a device and into the microgrid, as produced by a source. Negative real power flows into a device, as consumed by a load.

3. Positive reactive power represents the current waveform lagging the voltage waveform, as consumed by an inductive load. Negative reactive power represents current leading voltage, as consumed by a capacitive load.

4. Apparent power is the magnitude of the complex real and reactive power vector. It is a positive value and may be calculated as the square root of the real power squared plus the reactive power squared.

5. Phase angle is the difference between the voltage and current waveform angles. Positive phase angle rotates from positive real power towards positive reactive power. The range is from $-\pi$ to π.

These sign conventions are illustrated in Figure A.15 and example measurements are given in Figure A.16.

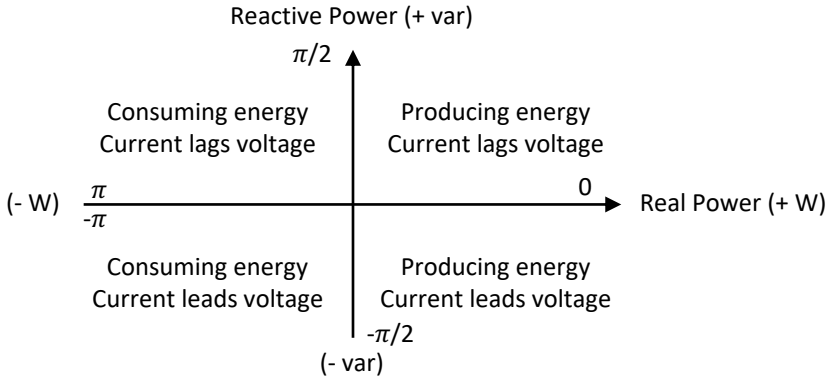


Figure A.15: Power measurement sign conventions.

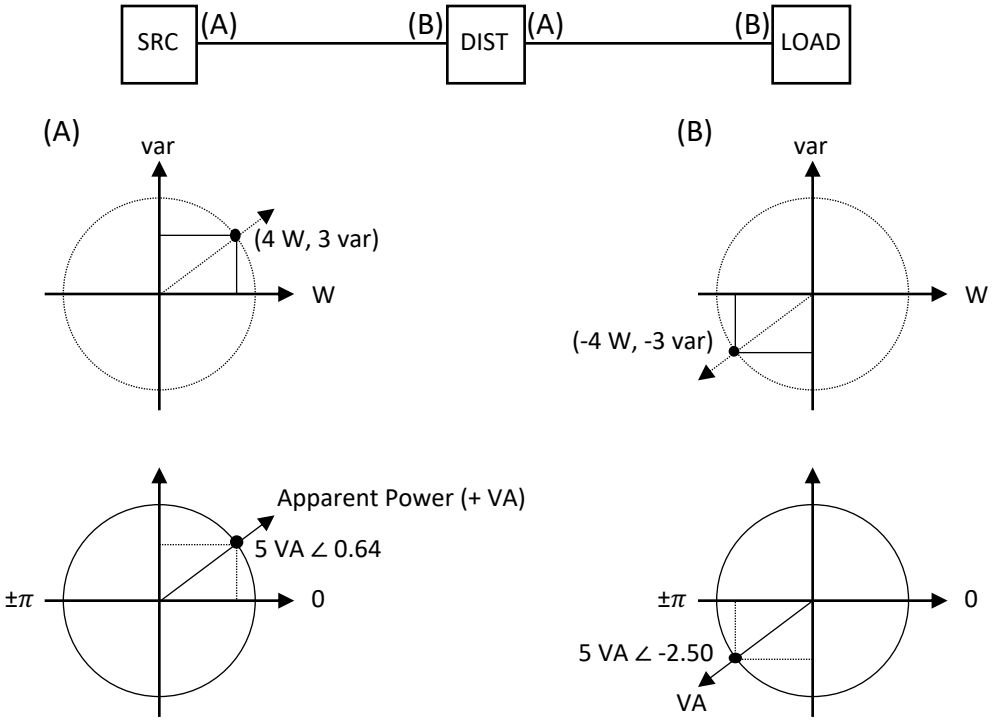


Figure A.16: Example measurements of a 0.8 power factor inductive load.

A.9.6 Primitive Data Types

Primitive data types, also known as atomic, scalar, or simple data types, hold values that have a single assignable part. Primitive data types represent individual pieces of information and are generally used as building blocks in other types. An overview of how primitives are generally used is given in Table A.9.

Table A.9: Overview of Primitive Data Types and their General Usage

Data Type	General Usage
Boolean	True/false values
Character	Human-readable text
Octet	Container for uninterpreted values
Integer Number	Countable, discrete quantities and indices
Floating-Point Number	Measurable, continuous quantities
Enumeration	Named categories and options

A.9.6.1 Boolean

A boolean, or bool for short, is a value that is either true or false.

The data model represents all Booleans as separate, discrete values. Implementations may pack multiple booleans as individual bits within a larger data type. For example, eight separate booleans may be packed together as individual ones and zeros within a single 8-bit integer.

For an example of TMS boolean, see the `hasRealtimeClock` attribute in `tms::ControlHardwareInfo`.

A.9.6.2 Character

A character, or char for short, is an 8-bit value that represents a single UTF-8 code unit in the Universal Coded Character Set (UCS, Unicode), as defined in ISO/IEC 10646. Each UCS character consists of one or more code points, and each code point consists of one or more code units.

TMS devices are only required to properly display the single-unit characters that are in the Portable Character Set, as defined in POSIX.1 / IEEE Std 1003.1 (Table A.10). Control characters other than the newline (U+000A, '\n') should not be used, and strings containing a newline should also be readable if it is replaced by a space. Devices may render other characters with a "missing symbol" character such as a white square (U+25A1). To maximize interoperability, publishers should only use these portable characters.

Table A.10: UCS Portable Character Set

UCS	Char	UCS	Char	UCS	Char	UCS	Char
U+0000	Null	U+0037	7	U+004F	O	U+0067	g
U+000A	Newline	U+0038	8	U+0050	P	U+0068	h
U+0020	Space	U+0039	9	U+0051	Q	U+0069	i
U+0021	!	U+003A	:	U+0052	R	U+006A	j
U+0022	”	U+003B	;	U+0053	S	U+006B	k
U+0023	#	U+003C	<	U+0054	T	U+006C	l
U+0024	$	U+003D	=	U+0055	U	U+006D	m
U+0026	&	U+003E	>	U+0056	V	U+006E	n
U+0027	’	U+003F	?	U+0057	W	U+006F	o
U+0028	(	U+0040	@	U+0058	X	U+0070	p
U+0029	)	U+0041	A	U+0059	Y	U+0071	q
U+002A	*	U+0042	B	U+005A	Z	U+0072	r
U+002B	+	U+0043	C	U+005B	[	U+0073	s
U+002C	,	U+0044	D	U+005C	\	U+0074	t
U+002D	-	U+0045	E	U+005D	]	U+0075	u
U+002E	.	U+0046	F	U+005E	^	U+0076	v
U+002F	/	U+0047	G	U+005F	_	U+0077	w
U+0030	0	U+0048	H	U+0060	‘	U+0078	x
U+0031	1	U+0049	I	U+0061	a	U+0079	y
U+0032	2	U+004A	J	U+0062	b	U+007A	z
U+0033	3	U+004B	K	U+0063	c	U+007B	{
U+0034	4	U+004C	L	U+0064	d	U+007C	|
U+0035	5	U+004D	M	U+0065	e	U+007D	}
U+0036	6	U+004E	N	U+0066	f	U+007E	~

For an example of TMS character, see the `tms::NatoStockNumber`.

A.9.6.3 Octet

An `octet` is an opaque container for an 8-bit value. Communications between devices are guaranteed to preserve the order and value of all eight bits. The value sent always equals the value received. No other interpretation is defined for the contents. In particular, octets are not interpreted as numbers, so there is no notion of “signed” or “unsigned” octets.

For an example of TMS octet, see the `tms::Identity`.

A.9.6.4 Integer Numbers

An integer number, or int for short, is a whole number with zero fractional value. Signed integers can have positive or negative values. Unsigned integers can only store zero and positive values.

Integers are specified using the fixed-width type names defined in POSIX.1 / IEEE Std 1003.1. The “_t” suffix is dropped from these type names for brevity (it was merely added to avoid naming conflicts with legacy source code).

The following integer types are required:

Table A.11: Required Integer Types

Size	Signed	Unsigned
8	int8	uint8
16	int16	uint16
32	int32	uint32
64	int64	uint64

For examples of TMS integers, see the `numNetworkPorts` attribute in `tms::TopicInfo` and both attributes in `tms::ClockMonotonic`.

A.9.6.5 Floating-Point Numbers

A floating-point number, or float for short, is a number that may have a nonzero fractional value. Floating-point numbers are signed values that may be positive or negative. Floating-point numbers also have special "not a number" (NaN) values.

Floating-point numers are defined in IEEE Std 754 / IEC 60559. The only required type is float32, corresponding to "single precision" or binary32 in IEEE Std 754.

Publishers may use a NaN value to indicate that a measurement is not available, whether due to design or fault.

For examples of TMS floats, see the attributes in `tms::ControlHardwareInfo` and `tms::ControlHardwareUpdate`.

A.9.6.6 Enumeration

An enumeration, or enum for short, is a set of named values and each value is unique. Every instance of the enumerated type contains exactly one of these named values.

The first value in every TMS enumeration is UNKNOWN, so default-initialized values can be distinguished from actual commands or measurements. The usage of all values including UNKNOWN may be specified in the definitions of both the enumeration itself and the data types that use it.

The maximum number of values in an enumeration is equal to the number of values that can be stored in an int32. Any mapping between named values and integers should not be relied upon in application software, as it is an implementation detail that may not be portable or interoperable. Implementations should preserve the relative ordering of values as specified in the data model. For example, different implementations may represent enumerations as integers counting from 0, integers counting from 1, or characters.

The following naming convention is used to give each enumerated value a unique name. Enumeration type names start with a capital letter and each new word is capitalized. Enumeration value names start with a capitalized abbreviation of the type name. Each word is capitalized and prefixed with an underscore ('_'). For example, ExampleEnum may contain EE_VALUE1 and EE_VALUE2.

For an example of TMS enumeration, see `tms::DeviceRole`.

A.9.7 Constructed Data Types

Constructed data types, also known as aggregate, compound, or complex data types, hold values that may have multiple, separately assignable parts. Constructed data types represent relationships between information and are generally used to provide richer semantics. Some constructed types are used as building blocks for larger types. All TMS topics use constructed types.

Constructed types may have multiple pre-defined attributes that are unique to the constructed type and provide further definition of the possible values for that type. The meaning of each attribute is specific to the constructed type in which it is used.

A.9.7.1 Structure

A structure, or struct for short, is a logical collection of attributes that are transmitted as a unit. Each attribute has a name and a data type. Attribute names are lexically-scoped within the structure type. Attribute names must be unique within the structure but may be re-used in other structures.

The value of a structure is the collective value of all of the type's attributes. For example, a Date type could have attributes for the year (integer), month (enumeration or 1-12), and day (1-31). The value of this Date type is the year, month, and day as a combined unit.

A naming convention has been established for structures. Structure type names start with a capital letter and each new word is capitalized. Structure attribute names start with a lowercase letter and each new word is capitalized. For example, `ExampleStructure` may contain `exampleAttribute1` and `exampleAttribute2`.

For an example of TMS structure, see `tms::ReplyStatus`.

A.9.7.2 Array

An array is a fixed length, ordered list of elements, all having the same type.

Each array has two attributes, the data type and the length.

The notation for an array is `type[length]`.

Individual elements in an array may be specified by an array index. Ordering must be preserved but indices are not specified. In particular the data model does not use integers to reference specific entried in an array.

A.9.7.3 Sequence

A sequence is a variable length, ordered list of elements, all having the same type. The name "sequence" is consistent with data schemas such as OMG IDL and W3C XSD. Programming languages may use other names for this data type such as array list, dynamic array, ordered collection, and vector.

The notation for a sequence is `sequence<T,minOccurs,maxOccurs>`. The three parameters inside the < and > type template are defined as follows.

- The `T` is the data type of all elements in the sequence.
- The `minOccurs` is the minimum number of elements required by the protocol.

- The `maxOccurs` is the maximum number of elements allowed by the protocol.

Publishers shall set the sequence length between `minOccurs` and `maxOccurs`. Subscribers should verify that the length is between these bounds.

Both `minOccurs` and `maxOccurs` should be checked by sending and receiving devices. Devices may also use `maxOccurs` to reserve memory for messaging. A sequence value that has zero length is called "empty".

A.9.7.4 String

A string is a specialized sequence of characters.

Each string has two attributes, the minimum number of characters and the maximum number of characters.

The notation for a string is `string<minLength,maxLength>`.

A string is nearly equivalent to `sequence<char,minLength,maxLength>`. There are two important differences. The `string` cannot contain a null (U+0000) character. The `string` will be terminated by an additional null character as appropriate for the native programming language.

For examples of TMS string values, see the uses of `tms::String32` in `tms::ProductInfo`.

A.9.7.5 Optional

An optional is a container for a single value that may be present or absent. Each optional has a single attribute specifying the type of the contained value. The notation is `optional<T>`, where `T` is the type of the contained value.

For examples of TMS optional values, see the `controlHardware` and `powerHardware` attributes in `tms::DeviceInfo`.

A.9.7.6 Typedef

A typedef creates an alias name for another datatype. The type represented by the alias name contains the same values as the original datatype. This alias name is used to indicate a special meaning for uses of the type. In TMS, the typedef can be used both as an individual variable in a another type or as a wholly-defined type, where it becomes a constant of that defined type.

For instance, if a sequence type called "SerNum" is defined with a set of attributes, a typedef could be made called "PublicKey" that has SerNum as one of its attributes. Or if a structure called "Date" is defined with attributes of "Month" and "Year" then a typedef is defined called "TestDate" that uses the Date structure. The specific instance of TestDate calls the Date type and binds the Date attributes to it.

This is a useful type for building common domain-specific constructed types that can then be called out in other domain-specific constructed types.

For example, it may be used to specify units for primitive types or to specify fixed attribute values (type and length) for array and sequence types.

For an example of TMS typedef, see `tms::PowerPortNumber`.

A.9.8 Domain-Specific Data Types

As explained in Section A.9.1, primitive and constructed data types are building blocks used to define domain-specific data types. These domain-specific data types specify the actual values that are communicated between devices.

Domain-specific data types are defined in Chapter B using one of the following three patterns.

1. **Enumeration.** Define a new enumeration (Section A.9.6.6) and its named values.
2. **Structure.** Define a new structure (Section A.9.7.1) and the attributes it contains.
3. **Typedef.** Use a typedef (Section A.9.7.6) to define new semantics for an existing type.

A.9.9 Constants

Each domain-specific type may include definitions for one or more constants. Constants are named values that have special semantics when associated with a domain-specific data type.

Each constant is given a unique name. Multiple constants may have the same value, each with its own semantics.

The data model also defines special constants for every constructed types that has a specified length. These are listed in Table A.12.

Table A.12: Naming convention for pre-defined length constants.

Constant Name	**Constant Description**
*TypeName*_LEN	Length of array defined as typedef.
*AttributeName*_LEN	Length of array defined in structure.
*TypeName*_MINLEN	Minimum length of sequence.
*TypeName*_MAXLEN	Maximum length of sequence.

The ratio of a circle's circumference to its diameter, written as π and spelled as "PI", is used to bound some radian measurements. PI has an approximate value of 3.14159; the full precision value should be obtained from a software math library.

For an example of TMS use of constants, see `tms::PowerPortNumber`.

A.9.10 Data Accuracy and Completeness

The data inside of a topic is expected to model reality as closely as possible for optimal situational awareness. Publishers shall provide a value for all non-optional fields within a published topic.

A.9.10.1 Derived Values

In most cases, published values are obtained from design information or direct physical measurement. Data types such as `tms::ac::MeasurementUpdate` provide variations for different wiring configurations rather than trying to use abstract values that hide important details between them.

In some cases, published values may be derived from physical locations other than where they are representated in this data model. For example, a device may use Kirchhoff's current law to calculate the current flowing through one branch of a node using measurements and switch states for other branches of the node. Any such calculated values must be deterministic based on measured values and established physical laws.

During compliance testing, manufacturers may need to provide design artifacts and internal test data to justify derived values.

If a measurement cannot be obtained directly or derived deterministically, then a default value must be used.

A.9.10.2 Default Values

Default values are used when no other specified value is available. Unless otherwise specified, the default values in Table A.13 may be used.

Table A.13: Default Values for all Data Types

Data Type	Default Value
Boolean	False
Character	U+0000 (Null)
Octet	0x00 (Zero)
Integer	0 (Zero)
Float	Not a Number (NaN)
Enumeration	UNKNOWN named value
Structure	Set each attribute to default value
Array	Set each element to default value
Sequence	Zero-length sequence
String	Zero-length string
Optional	Empty (no contained value)
Typedef	See original data type

A.9.11 Annotations

Domain-specific data types and their attributes may have annotations that provide additional strutured information about their semantics. These annotations and their meanings are described below.

A.9.11.1 Extensibility

In consideration of future TMS development, some domain-specific data types may evolve with further functionality, such as new fields in a structure or increased sequence size. To ensure that such development does not break interoperability with previously-deployed systems, future development will follow the three kinds of extensibility types defined in this section.

- **FINAL** - indicates that no future changes may be initiated.

- **APPENDABLE** - indicates that additional range, values, or attributes may be added. All changes must be contained within all elements/members of the existing type and appended to the end whereby existing systems can ignore or cut-off such additional appended information without impact to operations.
- **MUTABLE** - indicates that the type may differ from existing systems in regards to attributes of the type that may be added, removed, or renamed.

In general, most domain-specific types are final, some are appendable, and very few are mutable.

Table A.14 summarizes the appendable values of the extensibility type.

Table A.14: Appendable Extensibility for all Data Types

Data Type	**Appendable Values**
Boolean	N/A
Character	N/A
Octet	N/A
Integer Numbers	Smaller min, larger max
Floating-Point Number	Smaller min, larger max
Enumeration	Add new named value(s) to the end
Structure	Add new attributes to the end
Array	Increase length / add new elements to the end
Sequence	Increase maximum length
String	Increase maximum length
Optional	N/A
Typedef	See original data type

A.9.11.2 Topic Usage

Each domain-specific structure type has a *Topic Usage* naming all topics that use it directly as the specified type, or set to "Nested" if it is only indirectly through another data type. This annotation is not applied to enumeration or typedef types, since they are never used directly as the specified type of a topic.

A.9.11.3 Key Values

As described in Section A.8.4, some structure attributes may be marked by the *keyval* annotation to indicate that they distinguish sample instances. All structure attributes without this annotation are ordinary sample values.

A.9.11.4 Bounded Values

The *min* and *max* annotations indicate bounds on a numeric value. All values X must fall within the range $min \leq X \leq max$ to be valid. Bounds do not apply to the floating point NaN.

A.9.11.5 Units of Measure

The *units* annotation indicates the units of measure for a numeric value. The possible units of measure are listed in Table A.8.

A.10 Resource Allocation

Various features of the data model can be used together to estimate the resources required for microgrid operation.

1. Publishers and subscribers define the communications end points.
2. Topics and instances define the number of samples.
3. Timing defines the rate of each sample.
4. Data types define the size of each sample.

These features should be used to derive hardware and software resource allocation requirements for individual devices. These include but are not limited to the following.

1. Network bandwidth.
2. Processor speed.
3. Memory capacity.
4. Log file storage.

Estimates may be made using static design information and verified using dynamic operating information. Additional resources should be reserved for future upgrades, including new microgrid features and cybersecurity protections.

Procurement documents will require the number of publishers and subscribers that each device must support. For example, see Table A.15.

Table A.15: Example quantities for a small microgrid.

Device Role	Count
MC	2
MD	2
SRC	6
DIST	10
STOR	2
LOAD	10
CONV	2

A.11 Data Model Extensions

This data model is intended to grow over time. This section defines extensibility constraints that are intended to avoid conflicts with future versions of this standard. A device does not conform to the present version of this standard if any of these constraints are not met.

Manufacturers may choose to work with the Government change control process and propose extensions to be included in future versions of this standard. Such standards-track extensions may ignore the following constraints, at the risk of manufacturers needing to re-work their products as part of the standardization process. It is anticipated that standards-track extensions would be added to future versions of this standard.

Manufacturers may choose to make extensions without involving the Government. Such proprietary extensions are subject to the following constraints.

1. Protocols.
   (a) New protocols must use different network ports than those used in this standard. Network utilization must be documented for non-interference.
   (b) Protocol changes must be backwards-compatible with the version used in this standard and should be approved by the appropriate governing body.
2. Data Flows.
   (a) Device roles shall not be granted new publication rights to exising request topics. This would violate the intentional separation of concerns.
   (b) New data flows are encouraged to re-use existing topics, subject to the previous constraint, as long as the timing and data types do not change.
   (c) New topics should be in a different domain, to avoid conflict with future versions of this standard.
   (d) Assigned data types should not be modified. Any new types must define and demonstrate interoperability with the standard types.
   (e) Timing and QoS should not be modified. Any such changes must be compatible with the full range of standard timing.
   (f) Data topics may be left unused, subject to the constraints given in Section A.8.7
3. Data Types.
   (a) New data types should be defined outside the "tms" namespace unless they are proposed for addition to this standard. Any proposed types may be modified or replaced by future versions of this standard.
   (b) Existing data types should not be modified. Such changes may affect resource allocation and could conflict with future versions of this standard. The extensibility annotations are reserved solely for use by this standard. Any modification must be consistent with the existing annotations.
   (c) New annotations may only be added if they interoperate with participants that do not have them.

Any device that extends the data model has the following additional requirements.

1. A simple on-device visual indicator when any extensions are enabled.
2. A simple on-device control to disable all extensions.
3. Publish a warning in ActiveDiagnostics.
4. Design and test artifacts that demonstrate non-interference with standard-conformant devices when the extensions are enabled.
5. Design artifacts explaining how extensions could be added to or interoperate with future versions of this standard. Any constraints imposed by the extension must be clearly documented.

Conformance testing will verify the accuracy of enabled indicator, effectiveness of disabled control, and the ability to fully meet all threshold requirements with the extensions disabled. Additional conformance testing may also be required to demonstrate interoperability with other devices that do not have the extensions.

A.12 Change History

The data model will change over time by adding new functionality, modifying existing functionality, and removing obsolete functionality. Each change to the data model is indicated by a *Change History* that contains the following information.

1. Version. TMS data model version when the change was made.
2. Change. One of the following ways the data model was changed.
   (a) Add. Create a new data type or topic.
   (b) Modify. Change an existing data type or topic.
   (c) Deprecate. Mark an existing data type or topic for eventual deletion. New devices should start migrating to other solutions.
   (d) Delete. Remove an existing data type or topic. New devices should no longer use deleted types or topics except to interoperate with old devices.
3. Interoperability. Yes indicates that the change should be compatible with the previous version of TMS. No indicates there is a known incompatibility, along with one of the following descriptions.
   (a) Xtypes. The change is not interoperable at the communications protocol level.
   (b) Business logic. The change is not interoperable at the business logic level.
4. Description. Brief explanation of what changed.

Where appropriate, longer explanations of interoperability with previous versions of the data model will appear in the sections containing the affected type or topic definitions. Listings of all changes by version are provided in the Change History starting on page 419.

B. Detailed Data Model Requirements

The detailed data model requirements define the data flows and data types used to implement TMS functionality. These definitions are organized into *functional groups*. Multiple functional groups may be used together to achieve higher-level system behaviors. For example, power measurements may cause an MC to update the load sharing.

Each functional group defines the data flows and data types that implement it. The data flows define topics, including their participants and timing. The data types define domain-specific values that can be communicated on each topic. Most domain-specific data types follow the structure or typedef patterns.

Section B.1, Section B.2, and Section B.3 provide introductory material that applies to all functional groups. The following sections define the functional groups. Section B.100 provides cross-references for finding the data types and constants that were defined in the functional groups.

Table B.1 provides an overview of which device roles publish or subscribe to each topic. Table B.2 provides an overview of the usage requirements for each topic. These two tables may be used to find where topics are defined and to help identify which functional groups are relevant to a particular device.

Table B.1: Overview of Topic Participants. P=publisher, S=subscriber.

	Participants							
Topic	MD	MC	SRC	STOR	DIST	LOAD	CONV	Section
AcLoadSharingRequest		P	S	S			S	B.21.1
AcLoadSharingState	S	S	P	P			P	B.21.1
AcMeasurementUpdate	S	S	P	P	P	P	P	B.18.1
AcSummaryMeasurementUpdate		S	P	P	P	P	P	B.18.1
ActiveDiagnosticState	S	P S	P	P	P	P	P	B.12.1
ActiveMicrogridControllerState	S	S	P	P	P	P	P	B.13.1
AuthorizationToEnergizeReply	P S		S	S	S		S	B.24.1
AuthorizationToEnergizeRequest	S		P	P	P		P	B.24.1
AuthorizationToEnergizeResult	S	S	P	P	P		P	B.24.1
ClockState	P S	P S	P	P	P	P	P	B.23.1
ControlHardwareUpdate	P S	P S	P	P	P	P	P	B.8.1
ControlParameterRequest		P	S	S	S	S	S	B.14.1
ControlParameterState	S	S	P	P	P	P	P	B.14.1
DcLoadSharingRequest		P	S	S			S	B.22.1
DcLoadSharingState	S	S	P	P			P	B.22.1
DcMeasurementUpdate	S	S	P	P	P	P	P	B.19.1
DcSummaryMeasurementUpdate		S	P	P	P	P	P	B.19.1
DeviceIcon	P S	P	P	P	P	P	P	B.7.1
DeviceInfo	P S	P S	P S	P S	P S	P S	P S	B.5.1
DiscoveredPowerConnectionState		S	P	P	P	P	P	B.20.1
EnergyStartStopRequest		P	S	S				B.16.1
EnergyStartStopState	S	S	P	P				B.16.1
GroundingCircuitRequest		P	S	S	S	S	S	B.15.1
GroundingCircuitState	S	S	P	P	P	P	P	B.15.1
Heartbeat	P S	P S	P S	P S	P S	P S	P S	B.4.1
IdentityNicknameRequest	P S	S	S	S	S	S	S	B.6.1
IdentityNicknameState	P S	P	P	P	P	P	P	B.6.1
MetricParameterState	P S	P S	P	P	P	P	P	B.14.1
MicrogridPowerConnectionState	S	P						B.20.1
OperatorIntentRequest	P	S						B.11.1
OperatorIntentState	S	P						B.11.1
OperatorPowerConnectionState	P	S						B.20.1
PowerHardwareUpdate	S	S	P	P	P	P	P	B.9.1
PowerPortState	S	S	P	P	P	P	P	B.17.1
PowerSwitchRequest		P			S	S	S	B.17.1
Reply	P S	P S	P	P	P	P	P	B.22.1
StorageUpdate	S	S		P				B.10.1

Table B.2: Overview of Topic Usage.

Topic	Usage	Section
AcLoadSharingRequest	Optional	B.21.1
AcLoadSharingState	Conditional on HAS_AC_PORTS	B.21.1
AcMeasurementUpdate	Conditional on HAS_AC_METERS	B.18.1
AcSummaryMeasurementUpdate	Conditional on HAS_AC_SUMMARY_METERS	B.18.1
ActiveDiagnosticState	Required	B.12.1
ActiveMicrogridControllerState	Required	B.13.1
AuthorizationToEnergizeReply	Conditional on SUPPORTS_REQUEST	B.24.1
AuthorizationToEnergizeRequest	Optional	B.24.1
AuthorizationToEnergizeResult	Optional	B.24.1
ClockState	Optional	B.23.1
ControlHardwareUpdate	Optional	B.8.1
ControlParameterRequest	Optional	B.14.1
ControlParameterState	Conditional on HAS_CONTROL_PARAMETERS	B.14.1
DcLoadSharingRequest	Optional	B.22.1
DcLoadSharingState	Conditional on HAS_DC_PORTS	B.22.1
DcMeasurementUpdate	Conditional on HAS_DC_METERS	B.19.1
DcSummaryMeasurementUpdate	Conditional on HAS_DC_SUMMARY_METERS	B.19.1
DeviceIcon	Optional	B.7.1
DeviceInfo	Required	B.5.1
DiscoveredPowerConnectionState	Optional	B.20.1
EnergyStartStopRequest	Optional	B.16.1
EnergyStartStopState	Required	B.16.1
GroundingCircuitRequest	Optional	B.15.1
GroundingCircuitState	Conditional on HAS_DEVICE_GROUND_PORTS	B.15.1
Heartbeat	Required	B.4.1
IdentityNicknameRequest	Optional	B.6.1
IdentityNicknameState	Conditional on SUPPORTS_REQUEST	B.6.1
MetricParameterState	Optional	B.14.1
MicrogridPowerConnectionState	Optional	B.20.1
OperatorIntentRequest	Optional	B.11.1
OperatorIntentState	Conditional on SUPPORTS_REQUEST	B.11.1
OperatorPowerConnectionState	Optional	B.20.1
PowerHardwareUpdate	Optional	B.9.1
PowerPortState	Required	B.17.1
PowerSwitchRequest	Optional	B.17.1
Reply	Required	B.22.1
StorageUpdate	Required	B.10.1

B.1 Device Role Details

As described in Section A.6, each TMS device is assigned a single specific device role in the microgrid. The data model supports different power functionality for each device role.

This section provides an overview of how the device roles are represented in the data model. These descriptions are intended to explain how the data model supports the major components, how they may be connected, and how they may be monitored or controlled. These descriptions are not intended to require any specific hardware or software implementation. In particular, not all components are required, components may replaced by others having similar functionality, and other software architectures may be used.

Manufacturers are encouraged to find innovative ways of building devices and using the TMS data model to build more capable systems. As this is an interface standard, all requirements are intended to govern externally visible behaviors and not internal implementations. Detail or performance specifications may add requirements to implement specific functions and behaviors.

B.1.1 Control Device Roles

The TMS Control Devices include Microgrid Controllers (MC), Microgrid Dashboards (MD), and Montiors (MON). Each is discussed individually in this section.

B.1.1.1 Microgrid Controller (MC) Logic

The MC receives instructions from the MD and operating information from power devices, uses this information to build situational awareness of what is happening, decides whether any coordination changes are required, and sends requests to the individual devices as appropriate.

Figure B.1 shows the major components in the Microgrid Controllers (MC) device role. The wide vertical arrows represent incoming and outgoing information with the rounded boxes on these arrows representing what types of topics are involved. The gray square boxes represent functions performed with the white boxes inside these functions representing stored information.

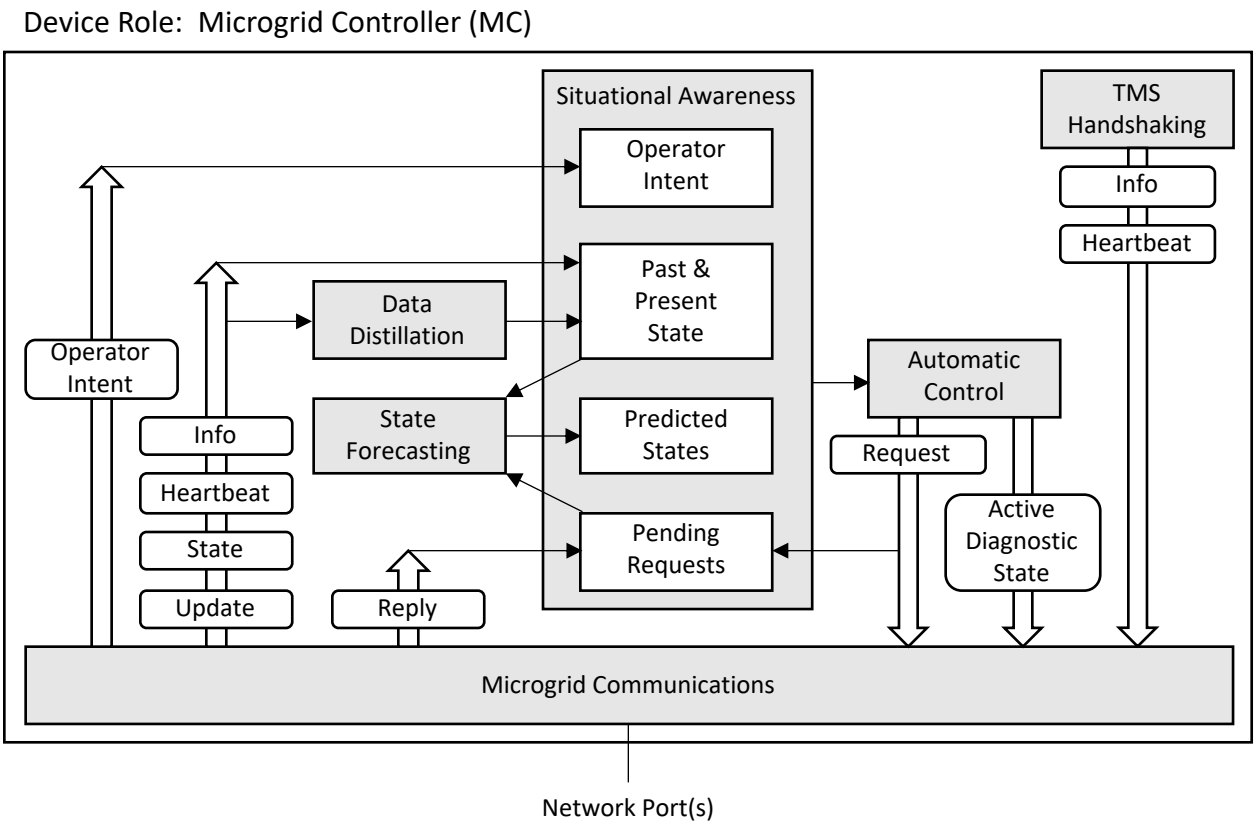


Figure B.1: Conceptual logic of the Microgrid Controller device role.

TMS Handshaking. All microgrid controllers must announce their presence on the grid. The MC must publish its DeviceInfo (Section B.5.1) upon initial power up and also publish a periodic Heartbeat (Section B.4.1) that confirms its own availability to the TMS network.

Situational Awareness. All TMS microgrid controllers maintain a situational awareness of the microgrid that may contain a wide range of information, such as a list of the available devices, the health and status of each device, power flows, available energy, and the automatic controls being performed by the power devices as influenced by the MC. The situational awareness may contain four types of information.

1. **Operator Intent.** Sent by the microgrid dashboard (MD) to inform the MC of the operator's priorities and any actions that need to be taken. While the operator can control each device individually from its front panel, this information allows the operator to coordinate with the MC for optimal performance.

2. **Past and Present State.** Uses incoming health and status information to build the present state of the microgrid. Saves previous information to detect changes or trends. Some information is stored directly as received and some may be provided by a Data Distillation function.

3. **Predicted States.** The State Forecasting function predicts future states of the microgrid. Multiple predictions may be made with different assumptions, such as whether pending requests are processed, load demand changes, or power device failures.

4. **Pending Requests.** A record of requests that have been sent but not yet been replied to. Requests are added when sent and updated when a reply is received. When updated, the information is moved from the Pending Requests state to the Past and Present State.

Data Distillation. The MC aggregates and synthesizes additional information, such as calculating the total operating capacity of the microgrid by adding the operating capacities of each individual source.

State Forecasting. The MC predicts possible future states of the microgrid based on the Past & Present State, Pending Requests that may be accepted or denied, and other possible changes such as load swings or device faults.

Automatic Control. The MC evaluates the operator intent by comparing it to the situational awareness past, present, and predicted states to determine if any changes to the microgrid are needed. If changes are needed, the MC sends out requests to the appropriate devices to make the changes. The MC may also detect grid conditions that require operator action and will send notifications through the ActiveDiagnosticState (Section B.12 to the microgrid dashboard.

Microgrid Communications. This function handles the actual transmission of messages between devices.

B.1.1.2 Microgrid Dashboard (MD) Logic

The MD receives operating information from the MC and power devices, uses this information to build situational awareness of what is happening, displays a simplified view to the operator, accepts operator inputs to control the microgrid, and alerts the operator to potential issues. Some MD devices may be view-only and not allow operator input.

Figure B.2 shows the major components in the microgrid dashboard (MD) device role. The wide vertical arrows represent incoming and outgoing information with the rounded boxes on these arrows representing what types of topics are involved. The gray square boxes represent functions performed with the white shapes within these functions representing stored information.

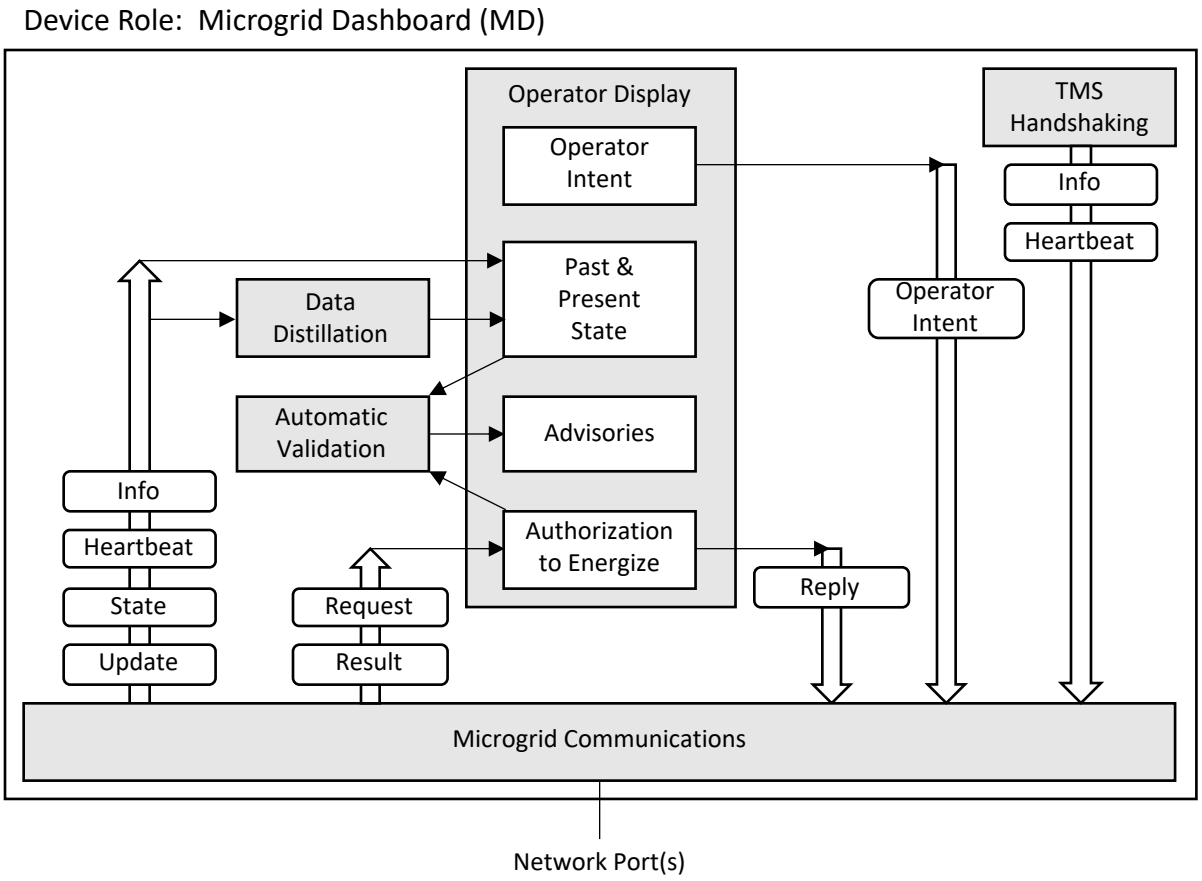


Figure B.2: Conceptual logic of the Microgrid Dashboard device role.

TMS Handshaking. All microgrid dashboards must announce their presence on the grid. The MD must publish its DeviceInfo (Section B.5.1) upon initial power up and also publish a periodic Heartbeat (Section B.4.1) that confirms its own availability to the TMS network.

Operator Display. The MD uses incoming information to build up situational awareness of what is happening. It simplifies and summarizes the raw information in order to present the operator with useful and actionable information. This display also enables the operator to control the microgrid through OperatorIntent (Section B.11).

1. **Operator Intent.** Sent by the microgrid dashboard (MD) to inform the MC of the operator's priorities and any actions that need to be taken. While the operator can control each device individually from its front panel, this information allows the operator to coordinate with the MC for optimal performance.

2. **Past and Present State.** Uses incoming health and status information to build the present state of the microgrid. Saves previous information to detect changes or trends. Some information is stored directly as received and some may be provided by a Data Distillation function.

3. **Advisories.** If problems are found, this results in advisories appearing on the display. These advisories may include things like maintenance warnings, un-served load, potential overload, warnings about actions that could degrade performance, and recommended configuration settings.

4. **Authorization to Energize** If the automatic control detects that changing the state will energize a new circuit in the microgrid, the power device suspends the action, sends an AuthorizationToEnergizeRequest, and waits for an AuthorizationToEnergizeReply from the MD. When the reply is received or times out, the power device either completes or cancels the request. Either way, it sends an AuthorizationToEnergizeResult. See Section B.24.1 for details.

Data Distillation. The MD aggregates and synthesizes additional information, such as calculating the total operating capacity of the microgrid by adding the operating capacities of each individual source.

Automatic Validation. When information is received, the MD compares the state of the microgrid with the operator intent to detect potential operational problems. The MD also checks that the request came from the active MC, performs further validation to make sure the request can be executed, and then replies whether the request was accepted or denied.

Microgrid Communications. This handles the actual transmission of messages between devices.

B.1.1.3 Monitor (MON) Logic

Figure B.3 shows the major components in the Monitor (MON) device role. The wide vertical arrows represent incoming information with the rounded box on the arrow indicating that all topics may be received. The gray shapes represent functions performed.

The Monitor subscribes to all topics, then logs and displays the information received. Since the MON only receives and logs messages, it does not perform TMS handshaking.

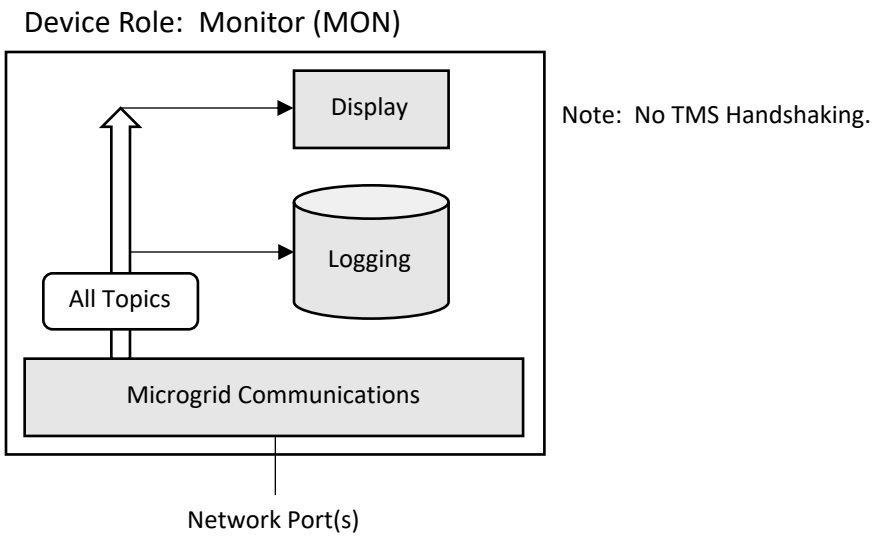


Figure B.3: Conceptual logic of the Monitor device role.

Display. This presents a real-time live display of the microgrid message activity.

Logging. This process logs a persistent record of all microgrid messages capable of recording all messages from all devices.

Microgrid Communications. This handles the actual transmission of messages between devices.

B.1.2 Power Device Roles

The TMS power device roles include Source (SRC), Storage (STOR), Load (LOAD), Distribution (DIST), and Conversion (CONV) devices. This section provides overviews of the power device communications and power port components as well as descriptions of the components supported by each individual device role.

There are three categories of power device roles:

1. Single Port Devices - includes SRC, STOR, and LOAD
2. Multi-Port Devices - includes DIST and CONV
3. Energy Resources - includes SRC and STOR

These categories group device roles that have similar functionality even though they may be implemented in a slightly different way. Functionality is defined individually for each device role and is not impacted by these categories.

All power devices use the sign conventions defined in Section A.9.5.

B.1.2.1 Device Controller Logic

All TMS power devices contain a device controller. Figure B.4 shows an overview of the control logic required for the device controller. The wide vertical arrows represent incoming

and outgoing information with the rounded boxes on these arrows representing what types of topics are involved. The gray square boxes represent functions performed.

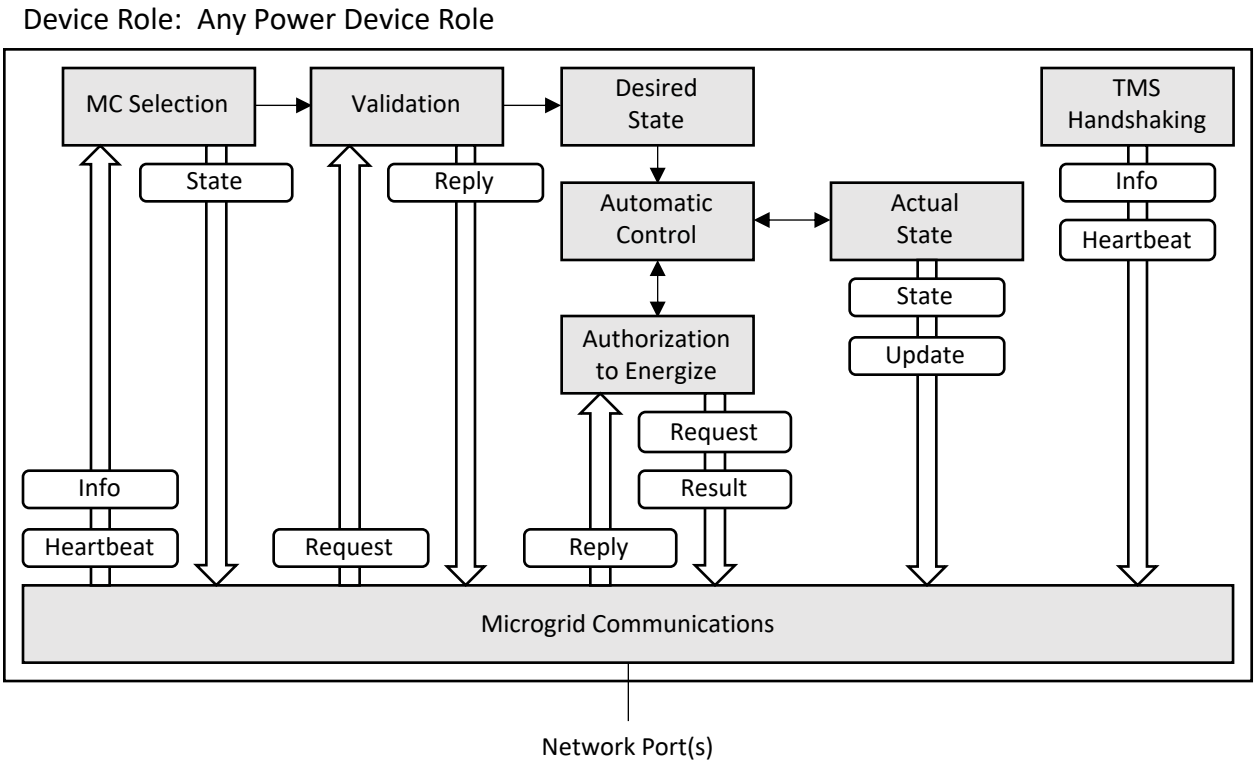


Figure B.4: Conceptual logic of all power device roles.

TMS Handshaking. All TMS power devices must announce their presence on the grid. The power device must publish its DeviceInfo (Section B.5.1) upon initial power up and also publish a periodic Heartbeat (Section B.4.1) that confirms its own availability to the TMS network.

MC Selection. Each power device determines which MC can send it requests according to the rules specified in Section A.7.5. The power device receives handshakes from the microgrid controllers, updates the active MC as appropriate, and sends its ActiveMicrogridControllerState (Section B.13.1).

Validation. When a request is received, the power device checks that it came from the active MC, performs further validation to make sure the power device can execute the request, and then replies whether the request was accepted or denied. If the request is accepted, the desired state is changed.

Desired State. This represents the desired physical behavior of the power device. It initializes to built-in default settings upon initial power up, unless otherwise specified. It may be updated by MC requests or by an operator through a built-in interface. Physical power switch positions are considered to be part of the Actual State.

Actual State. This represents the actual physical behavior of the power device. State messsages are sent when their values change, and update messages are sent at periodic intervals.

Automatic Control. This compares the desired and actual states then performs any changes required. As the actual state changes, the device sends out state and update messages.

Authorization to Energy If the automatic control detects that changing the state will energize a new circuit in the microgrid, the power device suspends the update, sends an AuthorizationToEnergizeRequest, and waits for an AuthorizationToEnergizeReply from the MD. When the reply is received or times out, the power device either completes or cancels the request. Either way, it sends an AuthorizationToEnergizeResult. See Section B.24.1 for details.

B.1.2.2 Power Port Components

All power devices contain one or more power ports. Every power port is identified by a port number that is unique within the device. Most power ports form a circuit between the internal power bus and an external power connector. For multi-port device roles (DIST and CONV), a special BUS port may also be used to provide internal bus measurements.

Figure B.5 shows the major hardware subcomponents in a power port. This schematic is intended to identify the main subcomponents and show how they are usually connected. It is not intended to require any specific implementation. In particular, not all subcomponents are required and substitutions of equivalent subcomponents are allowed.

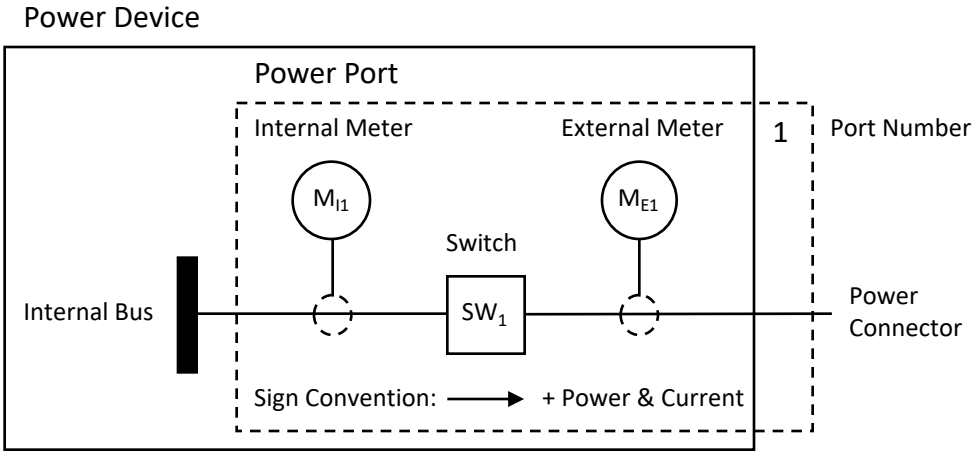


Figure B.5: Components of a power port.

Internal Bus. This represents a connection to another component inside the power device.

Power Port. This outline shows which subcomponents are defined within a power port.

Internal Meter. Takes power measurements inside the switch.

Switch. Controls the circuit continuity.

External Meter. Takes power measurements outside the switch.

Power Connector. Electrical connection to other devices and required for all except the BUS port.

Power Converter. (CONV device role only, not shown in this figure.) Changes the frequency and voltage between ports. May also regulate power flow.

This standard does not require devices to contain all supported components on every power port. Devices announce what ports and components they contain using the `PowerDeviceInfo` and `PowerPortInfo` data types. Detail specifications may require specific ports and components.

B.1.2.3 Source (SRC) Components

Figure B.6 shows the major hardware components in the Source (SRC) device role. This schematic is intended to identify the main components and show how they are usually connected. It is not intended to require any specific implementation. In particular, not all components are required and substitutions of equivalent components are allowed.

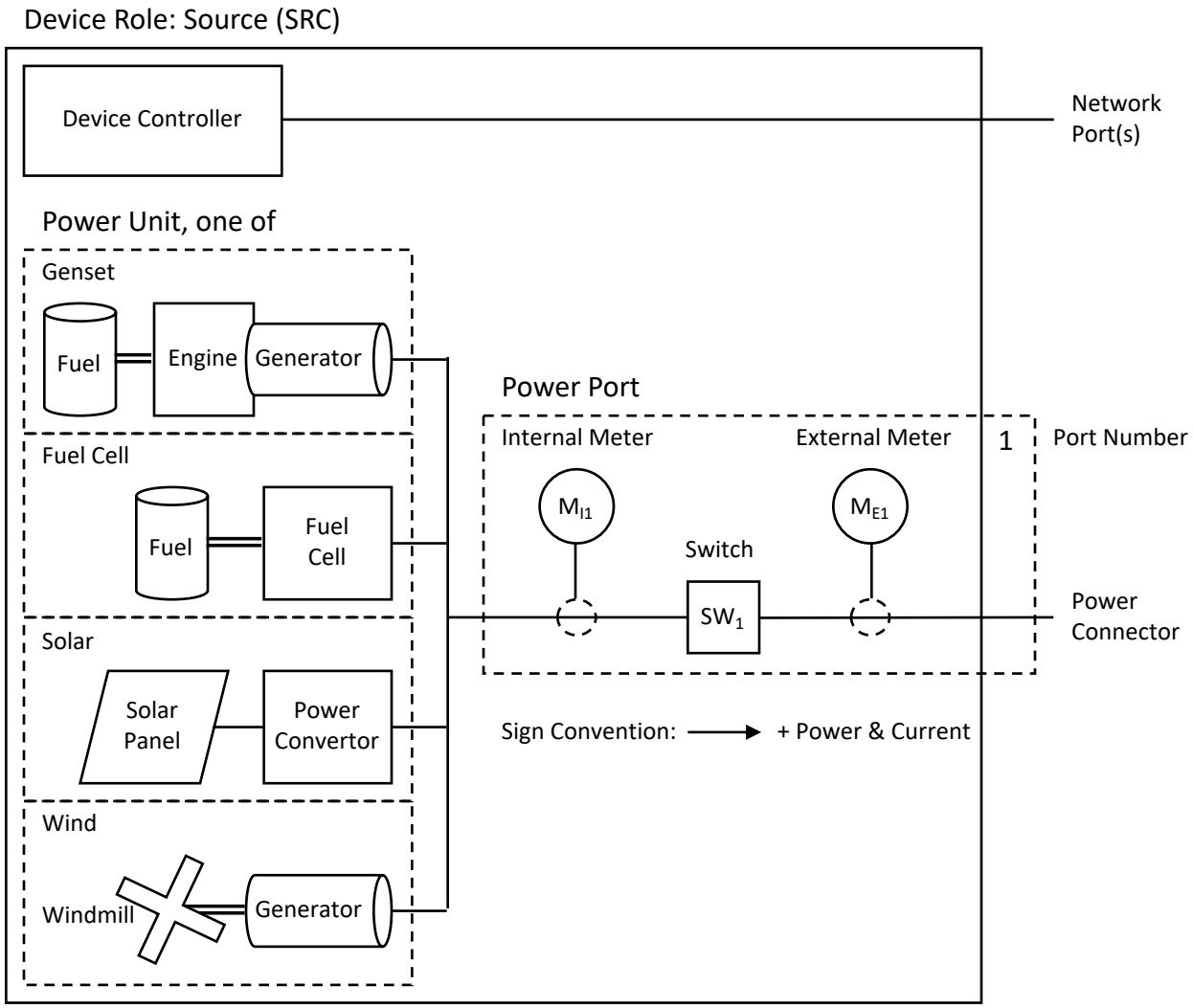


Figure B.6: Schematic diagram of the Source device role.

Device Controller. Every source device shall have a device controller connected to the TMS network. The device controller's function is to send and receive messages, translating as necessary between external messages and internal signals. See Section B.1.2.1.

Network Port(s). Every source device shall have at least one network port. More network ports may be required to allow for connections to multiple TMS devices.

Power Unit. Every source device shall have some form of power unit, usually a generator set (genset), fuel cell, solar, or wind power source. At present, the data model provides the most detailed support for gensets. More support for other power units may be added in future versions of this standard. The data model provides support for monitoring the energy source (fuel level) and also for starting, stopping, and regulating the amount of power produced.

Power Port. Every source device shall have a single power port with an external connector for other devices. The power port may have a switch that isolates the internal components from the external connector. Power measurements are made on the external side of this switch, and optionally, also on the internal side. The data model provides support for opening and closing the switch, measuring power, and syncronizing waveforms. See Section B.1.2.2.

If additional power ports are required, then the device controller should present both the source and distribution device roles, as described in Section A.6.3. The data model provides support for monitoring the connections between devices.

Not shown, every source device may have a grounding circuit, with its associated metering and switching capabilities.

Table B.3 identifies the data model sections that pertain to each component of the source device role.

Table B.3: Data Model Sections for Source Device Role Components

Source Component	**Data Model Sections**
Device Controller	Identity (B.3)
	Heartbeat (B.4)
	Device Announcement (B.5)
	Nickname (B.6)
	Device Icon (B.7)
	Control Hardware (B.8)
	Microgrid Controller Selection (B.13)
	Device Parameters (B.14)
	Clock Status (B.23)
All Hardware	Power Hardware (B.9)
	Diagnostics Reporting (B.12)
	Authorization To Energize (B.24)
Power Unit	Source and Storage Start/Stop (B.16)
	AC Load Sharing (B.21)
	DC Load Sharing (B.22)
Meter	AC Measurements (B.18)
	DC Measurements (B.19)
Switch & Connector	Grounding Circuits (B.15)
	Source and Storage Start/Stop (B.16)
	Power Switches and Connectors (B.17)
	Power Topology (B.20)

B.1.2.4 Storage (STOR) Components

Figure B.7 shows the major hardware components in the Storage (STOR) device role. This schematic is intended to name components and show how they are usually connected. It is not intended to require any particular implementation. In particular, not all components are required and substitutions of equivalent components are allowed.

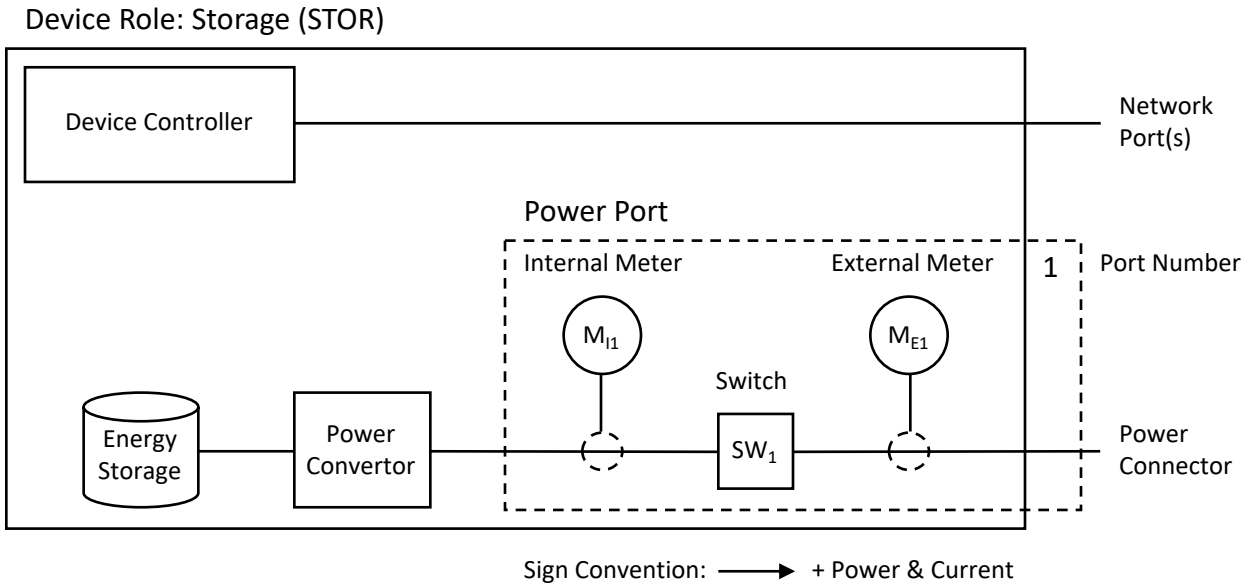


Figure B.7: Schematic diagram of the Storage device role.

Device Controller. Every storage device shall have a device controller connected to the TMS network. The device controller's function is to send and receive messages, translating as necessary between external messages and internal signals. See Section B.1.2.1.

Network Port(s). Every storage device shall have at least one network port. More network ports may be required to allow for connections to multiple TMS devices.

Energy Storage. Storage devices can store energy and estimate how much is available.

Power Convertor. Storage devices can regulate power for the external bus and control the rate of charge or discharge for the energy storage.

Power Port. Every storage device shall have a single power port with an external connector for other devices. The power port may have a switch that isolates the internal components from the external connector. Power measurements are made on the external side of this switch, and optionally, also on the internal side. See Section B.1.2.2.

If additional power ports are required, then the device controller should present both the storage and distribution device roles, as described in Section A.6.3. The data model provides support for monitoring the connections between devices.

Table B.4 identifies the data model sections that pertain to each component of the storage device role.

Table B.4: Data Model Sections for Storage Device Role Components

Storage Component	**Data Model Sections**
Device Controller	Identity (B.3)
	Heartbeat (B.4)
	Device Announcement (B.5)
	Nickname (B.6)
	Device Icon (B.7)
	Control Hardware (B.8)
	Microgrid Controller Selection (B.13)
	Device Parameters (B.14)
	Clock Status (B.23)
All Hardware	Power Hardware (B.9)
	Diagnostics Reporting (B.12)
	Authorization To Energize (B.24)
Energy Storage	Energy Storage (B.10)
	Source and Storage Start/Stop (B.16)
	AC Load Sharing (B.21)
	DC Load Sharing (B.22)
Meter	AC Measurements (B.18)
	DC Measurements (B.19)
Switch & Connector	Grounding Circuits (B.15)
	Source and Storage Start/Stop (B.16)
	Power Switches and Connectors (B.17)
	Power Topology (B.20)

B.1.2.5 Load (LOAD) Components

Figure B.8 shows the major hardware components in the Load (LOAD) device role. This schematic is intended to name components and show how they are usually connected. It is not intended to require any particular implementation. In particular, not all components are required, and substitutions of equivalent components are allowed.

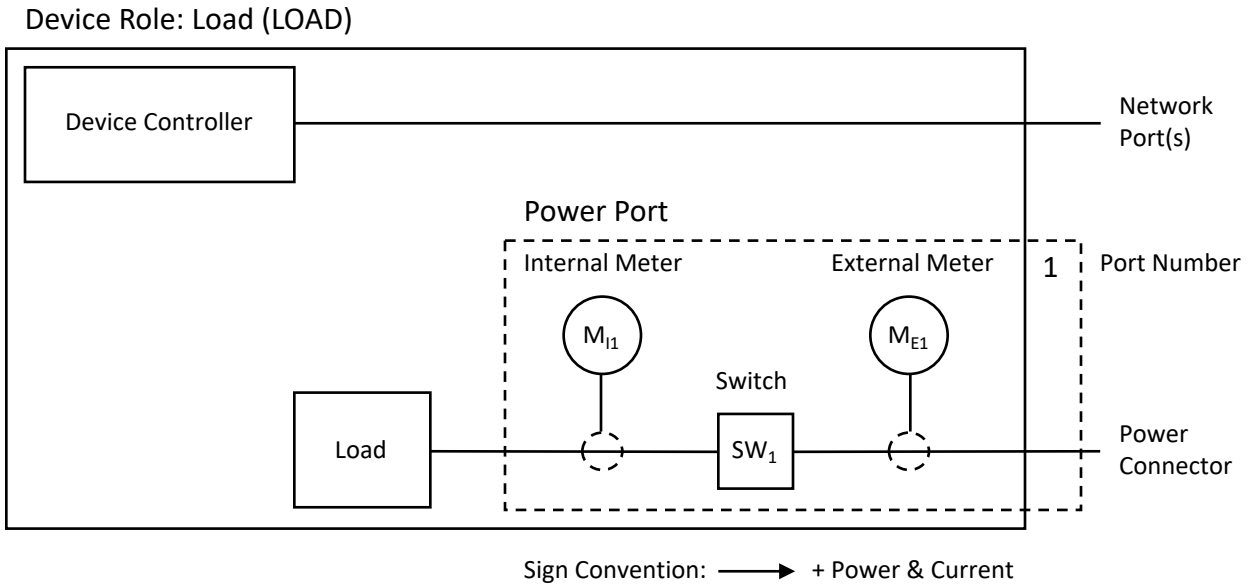


Figure B.8: Schematic diagram of the Load device role.

Device Controller. Every load device shall have a device controller connected to the TMS network. The device controller's function is to send and receive messages, translating as necessary between external messages and internal signals. See Section B.1.2.1.

Network Port(s). Every load device shall have at least one network port. More network ports may be required to allow for connections to multiple TMS devices.

Load. Load device roles contain an electrical load.

Power Port. Every load device shall have a single power port with an external connector for other devices. The power port may have a switch that isolates the internal components from the external connector. Power measurements are made on the external side of this switch, and optionally, also on the internal side. See Section B.1.2.2.

Table B.5 identifies the data model sections that pertain to each component of the load device role.

Table B.5: Data Model Sections for Load Device Role Components

Storage Component	**Data Model Sections**
Device Controller	Identity (B.3)
	Heartbeat (B.4)
	Device Announcement (B.5)
	Nickname (B.6)
	Device Icon (B.7)
	Control Hardware (B.8)
	Microgrid Controller Selection (B.13)
	Device Parameters (B.14)
	Clock Status (B.23)
All Hardware	Power Hardware (B.9)
	Diagnostics Reporting (B.12)
Meter	AC Measurements (B.18)
	DC Measurements (B.19)
Switch & Connector	Grounding Circuits (B.15)
	Power Switches and Connectors (B.17)
	Power Topology (B.20)

B.1.2.6 Distribution (DIST) Components

Figure B.9 shows the major hardware components in the Distribution (DIST) device role. This schematic is intended to name components and show how they are usually connected. It is not intended to require any specific implementation. In particular, not all components are required and substitutions of equivalent components are allowed.

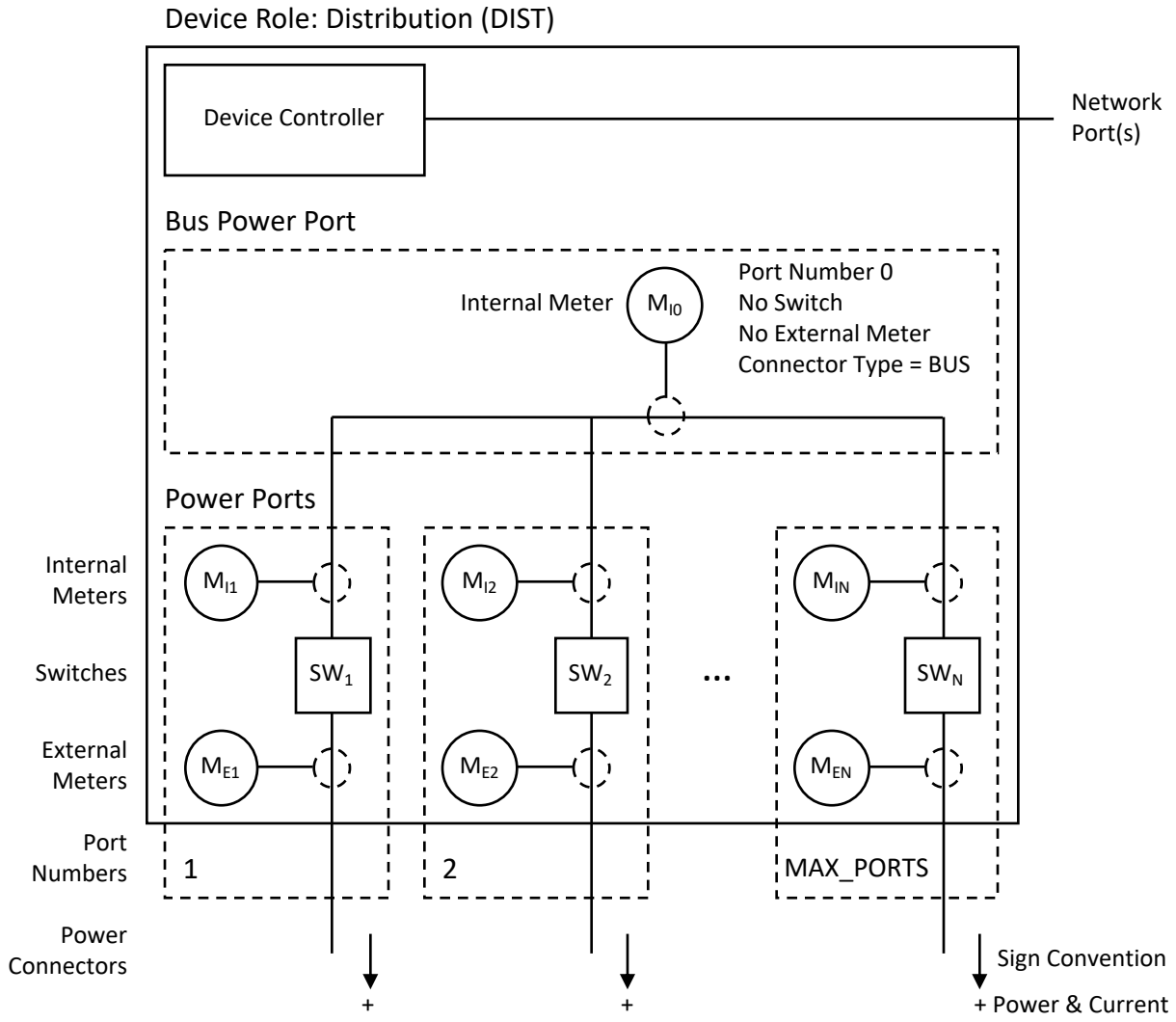


Figure B.9: Schematic diagram of the Distribution device role.

Device Controller. Every distribution device shall have a Device Controller that connects to the TMS network. The Device Controller's function is to send and receive messages, translating as necessary between external messages and internal signals. See Section B.1.2.1.

Network Port(s). Every distribution device shall have at least one network port. More network ports may be required to allow for connections to multiple TMS devices.

Bus Power Port. Distribution devices may also have a special bus power port to allow

for internal power measurements. The bus port does not have an external connector, switch, or meter. It should have an internal meter. The connector type must be set to bus. it allows internal power measurements with no external switches or meters

Power Ports. Distribution devices must have two or more power ports. Examples of two-port distribution include a clamp-on power meter and a pass-through switch. Additional power ports provide greater flexibility in power distribution. The number of ports used is capped by the MAX_PORTS constant, as defined in the `PowerPortNumber` data type (Section B.2.2.4). The power ports may have a switch that isolates the internal components from the external connector. Power measurements are made on the external side of this switch, and optionally, also on the internal side. See Section B.1.2.2.

Table B.6 identifies the data model sections that pertain to each component of the distribution device role.

Table B.6: Data Model Sections for Distribution Device Role Components

Distribution Component	**Data Model Sections**
Device Controller	Identity (B.3)
	Heartbeat (B.4)
	Device Announcement (B.5)
	Nickname (B.6)
	Device Icon (B.7)
	Control Hardware (B.8)
	Microgrid Controller Selection (B.13)
	Device Parameters (B.14)
	Clock Status (B.23)
All Hardware	Power Hardware (B.9)
	Diagnostics Reporting (B.12)
	Authorization To Energize (B.24)
Meter	AC Measurements (B.18)
	DC Measurements (B.19)
Switch & Connector	Grounding Circuits (B.15)
	Power Switches and Connectors (B.17)
	Power Topology (B.20)

B.1.2.7 Conversion (CONV) Components

Figure B.10 shows the major hardware components in the Conversion (CONV) device role. This schematic is intended to name components and show how they are usually connected. It is not intended to require any particular implementation. In particular, not all components are required and substitutions of equivalent components are allowed.

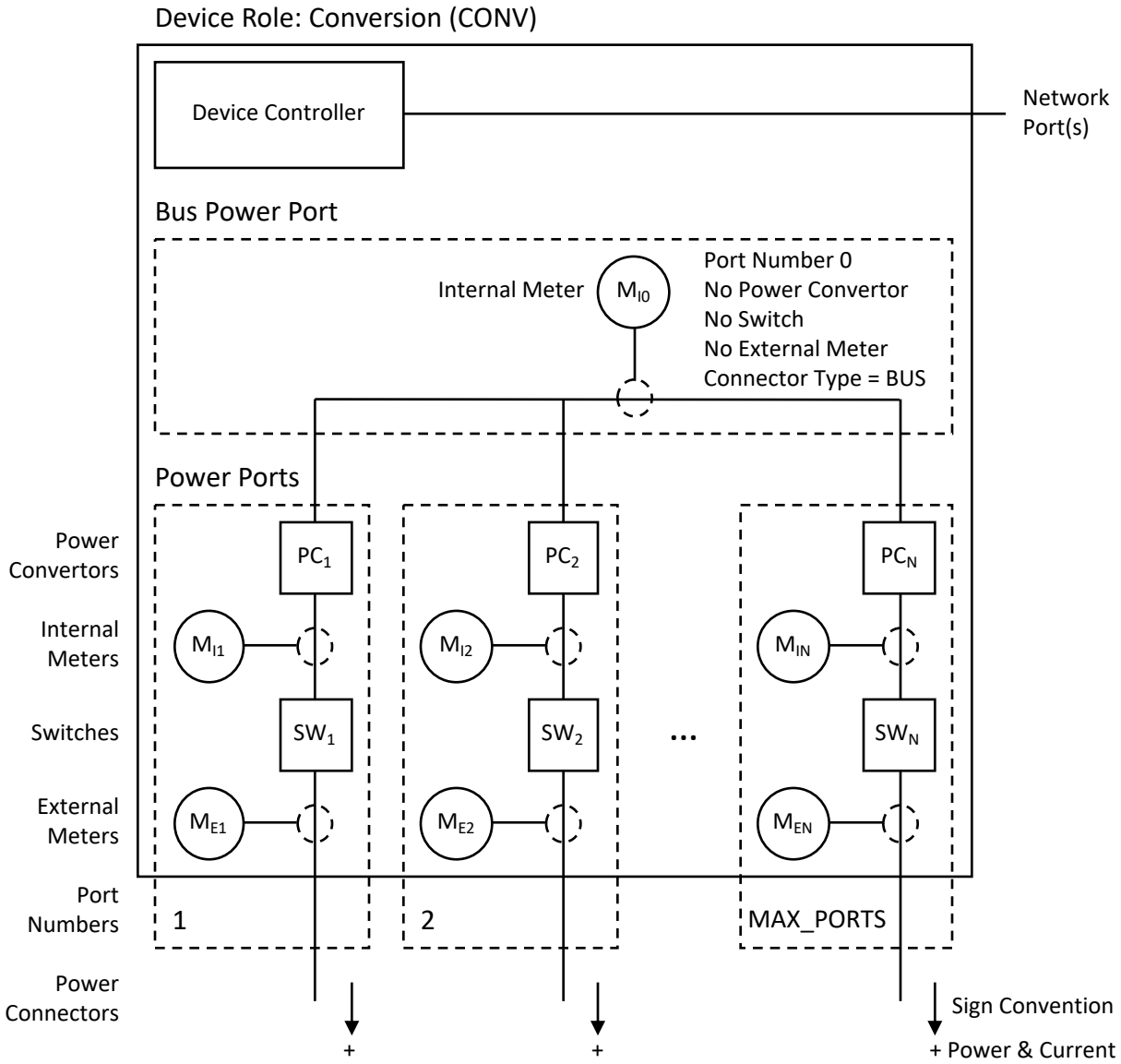


Figure B.10: Schematic diagram of the Conversion device role.

Device Controller. Every conversion device shall have a Device Controller that connects to the TMS network. The Device Controller's function is to send and receive messages, translating as necessary between external messages and internal signals. See Section B.1.2.1.

Network Port(s). Every conversion device shall have at least one network port. More network ports may be required to allow for connections to multiple TMS devices.

Bus Power Port. Conversion devices may also have a special bus power port to allow for internal power measurements. The bus port does not have an external connector, switch, or meter. It should have an internal meter. The connector type must be set to bus. it allows internal power measurements with no external switches or meters

Power Ports. Conversion devices must have two or more power ports. Examples of two-port distribution include a clamp-on power meter and a pass-through switch. Additional power ports provide greater flexibility in power distribution. The number of ports used is capped by the MAX_PORTS constant, as defined in the `PowerPortNumber` data type (Section B.2.2.4). The power ports may have a switch that isolates the internal components from the external connector. Power measurements are made on the external side of this switch, and optionally, also on the internal side. See Section B.1.2.2.

Conversion devices also contain power converters, one per power port, which convert both voltage and frequency. Power converters allow the voltage of each port to be different. The conversion ratio between the internal voltage and the external voltage is controlled by the power converters. the ratio of the voltage between two external connectors is the ratio of the product of the internal convertors

The TMS data model supports two different types of power convertors:

- Active power converters - actively controls the power, such as power electronics that are programmable and can support the load sharing features
- Passive power converters - allows selectable fixed power, such as transformers that only provide a fixed range of ratios, where selection is allowed

Table B.7 identifies the data model sections that pertain to each component of the conversion device role.

Table B.7: Data Model Sections for Conversion Device Role Components

Distribution Component	**Data Model Sections**
Device Controller	Identity (B.3)
	Heartbeat (B.4)
	Device Announcement (B.5)
	Nickname (B.6)
	Device Icon (B.7)
	Control Hardware (B.8)
	Microgrid Controller Selection (B.13)
	Device Parameters (B.14)
	Clock Status (B.23)
All Hardware	Power Hardware (B.9)
	Diagnostics Reporting (B.12)
	Authorization To Energize (B.24)
Power Convertor	AC Load Sharing (B.21)
	DC Load Sharing (B.22)
Meter	AC Measurements (B.18)
	DC Measurements (B.19)
Switch & Connector	Grounding Circuits (B.15)
	Power Switches and Connectors (B.17)
	Power Topology (B.20)

B.2 Request/Response Pattern

The request/response pattern is a common building block for TMS communications. One device, usually a microgrid controller, publishes a request and the target device or devices are responsible for responding.

Every request contains a `requestId` and a `sequenceId`. The `requestId` is a key value (Section A.8.4) that identifies which device sent the request and which devices or power ports are responsible for responding. The `sequenceId` is a counter that increments for each request. Taken together, the `requestId` and `sequenceId` uniquely identify each request and identify the order in which they were issued.

Some `requestId`s contain a `config` that indicates whether it should be executed immediately or saved for deferred action. Any `requestId` that does not contain a `config` is always intended for immediate effect. The `config` is of type `tms::ConfigId` and is published with one of the following three values.

1. `CONFIG_ACTIVE`. Request having immediate effect. Often called a command. May be saved in volatile memory unless otherwise specified.

2. `CONFIG_ON_REBOOT`. Request for effect after the device restarts or reboots and before communications are established. Must be saved in persistent memory.

3. `CONFIG_ON_COMMS_LOSS`. Request for effect after communications loss timeout. May be saved in volatile memory unless otherwise specified.

Requests are subject to timing constraints in order to avoid overloading the network or responding devices. If multiple requests are published without waiting for a reply, some requests may be discarded or executed out of order, depending on what QoS profiles and how many topics are involved. In particular, it is the microgrid controller's responsibility to explicitly sequence requests and wait for replies where there may be ambiguity or uncertainty in the order of execution. In general, the QoS profiles were defined so that a microgrid controller can issue one request of each type to each target device / port at a time.

In the event that a microgrid controller sends a burst of requests to a single key value, the responding device should process all data samples that it receives in the normal fashion. However, depending on the specific timing of events on both devices, some data samples may be discarded by the limited QoS history depth, and so the responding device will not reply to such discarded samples.

The response generally consists of following sequence of steps.

1. **Validate.** Check the request sender, receiver, time, and values.

2. **Reply.** Notify the sender whether the request was accepted or rejected.

3. **Act.** Start executing the request if the `requestId` does not contain a `config` or if the `config` equals `CONFIG_ACTIVE`. Otherwise save the request for deferred execution.

4. **Status Update.** As the request is executed, send event-based or periodic updates of its progress. Most updates are via the related status topic. Some updates may use ActiveDiagnostics or another topic.

Replies copy values from the `requestId` and `sequenceId` so they can be matched with the corresponding request. Status updates usually do not copy `requestId` or `sequenceId` values, but they usually do contain values from other parts of the request when the action completes.

B.2.1 Data Flows

There are no data flows specific to this pattern. The data flow shown below is a template for all request/response data flows. The wildcard *Request represents any request topic and the Reply represents the default reply topic.

Figure B.11 illustrates all the topics in the data flow.

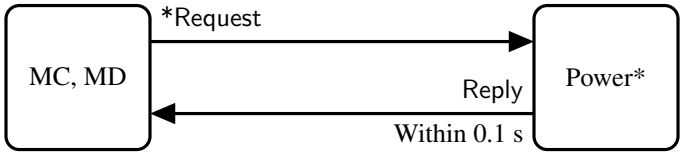


Figure B.11: Request/Response Pattern data flow

Most requests are from the control devices to the power devices and no type-specific return value is expected. All such requests share the same general-purpose Reply topic. For requests that have other needs, another response topic is specified in the data flow.

Table B.8 provides an overview of how each topic is used.

Table B.8: Description of the Request/Response Pattern topics.

Topic	Description
*Request	
Reply	The reply to a request.

Table B.9 specifies the publishers and subscribers for each topic as well as usage requirements as described in Section A.8.7.

Table B.9: Participants of the Request/Response Pattern topics.

Topic	Usage	MD	MC	MON	SRC	STOR	DIST	LOAD	CONV
*Request									
Reply	Required	Pub Sub	Pub Sub	Sub	Pub	Pub	Pub	Pub	Pub

This participants table has the following interpretation.

1. *Request may be subscribed by devices.

2. Reply shall be published by MD, MC, SRC, STOR, DIST, LOAD, and CONV devices.

3. Reply may be subscribed by MD, MC, and MON devices.

Table B.10 specifies the timing for when sample updates are sent to each topic. These parameters are defined in Section A.8.5.

Table B.10: Timing of the Request/Response Pattern topics.

Topic	Data Trigger	Rate Class	Burst Size	QoS Profile
*Request				
Reply	Within 0.1 s	All	1	Reply

B.2.2 Data Types

Table B.11 specifies the data types that are used as sample values for each topic.

Table B.11: Data types of the Request/Response Pattern topics.

Topic	Data Type
*Request	
Reply	`tms::Reply`

Topic-level data types are defined in the order shown in Table B.11. Then nested data types are defined in the order of appearance. Cross-references are provided for data types that have already been defined.

The data types are listed in the order shown in the specifications table above. Nested data types are displayed in the order of appearance. Data types that have been described in previous sections are referenced to the section containing the data type description.

B.2.2.1 `tms::Reply`

PURPOSE: Respond whether a request was accepted for execution or rejected. Supported requestId types include GridRequest, DeviceRequest, DeviceConfigRequest, and PowerPortConfigRequest.

TOPIC USAGE: Reply

EXTENSIBILITY: extensibility(APPENDABLE)

PATTERN: Structure

ATTRIBUTES:

Name	Type and Description

`requestingDeviceId`
: `Identity`
Copy of requestingDeviceId from the requestId.
Annotations: *keyval*

`targetDeviceId`
: `Identity`
Identity of the device that sent this Reply.
Annotations: *keyval*

`config`
: `ConfigId`
Copy of configId from a requestId of type DeviceConfigRequest or PowerPortConfigRequest, otherwise ConfigId.CONFIG_UNKNOWN.

`portNumber`
: `PowerPortNumber`
Copy of portNumber from a requestId of type PowerPortConfigRequest, otherwise PowerPortNumber.INVALID_PORT_NUMBER.

`requestSequenceId`
: `RequestSequence`
Copy of sequenceId from the requestId. The requestingDeviceId and requestSequenceId uniquely match the reply code to the original request.

`status`
: `ReplyStatus`
Decision to accept or reason to reject the request.

B.2.2.2 `tms::Identity`

PURPOSE: Universally unique identifier for a device, person, platform, or other entity.

EXTENSIBILITY: extensibility(APPENDABLE)

PATTERN: Typedef

ORIGINAL TYPE: `string<1,32>`

DESCRIPTION:

These identifiers must be unique. This prevents confusion about which information belongs to which device.

These identifiers should be human readable. Operators may need to authorize devices, verify what devices are present, and verbally communicate in a noisy environment or over a radio. Complex, long names are slow to read, communicate, and compare.

These identifiers should indicate the device type, role, and performance ratings. This assists with communications and verification of authorized devices.

These identifiers should be cryptographically bound to a private key, device role, and performance ratings. This makes it harder to impersonate another device, role, or power rating. For example, the Identity may be an X.509 subject name or subject alternative name, and X.509 extensions may indicate the device role and power ratings.

Different formats may be used by different acquisition programs, as long as the resulting range of values do not overlap. For example, a prefix may indicate the format or acquisition program.

These identifiers may be subject to additional DoD Item Unique Identification (IUID) requirements as established by DOD Directive 8320.03 and MIL-STD-130.

B.2.2.3 `tms::ConfigId`

PURPOSE: Identifies the time when configuration settings are to be used.

EXTENSIBILITY: extensibility(APPENDABLE)

PATTERN: Enumeration

VALUES:

Name	Description
`CONFIG_UNKNOWN`	Unknown value. This is a generic placeholder representing the case in which an appropriate value could not be determined.
`CONFIG_DEFAULTS`	Read-only manufacturer defaults.
`CONFIG_ACTIVE`	Settings for use at the present time.

`CONFIG_ON_REBOOT`	Settings for use after a reboot / restart.
`CONFIG_ON_COMMS_LOSS`	Settings for use after a communications loss timeout.

B.2.2.4 `tms::PowerPortNumber`

PURPOSE: Identifies a port within a device. Port numbers below 0 are invalid.

EXTENSIBILITY: extensibility(APPENDABLE)

PATTERN: Typedef

ORIGINAL TYPE: `int32`

CONSTANTS:

Name	Type, Value, and Description
`MAX_PORTS`	`uint32`, 64 Maximum number of power ports that a device may contain.
`INVALID_PORT_NUMBER`	`int32`, -1 A constant value for an unspecified port number (-1).

B.2.2.5 `tms::RequestSequence`

PURPOSE: Allows Request / Reply topics to be paired together. Value is requester assigned sequence number sent in a request and returned in a reply.

EXTENSIBILITY: extensibility(FINAL)

PATTERN: Typedef

ORIGINAL TYPE: `uint64`

B.2.2.6 `tms::ReplyStatus`

PURPOSE: Indicate the type of success or failure of a request.

TOPIC USAGE: Nested

EXTENSIBILITY: extensibility(APPENDABLE)

PATTERN: Structure

ATTRIBUTES:

Name	Type and Description
`code`	`ReplyCode` Indicate whether the request was accepted or rejected.
`reason`	`String32` Short textual description of the reply code intended for human operators.

DESCRIPTION:

Based on the "Status Code and Reason Phrase" defined in RFC2616 and updated as the "Status Line" in RFC 7230.

B.2.2.7 `tms::ReplyCode`

PURPOSE: Indicator of success or failure. Intended to support automatic handling.

EXTENSIBILITY: extensibility(APPENDABLE)

PATTERN: Enumeration

VALUES:

Name	Description
`REPLY_UNKNOWN`	Unknown value. This is a generic placeholder representing the case in which an appropriate value could not be determined.
`REPLY_OK`	The request has succeeded.
`REPLY_BAD_REQUEST`	Some value in the request was invalid.
`REPLY_METHOD_NOT_ALLOWED`	A request method is not supported for the requested resource.
`REPLY_CONFLICT`	Indicates that the request could not be processed because of conflict in the current state of the resource.

REPLY_GONE

Indicates that the resource requested is no longer available and will not be available again. This should be used when a resource has been intentionally removed and the resource should be purged.

REPLY_PRECONDITION_FAILED

The request does not meet one of the preconditions.

REPLY_INTERNAL_SERVER_ERROR

A generic error message, given when an unexpected condition was encountered and no more specific message is suitable.

REPLY_NOT_IMPLEMENTED

The request either does not recognize the request method, or it lacks the ability to fulfil the request.

REPLY_SERVICE_UNAVAILABLE

The device cannot handle the request (because it is overloaded or down for maintenance). Generally, this is a temporary state

REPLY_PENDING_AUTHORIZATION

Request is valid, and authorization is required before processing.

REPLY_NOT_MASTER

The requestingDeviceId is not the selected master MC.

REPLY_PERMISSION_DENIED

Remote control is disabled for this request.

Change History:

1. *Version:* 2.0.1, *Change:* add, *Interoperability:* No-xtypes
   Description: Allow device to provide more specified reason a request is denied.

B.2.2.8 `tms::String32`

PURPOSE: A variable length string.

EXTENSIBILITY: extensibility(FINAL)

PATTERN: Typedef

ORIGINAL TYPE: `string<0,32>`

B.3 Identity

As discussed in Section A.5, every TMS device has a unique identity which is bound to a single specific device role that has a defined set of requirements and permissions.

A strong, cryptographically protected identity with bound interface information is required to establish trustworthy communications between devices. Strong identities may also be used by operators to authorize specific devices and by other systems to enforce access controls.

Trust is established when devices provide evidence that their identity belongs to a trustworthy device that has been approved by an authorized operator. Authorized operators may use privileged credentials and physical access to manage trust between devices.

Operator identities may be used to identify manual actions or to control network access to TMS devices. Operators may be granted access to specific devices or groups of devices on the grid. The operator's ability to interact with TMS devices is constrained by the assigned access. An *authorized operator* is granted access to control all normal operating modes of the device(s). Additional operator roles may include an *Administrator*, granted elevated device privileges to control access rights and operating bounds of other users, and *Technician*, granted limited device access to enable diagnostics and other functionality in support of device maintenance and repair.

Note: This section relies on lower-level protocol standards to provide secure communications. See Chapter C for details.

There are no data flows associated with this functionality.

B.3.1 Data Types

A single data type provides this functionality.

B.3.1.1 `tms::Identity`

See definition in Section B.2.2.2.

B.4 Heartbeat

Heartbeats periodically assert that a device is present and is available for TMS communications.

B.4.1 Data Flows

Each device publishes a new Heartbeat once per second. Control devices use Heartbeats to monitor the status of power devices. Power devices use MC Heartbeats according to Section A.7.5.

Figure B.12 illustrates all the topics in the data flow.

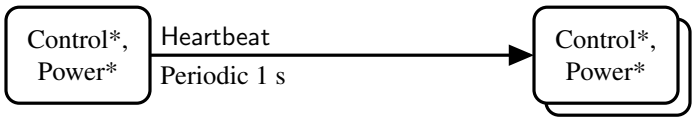


Figure B.12: Heartbeat data flow

Table B.17 provides an overview of how each topic is used.

Table B.17: Description of the Heartbeat topics.

Topic	Description
Heartbeat	Liveliness heartbeat for all devices.

Table B.18 specifies the publishers and subscribers for each topic as well as usage requirements as described in Section A.8.7.

Table B.18: Participants of the Heartbeat topics.

Topic	Usage	MD	MC	MON	SRC	STOR	DIST	LOAD	CONV
Heartbeat	Required	Pub Sub	Pub Sub	Sub	Pub Sub	Pub Sub	Pub Sub	Pub Sub	Pub Sub

This participants table has the following interpretation.

1. Heartbeat shall be published by MD, MC, SRC, STOR, DIST, LOAD, and CONV devices.

2. Heartbeat may be subscribed by MD, MC, MON, SRC, STOR, DIST, LOAD, and CONV devices.

Table B.19 specifies the timing for when sample updates are sent to each topic. These parameters are defined in Section A.8.5.

Table B.19: Timing of the Heartbeat topics.

Topic	Data Trigger	Rate Class	Burst Size	QoS Profile
Heartbeat	Periodic 1 s	1 s	1	Medium

B.4.2 Data Types

Table B.20 specifies the data types that are used as sample values for each topic.

Table B.20: Data types of the Heartbeat topics.

Topic	Data Type
Heartbeat	`tms::Heartbeat`

Topic-level data types are defined in the order shown in Table B.20. Then nested data types are defined in the order of appearance. Cross-references are provided for data types that have already been defined.

B.4.2.1 `tms::Heartbeat`

PURPOSE: Periodic indication of device availability.

TOPIC USAGE: Heartbeat

EXTENSIBILITY: extensibility(APPENDABLE)

PATTERN: Structure

ATTRIBUTES:

Name	Type and Description
`deviceId`	`Identity` The device described by this structure. Annotations: *keyval*
`sequenceNumber`	`uint32` A counter that starts at 0 and increments by 1 for each new heartbeat.

DESCRIPTION:

Network communications generally provides a low-level indication whether two devices are connected. The periodic update of `Heartbeat` with an incrementing `sequenceNumber` confirms availability of the application software.

B.4.2.2 `tms::Identity`

See definition in Section B.2.2.2.

B.5 Device Announcement

Device Announcement is the first data flow to start when communications are established, and all other data flows depend on it to function properly. This data flow associates the

identity of a device with its type, ratings, and interfaces. This "digital spec sheet" enables the rapid, ad-hoc deployment of trusted TMS devices. The data flow starts when the device is powered on, connected to the network, and authorized by the operator.

B.5.1 Data Flows

The device announcement message acts as a digital faceplate, providing information about the device to allow the microgrid controller to interact with it. The message is sent during startup, after trusted communication with the TMS is established, and describes the device's identity, type, ratings, and interfaces available. Essentially, it is an announcement that the device is available to the network. For the human operator, the device announcement also confirms that the digital faceplate matches the physical device in the microgrid. Device Announcements also remain available for other devices that connect at a later time.

Figure B.13 illustrates all the topics in the data flow.

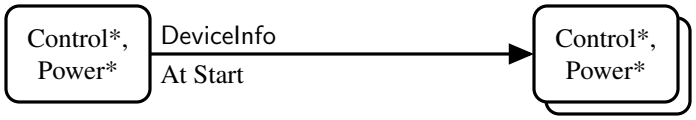


Figure B.13: Device Announcement data flow

When each device starts, it publishes a DeviceInfo so it can be discovered by neighboring devices. This registers it with the MC and causes it to appear on the MD. Power devices use this to find MCs to follow.

For information on the additional messages sent during startup that are not covered in this section, refer to the Device Power Port List and Device Grounding data types.

Table B.22 provides an overview of how each topic is used.

Table B.22: Description of the Device Announcement topics.

Topic	Description
DeviceInfo	Product information for the device. Used to discover TMS compliant devices.

Table B.23 specifies the publishers and subscribers for each topic as well as usage requirements as described in Section A.8.7.

Table B.23: Participants of the Device Announcement topics.

Topic	Usage	MD	MC	MON	SRC	STOR	DIST	LOAD	CONV
DeviceInfo	Required	Pub Sub	Pub Sub	Sub	Pub Sub	Pub Sub	Pub Sub	Pub Sub	Pub Sub

This participants table has the following interpretation.

1. DeviceInfo shall be published by MD, MC, SRC, STOR, DIST, LOAD, and CONV devices.

2. DeviceInfo may be subscribed by MD, MC, MON, SRC, STOR, DIST, LOAD, and CONV devices.

Table B.24 specifies the timing for when sample updates are sent to each topic. These parameters are defined in Section A.8.5.

Table B.24: Timing of the Device Announcement topics.

Topic	Data Trigger	Rate Class	Burst Size	QoS Profile
DeviceInfo	At Start	Once	1	PublishLast

B.5.2 Data Types

Table B.25 specifies the data types that are used as sample values for each topic.

Table B.25: Data types of the Device Announcement topics.

Topic	Data Type
DeviceInfo	`tms::DeviceInfo`

Topic-level data types are defined in the order shown in Table B.25. Then nested data types are defined in the order of appearance. Cross-references are provided for data types that have already been defined.

B.5.2.1 `tms::DeviceInfo`

PURPOSE: Design information for the device.

TOPIC USAGE: DeviceInfo

EXTENSIBILITY: extensibility(APPENDABLE)

PATTERN: Structure

ATTRIBUTES:

Name	Type and Description
`deviceId`	`Identity` The device described by this structure. Annotations: *keyval*
`role`	`DeviceRole` TMS role implemented by this device.
`product`	`ProductInfo` Product information.
`topics`	`TopicInfo` communications interfaces implemented by this device
`controlHardware`	`optional<ControlHardwareInfo>` Information about the device controller. May apply to both control services and power devices.
`powerHardware`	`optional<PowerHardwareInfo>` Information about the power hardware, if applicable.
`controlParameters`	`optional<ParameterMetadataSequence>` A complete list of read-write control parameters that must be present if the device implements the ControlParameterState topic.
`metricParameters`	`optional<ParameterMetadataSequence>` A complete list of read-only metric parameters that must be present if the device implements the MetricParameterState topic.
`controlService`	`optional<ControlServiceInfo>` Control role-specific information. Required for MC or MD device roles and empty otherwise.

`powerDevice`
: `optional<PowerDeviceInfo>`
Power device role-specific information. Required for CONV, STOR, DIST, SRC, or LOAD device roles and empty otherwise.

B.5.2.2 `tms::Identity`

See definition in Section B.2.2.2.

B.5.2.3 `tms::DeviceRole`

PURPOSE: Identify the role filled by a device.

EXTENSIBILITY: extensibility(APPENDABLE)

PATTERN: Enumeration

VALUES:

Name	Description
`ROLE_UNKNOWN`	Unknown value. This is a generic placeholder representing the case in which an appropriate value could not be determined. Never transmitted in this version of the standard.
`ROLE_MICROGRID_CONTROLLER`	Microgrid controller, a control service.
`ROLE_SOURCE`	Source power device.
`ROLE_LOAD`	Load power device.
`ROLE_STORAGE`	Storage power device.
`ROLE_DISTRIBUTION`	Distribution power device.
`ROLE_MICROGRID_DASHBOARD`	Microgrid dashboard, a control service.
`ROLE_CONVERSION`	Conversion power device.
`ROLE_MONITOR`	Monitor microgrid, read-only access.

B.5.2.4 `tms::ProductInfo`

PURPOSE: Information on how a product is marketed.

TOPIC USAGE: Nested

EXTENSIBILITY: extensibility(APPENDABLE)

PATTERN: Structure

ATTRIBUTES:

Name	Type and Description
`nsn`	`NatoStockNumber` Supply type of the product.
`gtin`	`GlobalTradeItemNumber` Commercial type of the product.
`manufacturerName`	`String32` Name of the product manufacturer.
`modelName`	`String32` Name of the product model.
`modelNumber`	`String32` Manufacturer number for the product model. May include non-numeric characters.
`serialNumber`	`String32` Unique production number for a specific product. Large enough to contain a variety of formats, including the GS1 serial number (AI 21).
`softwareVersion`	`String32` Software Version.
`platformId`	`optional<Identity>` A platform groups TMS products into a larger physical or logical package. This grouping may be shown on the microgrid dashboard, and it may supplement topology information in the microgrid controller. If a product is re-assigned to a different platform, it must resend the DeviceInfo as if it restarted. Present when such a grouping is known to the product and absent otherwise.
`platformRepresentative`	`optional<boolean>` When true, indicates this device acts as the platform representative responsible for publishing platform-specific information such as nickname or device icon. Only one device within a platform may set this to true. Change History:

1. *Version:* 2.0.3, *Change:* add, *Interoperability:* Yes
   Description: Added boolean to indicate the platform representative.

DESCRIPTION:

This information is intended to aid in logistics, operator interaction, maintenance, logging, and other tasks. Some of this information should be presented to the operator, so they can confirm the microgrid configuration. Some of this information should be used to automatically validate the values in other messages.

B.5.2.5 `tms::NatoStockNumber`

PURPOSE: Encodes a NATO Stock Number (NSN), aka National Stock Number.

EXTENSIBILITY: extensibility(APPENDABLE)

PATTERN: Typedef

ORIGINAL TYPE: `char[13]`

DESCRIPTION:

A NATO Stock Number (NSN) identifies the supply type (form, fit, and function) of a device. NSN codes are managed according to the NATO Standardization Agreements. The NSN is stored without dashes or other delimiters and is encoded as characters. The NSN value 0000-00-000-0000 is reserved as invalid/not available.

Note: These numbers are designed for human interpretation so do not follow normal algebraic conventions and in some cases, extensions may use characters instead of numbers.

B.5.2.6 `tms::GlobalTradeItemNumber`

PURPOSE: Encodes a Global Trade Item Number (GTIN).

EXTENSIBILITY: extensibility(APPENDABLE)

PATTERN: Typedef

ORIGINAL TYPE: `char[14]`

DESCRIPTION:

A Global Trade Item Number (GTIN) identifies the actual product type of a device and are managed by GS1[1].

As specified by GS1, shorter GTINs, such as the 12-character UPC code, must be prefixed by leading zero digits. If a GTIN has not been assigned, then Local Assigned Codes (LACs)

[1] http://www.gs1.org/

must be used, for example, by using a seven-zero prefix. All characters in the GTIN must be a digit, 0 to 9. See GS1 General Specifications[2] for more details.

B.5.2.7 `tms::String32`

See definition in Section B.2.2.8.

B.5.2.8 `tms::TopicInfo`

PURPOSE: Topic information. Defines which conditional and optional topics are supported by this device.

TOPIC USAGE: Nested

EXTENSIBILITY: extensibility(APPENDABLE)

PATTERN: Structure

ATTRIBUTES:

Name	Type and Description
`dataModelVersion`	`String32` Release version of data topics and types. See tms.TMS_VERSION.
`publishedConditionalTopics`	`TopicList` List of conditional topic names that are published by this device.
`publishedOptionalTopics`	`TopicList` List of optional topic names that are published by this device.
`supportedRequestTopics`	`TopicList` List of request topic names that are subscribed to by this device.
`extensionTopics`	`optional<TopicList>` List of extension topic names that are published by this device. Extension topics are not specified in the standard.

[2] http://www.gs1.org/docs/barcodes/GS1_General_Specifications.pdf

B.5.2.9 `tms::TopicList`

PURPOSE: Specifies a list of communications data topics.

EXTENSIBILITY: extensibility(APPENDABLE)

PATTERN: Typedef

ORIGINAL TYPE: `sequence<TopicName,0,64>`

B.5.2.10 `tms::TopicName`

PURPOSE: Specifies a communications data topic.

EXTENSIBILITY: extensibility(APPENDABLE)

PATTERN: Typedef

ORIGINAL TYPE: `string<1,64>`

B.5.2.11 `tms::ControlHardwareInfo`

PURPOSE: Design information for control hardware.

TOPIC USAGE: Nested

EXTENSIBILITY: extensibility(APPENDABLE)

PATTERN: Structure

ATTRIBUTES:

Name	Type and Description
`hasRealtimeClock`	`boolean` True indicates that the device has a real time clock. False otherwise.
`minTemperature`	`float32` Minimum rated operating temperature. Annotations: *units*=degree Celsius (degC) Change History: 1. *Version:* 2.0.1, *Change:* modify, *Interoperability:* No-xtypes *Description:* Removed minimum and maximum range specification.
`maxTemperature`	`float32` Maximum rated operating temperature. Annotations: *units*=degree Celsius (degC) Change History: 1. *Version:* 2.0.1, *Change:* modify, *Interoperability:* No-xtypes *Description:* Removed minimum and maximum range specification.

`numNetworkPorts`	
	`uint16`
	Number of network port connectors.
	Annotations: *min*=0

B.5.2.12 `tms::PowerHardwareInfo`

PURPOSE: Design information for power hardware. Power hardware refers to accessories or supporting components of a power device, such as an engine, control boards, etc.

TOPIC USAGE: Nested

EXTENSIBILITY: extensibility(APPENDABLE)

PATTERN: Structure

ATTRIBUTES:

Name	Type and Description
`engine`	
	`optional<EngineInfo>`
	Engine information.
`fuel`	
	`optional<FuelInfo>`
	Fuel information.
`generator`	
	`optional<GeneratorInfo>`
	Generator information.
`energyStorage`	
	`optional<EnergyStorageInfo>`
	Energy storage information.
`powerElectronics`	
	`optional<PowerElectronicsInfo>`
	Power electronics information.
`thermal`	
	`optional<ThermalInfo>`
	Thermal zone information.
`synchronousMachineModel`	
	`optional<SynchronousMachineCoefficients>`
	Dynamic stability information for synchronous machines.

B.5.2.13 `tms::EngineInfo`

PURPOSE: Design information for an engine, such as on a generator set (genset).

TOPIC USAGE: Nested

EXTENSIBILITY: extensibility(APPENDABLE)

PATTERN: Structure

ATTRIBUTES:

Name	Type and Description
`minOilPressure`	`float32` Minimum oil pressure for the operation of the engine that prevents engine damage. Annotations: *units*=pascal (Pa) Change History: 1. *Version:* 2.0.1, *Change:* modify, *Interoperability:* No-xtypes *Description:* Removed minimum range specification.
`maxOilPressure`	`float32` Maximum oil pressure for the operation of the engine that prevents engine damage. Annotations: *units*=pascal (Pa) Change History: 1. *Version:* 2.0.1, *Change:* modify, *Interoperability:* No-xtypes *Description:* Removed minimum range specification.
`minCoolantTemperature`	`float32` Minimum coolant temperature for the operation of the engine that prevents engine damage. Annotations: *units*=degree Celsius (degC)
`maxCoolantTemperature`	`float32` Maximum coolant temperature for the operation of the engine that prevents engine damage. Annotations: *units*=degree Celsius (degC)
`minEngineSpeed`	`float32` Stall warning speed of the engine. Speeds below this threshold may cause a malfunction or protective shutdown. Annotations: *units*=radian per second (rad/s) Change History: 1. *Version:* 2.0.1, *Change:* modify, *Interoperability:* No-xtypes *Description:* Removed minimum range specification.

`maxEngineSpeed`

`float32`

Maximum engine speed. I.e. the red line

Annotations: *units*=radian per second (rad/s)

Change History:

1. *Version:* 2.0.1, *Change:* modify, *Interoperability:* No-xtypes *Description:* Removed minimum range specification.

`minWetStackPreventionLoad`

`float32`

Minimum load required to prevent wet stack build up.

Annotations: *units*=watt (W)

Change History:

1. *Version:* 2.0.1, *Change:* modify, *Interoperability:* No-xtypes *Description:* Removed minimum range specification.

`minWetStackMitigationLoad`

`float32`

Minimum load required to clear wet stack build up.

Annotations: *units*=watt (W)

Change History:

1. *Version:* 2.0.1, *Change:* modify, *Interoperability:* No-xtypes *Description:* Removed minimum range specification.

B.5.2.14 `tms::FuelInfo`

PURPOSE: Design information for the fuel associated with a power device.

TOPIC USAGE: Nested

EXTENSIBILITY: extensibility(APPENDABLE)

PATTERN: Structure

ATTRIBUTES:

Name	Type and Description
`maxFuelLevel`	`float32` The rated fuel capacity of the device. Annotations: *units*=liter (L) Change History: 1. *Version:* 2.0.1, *Change:* modify, *Interoperability:* No-xtypes *Description:* Removed minimum range specification.
`lowFuelLevelCutoff`	`float32` Level where engine automatically shuts down. Annotations: *min*=0, *max*=1, *units*=per unit (p.u.)

B.5.2.15 `tms::GeneratorInfo`

PURPOSE: Design information for a power generator.

TOPIC USAGE: Nested

EXTENSIBILITY: extensibility(APPENDABLE)

PATTERN: Structure

ATTRIBUTES:

Name	Type and Description
`minFieldCurrent`	`float32` Minimum field current value for normal operation. Annotations: *units*=ampere (A)
`maxFieldCurrent`	`float32` Maximum field current value for normal operation. Annotations: *units*=ampere (A)
`maxStatorTemperature`	`float32` Maximum stator temperature. Indicates overheating.

Annotations: *units*=degree Celsius (degC)
Change History:

1. *Version:* 2.0.1, *Change:* modify, *Interoperability:* No-xtypes
   Description: Removed minimum and maximum range specification.

B.5.2.16 `tms::EnergyStorageInfo`

PURPOSE: Design information for an energy storage device or unit.

TOPIC USAGE: Nested

EXTENSIBILITY: extensibility(APPENDABLE)

PATTERN: Structure

ATTRIBUTES:

Name	Type and Description

`highStateOfCharge`

`optional<float32>`
The high state of charge during continuous operations, only to be exceeded for limited-duration operation. Increased wear when charging above this point
Annotations: *min*=0, *max*=1, *units*=per unit (p.u.)
Change History:

1. *Version:* 2.0.1, *Change:* modify, *Interoperability:* No-xtypes
   Description: Made the field optional.

`lowStateOfCharge`

`optional<float32>`
The low state of charge during continuous operations, only to go below for limited-duration operation. Increased wear when discharging below this point.
Annotations: *min*=0, *max*=1, *units*=per unit (p.u.)
Change History:

1. *Version:* 2.0.1, *Change:* modify, *Interoperability:* No-xtypes
   Description: Made the field optional.

`minTemperature`

`float32`
Minimum rated operating temperature.
Annotations: *units*=degree Celsius (degC)
Change History:

1. *Version:* 2.0.1, *Change:* modify, *Interoperability:* No-xtypes
   Description: Removed minimum and maximum range specification.

`nomTemperature`

`float32`
Most efficient operating temperature.

Annotations: *units*=degree Celsius (degC)
Change History:

1. *Version:* 2.0.1, *Change:* modify, *Interoperability:* No-xtypes
   Description: Removed minimum and maximum range specification.

`maxTemperature`
`float32`
Maximum rated operating temperature.
Annotations: *units*=degree Celsius (degC)
Change History:

1. *Version:* 2.0.1, *Change:* modify, *Interoperability:* No-xtypes
   Description: Removed minimum and maximum range specification.

B.5.2.17 `tms::PowerElectronicsInfo`

PURPOSE: Design information for power electronics.

TOPIC USAGE: Nested

EXTENSIBILITY: extensibility(APPENDABLE)

PATTERN: Structure

ATTRIBUTES:

Name	Type and Description

`minTemperature`
`float32`
Minimum rated operating temperature.
Annotations: *units*=degree Celsius (degC)
Change History:

1. *Version:* 2.0.1, *Change:* modify, *Interoperability:* No-xtypes
   Description: Removed minimum and maximum range specification.

`maxTemperature`
`float32`
Maximum rated operating temperature.
Annotations: *units*=degree Celsius (degC)
Change History:

1. *Version:* 2.0.1, *Change:* modify, *Interoperability:* No-xtypes
   Description: Removed minimum and maximum range specification.

B.5.2.18 `tms::ThermalInfo`

PURPOSE: Design information for thermal management.

TOPIC USAGE: Nested

EXTENSIBILITY: extensibility(APPENDABLE)

PATTERN: Structure

ATTRIBUTES:

Name	Type and Description
`thermalZone`	`ThermalZoneSequence` Name of each thermal zone.

B.5.2.19 `tms::ThermalZoneSequence`

PURPOSE: A sequence of String1_32.

EXTENSIBILITY: extensibility(FINAL)

PATTERN: Typedef

ORIGINAL TYPE: `sequence<String1_32,0,5>`

B.5.2.20 `tms::String1-32`

PURPOSE: A variable length string with a fixed minimum.

EXTENSIBILITY: extensibility(FINAL)

PATTERN: Typedef

ORIGINAL TYPE: `string<1,32>`

B.5.2.21 `tms::SynchronousMachineCoefficients`

PURPOSE: Model a synchronous generator, motor, or similar device for simulation and control. Direct-axis and quadrature-axis (DQ) synchronous machine model based on IEEE 115.

TOPIC USAGE: Nested

EXTENSIBILITY: extensibility(APPENDABLE)

PATTERN: Structure

ATTRIBUTES:

Name	Type and Description
`statorResistancePerPhase`	`float32` Ra, stator resistance per phase Annotations: *units*=ohm (ohm)
`statorLeakageReactance`	`float32` Xl, stator leakage reactance Annotations: *units*=ohm (ohm)
`zeroSequenceReactance`	`float32` X0, zero-sequence reactance Annotations: *units*=ohm (ohm)
`negativeSequenceReactance`	`float32` X2, negative-sequence reactance Annotations: *units*=ohm (ohm)
`zeroSequenceResistance`	`float32` R0, zero-sequence resistance Annotations: *units*=ohm (ohm)
`negativeSequenceResistance`	`float32` R2 negative-sequence resistance Annotations: *units*=ohm (ohm)
`directAxisSynchronousReactance`	`float32` Xd, direct-axis synchronous reactance Annotations: *units*=ohm (ohm)
`directAxisTransientReactance`	`float32` Xd', direct-axis transient reactance Annotations: *units*=ohm (ohm)

`directAxisSubtransientReactance`
: `float32`
Xd", direct-axis subtransient reactance
Annotations: *units*=ohm (ohm)

`quadratureAxisSynchronousReactance`
: `float32`
Xq, quadrature-axis synchronous reactance
Annotations: *units*=ohm (ohm)

`quadratureAxisTransientReactance`
: `float32`
Xq', quadrature-axis transient reactance
Annotations: *units*=ohm (ohm)

`quadratureAxisSubtransientReactance`
: `float32`
Xq", quadrature-axis subtransient reactance
Annotations: *units*=ohm (ohm)

`directAxisTransientShortCircuitTimeConstant`
: `float32`
Td', direct-axis transient short-circuit time constant
Annotations: *units*=second (s)

`directAxisSubtransientShortCircuitTimeConstant`
: `float32`
Td", direct-axis subtransient short-circuit time constant
Annotations: *units*=second (s)

`directAxisTransientOpenCircuitTimeConstant`
: `float32`
TdO', direct-axis transient open-circuit time constant
Annotations: *units*=second (s)

`directAxisSubtransientOpenCircuitTimeConstant`
: `float32`
TdO", direct-axis subtransient open-circuit time constant
Annotations: *units*=second (s)

`quadratureAxisTransientShortCircuitTimeConstant`
: `float32`
Tq', quadrature-axis transient short-circuit time constant
Annotations: *units*=second (s)

`quadratureAxisSubtransientShortCircuitTimeConstant`
: `float32`
Tq", quadrature-axis subtransient short-circuit time constant
Annotations: *units*=second (s)

`quadratureAxisTransientOpenCircuitTimeConstant`
: `float32`
TqO', quadrature-axis transient open-circuit time constant
Annotations: *units*=second (s)

`quadratureAxisSubtransientOpenCircuitTimeConstant`
`float32`
TqO", quadrature-axis subtransient open-circuit time constant
Annotations: *units*=second (s)

B.5.2.22 `tms::ParameterMetadataSequence`

PURPOSE: A sequence of ParameterMetadata.

EXTENSIBILITY: extensibility(FINAL)

PATTERN: Typedef

ORIGINAL TYPE: `sequence<ParameterMetadata,0,128>`

B.5.2.23 `tms::ParameterMetadata`

PURPOSE: Describe the allowed values for a parameter.

TOPIC USAGE: Nested

EXTENSIBILITY: extensibility(APPENDABLE)

PATTERN: Structure

ATTRIBUTES:

Name	Type and Description
`name`	`String1_32` Unique name of this parameter.
`units`	`String16` Units of measure such as V, A, Hz, or V/Hz.
`nominalMinValue`	`float32` Recommended lower bound. Set to 0 if defining an enumeration, else set to -infinity if no limit, otherwise hardMinValue ≤ nominalMinValue ≤ nominalMaxValue ≤ hardMaxValue.
`nominalMaxValue`	`float32` Recommended upper bound. Set to number of enumerations - 1 if defining an enumeration, else set to +infinity if no limit, otherwise hardMinValue ≤ nominalMinValue ≤ nominalMaxValue ≤ hardMaxValue.
`hardMinValue`	`float32` Absolute lower bound. Lower values will be rejected. Set to -infinity if no limit, otherwise hardMinValue ≤ nominalMinValue ≤ nominalMaxValue ≤ hardMaxValue.
`hardMaxValue`	`float32` Absolute upper bound. Higher values will be rejected. Set to +infinity if no limit, otherwise hardMinValue ≤ nominalMinValue ≤ nominalMaxValue ≤ hardMaxValue.
`resolution`	`float32` Smallest change ≥ 0 that may have an effect. Set to 1 if defining an enumeration, otherwise set to 0 if no limit. Annotations: *min*=0

`enumLabels`
: `EnumLabelSequence`
Enumeration labels. The sequence length must equal the number of enumerated values, otherwise 0 if not defining an enumeration. Each value must be at least 1 character in length. No duplicates are allowed per ParameterMetadata.

CONSTANTS:

Name	Type, Value, and Description
`MAX_PARAMETERS`	`uint32`, 128 Maximum number of parameters that a device may contain.

DESCRIPTION:

Device parameters may encode most primitive data types (Section A.9.6) using the values shown in Table B.41. The `units` value should be a name from Table A.8 or a name from IEEE Std 945 for units not used in TMS.

Table B.41: ParameterMetadata values for primitive data types.

Primitive Type	`hardMinValue`	`hardMaxValue`	`resolution`	`enumLabels`
Boolean	0	1	1	“false”, “true”
Signed integer	-16,777,216 (-2^{24})	16,777,216 (2^{24})	1	empty
Floating point	-infinity	+infinity	0	empty
Enumeration	0	count−1	1	“label 0”, “label 1”, ...

B.5.2.24 `tms::String16`

PURPOSE: A variable length string.

EXTENSIBILITY: extensibility(FINAL)

PATTERN: Typedef

ORIGINAL TYPE: `string<0,16>`

B.5.2.25 `tms::EnumLabelSequence`

PURPOSE: A sequence of String1_32 values that contain unique labels for an enumerated parameter.

EXTENSIBILITY: extensibility(FINAL)

PATTERN: Typedef

ORIGINAL TYPE: `sequence<String1_32,0,128>`

B.5.2.26 `tms::ControlServiceInfo`

PURPOSE: Design information for control services on microgrid controllers and dashboards.

TOPIC USAGE: Nested

EXTENSIBILITY: extensibility(APPENDABLE)

PATTERN: Structure

ATTRIBUTES:

Name	Type and Description
`mc`	`optional<MicrogridControllerInfo>` Microgrid Controller information.
`md`	`optional<MicrogridDashboardInfo>` Microgrid Dashboard information.

B.5.2.27 `tms::MicrogridControllerInfo`

PURPOSE: Design information for a microgrid controller.

TOPIC USAGE: Nested

EXTENSIBILITY: extensibility(APPENDABLE)

PATTERN: Structure

ATTRIBUTES:

Name	Type and Description
`features`	`MicrogridControllerFeatureSequence` A set of features this device supports.
`priorityRanking`	`uint16` The numeric priority ranking of this MC where 0 is the highest priority and increasing values are lower priority.

Change History:

1. *Version:* 2.0.1, *Change:* add, *Interoperability:* No-Business Logic
   Description: The active Microgrid Controller selection has changed to consider this field. As a result, backwards compatibility is not guaranteed as older power devices will not implement the updated logic.

B.5.2.28 `tms::MicrogridControllerFeatureSequence`

PURPOSE: A sequence of MicrogridControllerFeature.

EXTENSIBILITY: extensibility(FINAL)

PATTERN: Typedef

ORIGINAL TYPE: `sequence<MicrogridControllerFeature,1,3>`

B.5.2.29 `tms::MicrogridControllerFeature`

PURPOSE: Describes a feature on a specific microgrid controller.

EXTENSIBILITY: extensibility(APPENDABLE)

PATTERN: Enumeration

VALUES:

Name	Description
`MCF_UNKNOWN`	Unknown value. This is a generic placeholder representing the case in which an appropriate value could not be determined.
`MCF_FIXED`	special-purpose microgrid controller designed for a single platform or limited microgrid configuration.
`MCF_GENERAL`	general-purpose microgrid controller.

B.5.2.30 `tms::MicrogridDashboardInfo`

PURPOSE: Design information for a microgrid dashboard.

TOPIC USAGE: Nested

EXTENSIBILITY: extensibility(APPENDABLE)

PATTERN: Structure

ATTRIBUTES:

Name	Type and Description
`features`	`MicrogridDashboardFeatureSequence` A set of features this device supports.

B.5.2.31 `tms::MicrogridDashboardFeatureSequence`

PURPOSE: A sequence of MicrogridDashboardFeature.

EXTENSIBILITY: extensibility(FINAL)

PATTERN: Typedef

ORIGINAL TYPE: `sequence<MicrogridDashboardFeature,1,3>`

B.5.2.32 `tms::MicrogridDashboardFeature`

PURPOSE: Describes a feature on a specific microgrid dashboard.

EXTENSIBILITY: extensibility(APPENDABLE)

PATTERN: Enumeration

VALUES:

Name	Description
`MDF_UNKNOWN`	Unknown value. This is a generic placeholder representing the case in which an appropriate value could not be determined.
`MDF_DISPLAY`	Can show microgrid status.
`MDF_CONTROL`	Can control microgrid status.

B.5.2.33 `tms::PowerDeviceInfo`

PURPOSE: Design information for a power device which is any TMS device role that generates, distributes or consumes power. SRC, STOR, LOAD, DIST and CONV.

TOPIC USAGE: Nested

EXTENSIBILITY: extensibility(APPENDABLE)

PATTERN: Structure

ATTRIBUTES:

Name	Type and Description
`powerPorts`	`PowerPortInfoSequence` 1 for SRC and STOR and LOAD, 2+ for DIST and CONV
`grounds`	`GroundingInfoSequence` Grounding port information.
`conversion`	`optional<ConversionInfo>` Conversion information. Required for the CONV device role and empty otherwise.
`distribution`	`optional<DistributionInfo>` Distribution device information. Required for the DIST device role and empty otherwise.
`source`	`optional<SourceInfo>` Power source device information. Required for the SRC device role and empty otherwise.
`storage`	`optional<StorageInfo>` Power storage device information. Required for the STOR device role and empty otherwise.
`load`	`optional<LoadInfo>` Power load information. Required for the LOAD device role and empty otherwise.

B.5.2.34 `tms::PowerPortInfoSequence`

PURPOSE: A sequence of PowerPortInfo.

EXTENSIBILITY: extensibility(FINAL)

PATTERN: Typedef

ORIGINAL TYPE: `sequence<PowerPortInfo,0,MAX_PORTS>`

B.5.2.35 `tms::PowerPortInfo`

PURPOSE: Design information for the connector, rated circuit, and switching capabilities of a power port.

TOPIC USAGE: Nested

EXTENSIBILITY: extensibility(APPENDABLE)

PATTERN: Structure

ATTRIBUTES:

Name	Type and Description
`portNumber`	`PowerPortNumber` Number shown on the device exterior to represent this port.
`wiring`	`CircuitWiring` Type of wiring provided by this port.
`directionality`	`PowerPortDirectionality` Anticipated direction of power flow.
`hasSwitch`	`boolean` Indicate whether this power port has a switch.
`hasExternalMeter`	`boolean` Indicate whether this power port has an external meter.
`hasInternalMeter`	`boolean` Indicate whether this power port has an internal meter.
`hasSummaryMeasurementUpdate`	`boolean` Indicate whether this port supports the AC or DC SummaryMeasurementUpdate. Depends on corresponding topic.
`minAmperage`	`float32` minimum (least output) rated operating amperage.

Annotations: *units*=ampere (A)

`maxAmperage`

`float32`

maximum (greatest output) rated operating amperage.

Annotations: *units*=ampere (A)

`shortCircuitAmperage`

`float32`

Maximum rated fault amperage that will not damage the power port. Must be greater than maxAmperage. Defined in the NEC as the short-circuit current rating (SCCR).

Annotations: *units*=ampere (A)

`minVoltage`

`float32`

minimum rated operating voltage.

Annotations: *units*=volt (V)

`maxVoltage`

`float32`

maximum rated operating voltage.

Annotations: *units*=volt (V)

`minFrequency`

`optional<float32>`

minimum rated operating frequency, not present for DC, required for AC

Annotations: *units*=hertz (Hz)

Change History:

1. *Version:* 2.0.2, *Change:* modify, *Interoperability:* Yes
   Description: Corrected the units from volt to hertz.

`maxFrequency`

`optional<float32>`

maximum rated operating frequency, not present for DC, required for AC.

Annotations: *units*=hertz (Hz)

Change History:

1. *Version:* 2.0.2, *Change:* modify, *Interoperability:* Yes
   Description: Corrected the units from volt to hertz.

`connectorInfo`

`PowerConnectorInfo`

Information about the connector. Every power port must have a connector.

`switchInfo`

`optional<PowerSwitchInfo>`

Information about the switch. If hasSwitch is true, then required, otherwise not present.

DESCRIPTION:

As per the sign conventions in Section A.9.5, the amperage limits may be negative depending on the direction of power flow. This is shown in Table B.49.

Table B.49: Usual relationship between `PowerPortInfo` directionality and amperage limits.

`directionality`	`minAmperage`	`maxAmperage`
`PPD_IN`	-rated	0
`PPD_OUT`	0	+rated
`PPD_IN_OUT`	-rated	+rated

B.5.2.36 `tms::PowerPortNumber`

See definition in Section B.2.2.4.

B.5.2.37 `tms::CircuitWiring`

PURPOSE: Indicate the wiring scheme for an AC or DC circuit.

EXTENSIBILITY: extensibility(APPENDABLE)

PATTERN: Enumeration

VALUES:

Name	Description
`WIRING_UNKNOWN`	Unknown value. This is a generic placeholder representing the case in which an appropriate value could not be determined.
`WIRING_AC_SINGLE`	Single-phase AC circuit. Hot wire.
`WIRING_AC_SPLIT`	Split-phase AC circuit. Wires A, B, and N.
`WIRING_AC_3WYE`	3-phase AC circuit, wye configuration. Wires A, B, C, and N.
`WIRING_AC_3DELTA`	3-phase AC circuit, delta configuration. Wires A, B, and C.
`WIRING_DC`	2-wire DC circuit. Positive and negative.
`WIRING_DC_3WIRE`	3-wire DC circuit. Positive, negative, and neutral.

B.5.2.38 `tms::PowerPortDirectionality`

PURPOSE: Anticipated direction of power flow.

EXTENSIBILITY: extensibility(APPENDABLE)

PATTERN: Enumeration

VALUES:

Name	Description
`PPD_UNKNOWN`	not initialized.
`PPD_NONE`	directionality not applicable (e.g., for a logical port, expect zero power).
`PPD_IN`	Port intended to receive power.
`PPD_OUT`	Port intended to send power .
`PPD_IN_OUT`	Port intended to send and receive power.

B.5.2.39 `tms::PowerConnectorInfo`

PURPOSE: Design information for power connectors.

TOPIC USAGE: Nested

EXTENSIBILITY: extensibility(APPENDABLE)

PATTERN: Structure

ATTRIBUTES:

Name	Type and Description
`features`	`PowerConnectorFeatureSequence` Zero length indicates no features.
`physicalType`	`PowerConnectorType` Physical type of this connector.
`polarity`	`PowerConnectorPolarity` Polarity of this connector.

B.5.2.40 `tms::PowerConnectorFeatureSequence`

PURPOSE: A sequence of PowerConnectorFeature.

EXTENSIBILITY: extensibility(FINAL)

PATTERN: Typedef

ORIGINAL TYPE: `sequence<PowerConnectorFeature,0,6>`

B.5.2.41 `tms::PowerConnectorFeature`

PURPOSE: Indicate how a cable can be detected.

EXTENSIBILITY: extensibility(APPENDABLE)

PATTERN: Enumeration

VALUES:

Name	Description
`PCF_UNKNOWN`	Unknown value. This is a generic placeholder representing the case in which an appropriate value could not be determined.
`PCF_CABLE_SENSE`	A continuity circuit or similar feature detects when a cable is connected.
`PCF_CABLE_ID_READER`	A cable ID reader is present.
`PCF_CABLE_MEASUREMENT`	Voltage and current monitoring can establish the presence of a cable.
`PCF_COMMUNICATION`	Contains network interface.
`PCF_TOPOLOGY_DISCOVERY`	Ability to discover connectivity to remote device power ports.

B.5.2.42 `tms::PowerConnectorType`

PURPOSE: Indicate the standard defining the physical shape and form of a power connector.

EXTENSIBILITY: extensibility(APPENDABLE)

PATTERN: Enumeration

VALUES:

Name	Description
`CONNECTOR_UNKNOWN`	Unknown value. This is a generic placeholder representing the case in which an appropriate value could not be determined.
`CONNECTOR_TERMINAL_BLOCK`	Terminal block.
`CONNECTOR_MILSTD`	Generic MIL spec connector. MIL-STD-1651, MIL-DTL-22992, and MIL-DTL-53126, as used by PDISE.
`CONNECTOR_NEMA5`	For convenience power ports, etc.
`CONNECTOR_CAMLOCK`	As used by MEPDIS-R.
`CONNECTOR_POWERLOCK`	ITT VEAM PowerLock
`CONNECTOR_IEC60309`	As used by MEPDIS-R.
`CONNECTOR_J1772`	Electric vehicle charging cable.
`CONNECTOR_POWERLOK`	Amphenol PowerLok.
`CONNECTOR_MILSTD1651`	MIL-STD-1651.
`CONNECTOR_MILDTL22992`	MIL-DTL-22992.
`CONNECTOR_MILDTL53126`	MIL-DTL-53126.
`CONNECTOR_OTHER`	Other connector type, not listed elsewhere.
`CONNECTOR_METER`	Used to represent a clamp on power meter.
`CONNECTOR_BUS`	Not a physical connector. Used to represent the device internal bus. Typically used by a Distribution or Conversion device but not limited to these device roles.
`CONNECTOR_VOLTARIUS`	Amphenol Voltarius high-voltage interconnect. Change History:

1. *Version:* 2.0.3, *Change:* add, *Interoperability:* Yes
   Description: New connector requested by user.

B.5.2.43 `tms::PowerConnectorPolarity`

PURPOSE: Indicate the polarity of a power connector.

EXTENSIBILITY: extensibility(APPENDABLE)

PATTERN: Enumeration

VALUES:

Name	Description
`POLARITY_UNKNOWN`	Unknown value. This is a generic placeholder representing the case in which an appropriate value could not be determined.
`POLARITY_PIN`	This connector contains pins to insert in sockets.
`POLARITY_SOCKET`	This connector contains sockets to receive pins. Includes terminal blocks.
`POLARITY_UNIVERSAL`	This connector is symmetric.

B.5.2.44 `tms::PowerSwitchInfo`

PURPOSE: Design information for power switches.

TOPIC USAGE: Nested

EXTENSIBILITY: extensibility(APPENDABLE)

PATTERN: Structure

ATTRIBUTES:

Name	Type and Description
`features`	`PowerSwitchFeatureSequence` Zero length indicates no features.
`interruptAmperage`	`float32` Maximum (in/out) interrupt current. Annotations: *units*=ampere (A)

B.5.2.45 `tms::PowerSwitchFeatureSequence`

PURPOSE: A sequence of PowerSwitchFeature.

EXTENSIBILITY: extensibility(FINAL)

PATTERN: Typedef

ORIGINAL TYPE: `sequence<PowerSwitchFeature,0,15>`

B.5.2.46 `tms::PowerSwitchFeature`

PURPOSE: Describes a feature on a switch. The MANUAL, AUTO, and REQUEST features cover what type of actors may change the switch state.

EXTENSIBILITY: extensibility(APPENDABLE)

PATTERN: Enumeration

VALUES:

Name	Description
`PSF_UNKNOWN`	Unknown value. This is a generic placeholder representing the case in which an appropriate value could not be determined.
`PSF_MANUAL_OPEN`	The operator may open the switch using an on-device interface.
`PSF_MANUAL_CLOSE`	The operator may close the switch using an on-device interface.
`PSF_AUTO_OPEN`	The power device may automatically open the switch. For example, load shed logic.
`PSF_AUTO_CLOSE`	The power device may automatically close the switch. For example, automatic backup power logic.
`PSF_REQUEST_OPEN`	A request such as PowerSwitchRequest or EnergyStartStopRequest may open the switch. For example, the switch may be a contactor, relay, motorized switch, or a circuit breaker with a shunt trip.
`PSF_REQUEST_CLOSE`	A request such as PowerSwitchRequest or EnergyStartStopRequest may close the switch. For example, the switch may be a contactor, relay, or motorized switch.
`PSF_LOCK`	The operator may lock the switch to prevent AUTO or REQUEST operation. This lock may be per-device or per-port. For example, IEEE C37.2 device function 1, 5, 69, or 86.
`PSF_RECLOSER`	Automatic recloser. For example, IEEE C37.2 device function 79. Depends on PSF_AUTO_CLOSE.

`PSF_BREAKER`
: Automatic over current / thermal protection. For example, IEEE C37.2 device function 50 or 51. Depends on PSF_AUTO_OPEN.

`PSF_GFI`
: Automatic ground fault protection. Depends on PSF_AUTO_OPEN.

`PSF_ARC_FLASH`
: Automatic arc flash protection. For example, IEEE C37.2 device function AFD. Depends on PSF_AUTO_OPEN.

`PSF_SYNCHRONIZER`
: Synchronize before close. For example, IEEE C37.2 device function 25. Depends on PSF_AUTO_CLOSE or PSF_REQUEST_CLOSE.

`PSF_SURGE`
: Automatic over voltage protection. Depends on PSF_AUTO_OPEN.

`PSF_SWITCH_CONDITIONS`
: Supports PowerSwitchRequest.switchConditions. Depends on PSF_REQUEST_CLOSE and PowerPortInfo.hasExternalMeter.

B.5.2.47 `tms::GroundingInfoSequence`

PURPOSE: A sequence of GroundingInfo.

EXTENSIBILITY: extensibility(FINAL)

PATTERN: Typedef

ORIGINAL TYPE: `sequence<GroundingInfo,0,MAX_GROUNDING_CIRCUITS>`

B.5.2.48 `tms::GroundingInfo`

PURPOSE: Design information describing a grounding circuit in a device.

TOPIC USAGE: Nested

EXTENSIBILITY: extensibility(APPENDABLE)

PATTERN: Structure

ATTRIBUTES:

Name	Type and Description
`groundNumber`	`GroundingCircuitNumber` Number used to represent this circuit.
`groundType`	`GroundingDesignType` Type of grounding provided by this circuit.
`protectedPorts`	`PowerPortNumberSequence` List of ports protected by this circuit.
`controlSwitchFeatures`	`PowerSwitchFeatureSequence` Not present if no control switch. Required if control switch exists. Length 0 indicates no features.
`pulseSwitchFeatures`	`PowerSwitchFeatureSequence` Not present if no pulse switch. Required if pulse switch exists. Length 0 indicates no features.

B.5.2.49 `tms::GroundingCircuitNumber`

PURPOSE: Identifies a grounding circuit within a device.

EXTENSIBILITY: extensibility(APPENDABLE)

PATTERN: Typedef

ORIGINAL TYPE: `int32`

CONSTANTS:

Name	Type, Value, and Description
`MAX_GROUNDING_CIRCUITS`	`int32`, 8 Maximum number of grounding circuits that a device may contain.
`INVALID_GROUNDING_CIRCUIT_NUMBER`	`int32`, -1 A constant value for an invalid grounding circuit (-1).

B.5.2.50 `tms::GroundingDesignType`

PURPOSE: Describe the design type of a grounding circuit.

EXTENSIBILITY: extensibility(APPENDABLE)

PATTERN: Enumeration

VALUES:

Name	Description
`GROUNDING_UNKNOWN`	Unknown value. This is a generic placeholder representing the case in which an appropriate value could not be determined.
`GROUNDING_UNGROUNDED`	No grounding.
`GROUNDING_SOLID`	Solidly grounded.
`GROUNDING_HIGH_RESISTANCE`	High-resistance grounded system. Generally designed for 1-10 A ground fault current.
`GROUNDING_LOW_RESISTANCE`	Low-resistance grounded system. Generally designed to limit equipment damage.
`GROUNDING_REACTANCE`	Reactance grounded system. Generally limits ground fault current to rated line current.

B.5.2.51 `tms::PowerPortNumberSequence`

PURPOSE: A sequence of PowerPortNumber.

EXTENSIBILITY: extensibility(FINAL)

PATTERN: Typedef

ORIGINAL TYPE: `sequence<PowerPortNumber,0,MAX_PORTS>`

B.5.2.52 `tms::ConversionInfo`

PURPOSE: Design information for power conversion devices. Each power port must be listed once in either activeConversion or passiveConversion but not both.

TOPIC USAGE: Nested

EXTENSIBILITY: extensibility(APPENDABLE)

PATTERN: Structure

ATTRIBUTES:

Name	Type and Description
`features`	`ConversionFeatureSequence` A set of features this device supports.
`activeConversion`	`optional<ActiveConversionInfo>` Power regulation for active devices.
`passiveConversion`	`optional<PassiveConversionInfo>` Power regulation for passive devices.

B.5.2.53 `tms::ConversionFeatureSequence`

PURPOSE: A sequence of ConversionFeature.

EXTENSIBILITY: extensibility(FINAL)

PATTERN: Typedef

ORIGINAL TYPE: `sequence<ConversionFeature,0,3>`

B.5.2.54 `tms::ConversionFeature`

PURPOSE: Describes a feature on a specific energy converter.

EXTENSIBILITY: extensibility(APPENDABLE)

PATTERN: Enumeration

VALUES:

Name	Description
`CONVF_UNKNOWN`	Unknown value. This is a generic placeholder representing the case in which an appropriate value could not be determined.
`CONVF_ACTIVE`	Uses power electronics or other active components (programmable).
`CONVF_PASSIVE`	Uses transformers or other passive components (fixed function).

B.5.2.55 `tms::ActiveConversionInfo`

PURPOSE: Design information for active conversion load sharing devices.

TOPIC USAGE: Nested

EXTENSIBILITY: extensibility(APPENDABLE)

PATTERN: Structure

ATTRIBUTES:

Name	Type and Description
`loadSharing`	`LoadSharingInfoSequence` Load sharing capabilities of each power port.

B.5.2.56 `tms::LoadSharingInfoSequence`

PURPOSE: A sequence of LoadSharingInfo.

EXTENSIBILITY: extensibility(FINAL)

PATTERN: Typedef

ORIGINAL TYPE: `sequence<LoadSharingInfo,1,MAX_PORTS>`

B.5.2.57 `tms::LoadSharingInfo`

PURPOSE: Design information for a device capable of load sharing.

TOPIC USAGE: Nested

EXTENSIBILITY: extensibility(APPENDABLE)

PATTERN: Structure

ATTRIBUTES:

Name	Type and Description
`portNumber`	`PowerPortNumber` The power port number.
`supportsDroop`	`boolean` The device can allow the frequency (AC) and voltage (AC and DC) to sag as load increases.
`supportsMultiSegmentDroop`	`boolean` The droop curves can have multiple piecewise linear segments.
`supportsConstantPower`	`boolean` The device can regulate output power instead of frequency and voltage.
`minRealPower`	`float32` Minimum rated real power. Annotations: *units*=watt (W)
`maxRealPower`	`float32` Maximum rated real power. Annotations: *units*=watt (W)
`maxOverloadRealPower`	`optional<Curve2D>` Rated overload output real power as a function of overload duration Annotations: *units*=second, watt (s, W)

`minReactivePower`
: `float32`
Minimum rated reactive power.
Annotations: *units*=volt ampere reactive (var)

`maxReactivePower`
: `float32`
Maximum rated reactive power.
Annotations: *units*=volt ampere reactive (var)

`maxApparentPower`
: `float32`
Maximum rated apparent power (absolute value).
Annotations: *units*=volt ampere (VA)

`powerFactor`
: `float32`
Rated power factor, usually 0.8.

`loadResponse`
: `optional<LoadStepResponseSequence>`
Expected response to various load steps.

DESCRIPTION:

This information applies to Section B.21 and Section B.22. Support for load sharing features depends on the state topics being listed in `TopicInfo.conditionalTopics` and the request topics being listed in `TopicInfo.optionalTopics`.

B.5.2.58 `tms::Curve2D`

PURPOSE: Represent a curve in a two-dimensional space.

TOPIC USAGE: Nested

EXTENSIBILITY: extensibility(APPENDABLE)

PATTERN: Structure

ATTRIBUTES:

Name	Type and Description
`points`	`Point2DSequence` Vertices on the curve.

DESCRIPTION:

The Curve2D specifies a single vertical value, y, for each horizontal value, x. Coordinates on a Curve2D are written as (x, y) and the corresponding units are written as [unit x, unit y].

Curve2D defines a set of straight lines connecting between specified points. The points shall be "in order" with points[i].x < points[i+1].x. It is an error for two or more points to have the same x value. To evaluate the Curve2D extrapolation for any point along the curve, the identified points in the sequence are used.

To calculate the vertical y values:

1. If x is one of the points in the sequence, the value of y is given in the sequence.
2. If x is between two points in the sequence, then the y value is a straight-line interpolation between the two points.
3. If x is before the point in the sequence, then the y value is a straight line projected through the first two points of the sequence.
4. If x is after the last point in the sequence, then the y value is a straight line projected through the last two points of the sequence.

Points

x	y
0	1
1	3
2	2

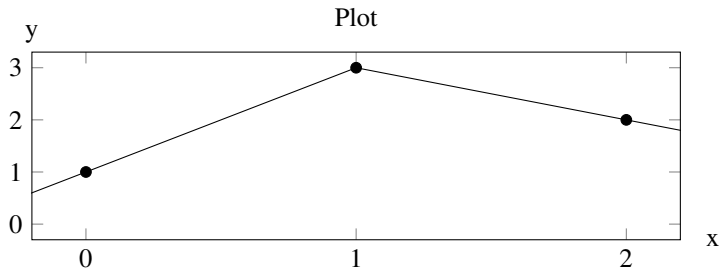


Figure B.14: Example `Curve2D` with three points.

B.5.2.59 `tms::Point2DSequence`

PURPOSE: A sequence of Point2D.

EXTENSIBILITY: extensibility(FINAL)

PATTERN: Typedef

ORIGINAL TYPE: `sequence<Point2D,0,21>`

B.5.2.60 `tms::Point2D`

PURPOSE: Represent a point in a two-dimensional space.

TOPIC USAGE: Nested

EXTENSIBILITY: extensibility(APPENDABLE)

PATTERN: Structure

ATTRIBUTES:

Name	Type and Description
`x`	`float32` Horizontal coordinate.
`y`	`float32` Vertical coordinate.

B.5.2.61 `tms::LoadStepResponseSequence`

PURPOSE: A sequence of LoadStepResponse.

EXTENSIBILITY: extensibility(FINAL)

PATTERN: Typedef

ORIGINAL TYPE: `sequence<LoadStepResponse,0,16>`

B.5.2.62 `tms::LoadStepResponse`

PURPOSE: Design information for a device capable of load sharing. Expected step response when directly connected to a load. Both acceptance and rejection responses start from steady state and stop when the response settles. The load step occurs at t=0, and the step size should be at least 50

TOPIC USAGE: Nested

EXTENSIBILITY: extensibility(APPENDABLE)

PATTERN: Structure

ATTRIBUTES:

Name	Type and Description
`loadStepLowRealPower`	`float32` Initial condition for the load acceptance step. Final condition for the load rejection step. Annotations: *units*=watt (W)
`loadStepHighRealPower`	`float32` Initial condition for the load rejection step. Final condition for the load acceptance step. Annotations: *units*=watt (W)
`loadStepLowReactivePower`	`float32` Initial condition for the load acceptance step. Final condition for the load rejection step. Annotations: *units*=volt ampere reactive (var)
`loadStepHighReactivePower`	`float32` Initial condition for the load rejection step. Final condition for the load acceptance step. Annotations: *units*=volt ampere reactive (var)
`loadAcceptanceFrequency`	`Curve2D` Plot of expected frequency response to the load acceptance step. Annotations: *units*=hertz, second (Hz, s)
`loadAcceptanceVoltage`	`Curve2D` Plot of expected voltage response to the load acceptance step. Annotations: *units*=volt, second (V, s)
`loadAcceptanceRealPower`	`Curve2D` Plot of expected real power response to the load acceptance step. Annotations: *units*=watt, second (W, s)

`loadAcceptanceReactivePower`
: `Curve2D`
Plot of expected reactive power response to the load acceptance step.
Annotations: *units*=volt ampere reactive, second (var, s)

`loadRejectionFrequency`
: `Curve2D`
Plot of expected frequency response to the load rejection step.
Annotations: *units*=hertz, second (Hz, s)

`loadRejectionVoltage`
: `Curve2D`
Plot of expected voltage response to the load rejection step.
Annotations: *units*=volt, second (V, s)

`loadRejectionRealPower`
: `Curve2D`
Plot of expected real power response to the load rejection step.
Annotations: *units*=watt, second (W, s)

`loadRejectionReactivePower`
: `Curve2D`
Plot of expected reactive power response to the load rejection step.
Annotations: *units*=volt ampere reactive, second (var, s)

B.5.2.63 `tms::PassiveConversionInfo`

PURPOSE: Design information for a passive energy converter.

TOPIC USAGE: Nested

EXTENSIBILITY: extensibility(APPENDABLE)

PATTERN: Structure

ATTRIBUTES:

Name	Type and Description
`portConversion`	`PowerPortConversionInfoSequence` Information for each port on the device.

B.5.2.64 `tms::PowerPortConversionInfoSequence`

PURPOSE: A sequence of PowerPortConversionInfo.

EXTENSIBILITY: extensibility(FINAL)

PATTERN: Typedef

ORIGINAL TYPE: `sequence<PowerPortConversionInfo,1,MAX_PORTS>`

B.5.2.65 `tms::PowerPortConversionInfo`

PURPOSE: Design information for power port conversation devices.

TOPIC USAGE: Nested

EXTENSIBILITY: extensibility(APPENDABLE)

PATTERN: Structure

ATTRIBUTES:

Name	Type and Description
`portNumber`	`PowerPortNumber` The power port number this structure represents.
`conversionTaps`	`ConversionTapInfoSequence` Sequence to support tap changers
`changeUnderLoad`	`boolean` False requires (or causes) circuit interrupt to change, true can change under rated load.

B.5.2.66 `tms::ConversionTapInfoSequence`

PURPOSE: A sequence of ConversionTapInfo.

EXTENSIBILITY: extensibility(FINAL)

PATTERN: Typedef

ORIGINAL TYPE: `sequence<ConversionTapInfo,1,MAX_TAPS>`

B.5.2.67 `tms::ConversionTapInfo`

PURPOSE: Design information for conversion taps.

TOPIC USAGE: Nested

EXTENSIBILITY: extensibility(APPENDABLE)

PATTERN: Structure

ATTRIBUTES:

Name	Type and Description
`tapId`	`TapNumber` Tap changer position.
`phaseShift`	`float32` Phase shift between this port and the internal bus.
`voltageRatio`	`float32` Voltage ratio between this port and the internal bus.

B.5.2.68 `tms::TapNumber`

PURPOSE: Tap changer index in a passive conversion device. Defaults to 0 when no changer is present.

EXTENSIBILITY: extensibility(APPENDABLE)

PATTERN: Typedef

ORIGINAL TYPE: `int32`

CONSTANTS:

Name	Type, Value, and Description
`MAX_TAPS`	`int32`, 64 Maximum number of positions in a tap changer. Many tap changers have 33 positions.

B.5.2.69 `tms::DistributionInfo`

PURPOSE: Design information for the distribution device role.

TOPIC USAGE: Nested

EXTENSIBILITY: extensibility(APPENDABLE)

PATTERN: Structure

ATTRIBUTES:

Name	Type and Description
`features`	`DistributionFeatureSequence` A set of features this device supports.

B.5.2.70 `tms::DistributionFeatureSequence`

PURPOSE: A sequence of DistributionFeature.

EXTENSIBILITY: extensibility(FINAL)

PATTERN: Typedef

ORIGINAL TYPE: `sequence<DistributionFeature,0,6>`

B.5.2.71 `tms::DistributionFeature`

PURPOSE: Describes a feature on a specific energy distribution device.

EXTENSIBILITY: extensibility(APPENDABLE)

PATTERN: Enumeration

VALUES:

Name	Description
`DISTF_UNKNOWN`	Unknown value. This is a generic placeholder representing the case in which an appropriate value could not be determined.
`DISTF_CLAMP_METER`	clamp-on power meter, two logical power ports.
`DISTF_TAP_METER`	pass-through power meter, two physical power ports.
`DISTF_PCC`	has additional capabilities for PCC interconnect with another grid.
`DISTF_FEEDER`	intended for supplying power to other DIST devices.

`DISTF_DISTRIBUTION`	intended for supplying power to LOAD devices.

B.5.2.72 `tms::SourceInfo`

PURPOSE: Design information for power sources.

TOPIC USAGE: Nested

EXTENSIBILITY: extensibility(APPENDABLE)

PATTERN: Structure

ATTRIBUTES:

Name	Type and Description
`features`	`SourceFeatureSequence` A set of features this device supports.
`loadSharing`	`LoadSharingInfo` Load sharing capabilities of this device.
`supportedEnergyStartStopLevels`	`EnergyStartStopLevelSequence` List of the EnergyStartStopLevels supported by this device for use in EnergyStartStopState or EnergyStartStopRequest.

B.5.2.73 `tms::SourceFeatureSequence`

PURPOSE: A sequence of SourceFeature.

EXTENSIBILITY: extensibility(FINAL)

PATTERN: Typedef

ORIGINAL TYPE: `sequence<SourceFeature,0,6>`

B.5.2.74 `tms::SourceFeature`

PURPOSE: Describes a feature on a specific energy source device.

EXTENSIBILITY: extensibility(APPENDABLE)

PATTERN: Enumeration

VALUES:

Name	Description
`SRCF_UNKNOWN`	Unknown value. This is a generic placeholder representing the case in which an appropriate value could not be determined.
`SRCF_GENSET`	Genset source.
`SRCF_FUEL_CELL`	Fuel cell source.
`SRCF_SOLAR`	Solar source.
`SRCF_WIND`	Wind source.
`SRCF_VEHICLE`	Source shared with a vehicle power train.

B.5.2.75 `tms::EnergyStartStopLevelSequence`

PURPOSE: A sequence of EnergyStartStopLevel, allowing each operating level to be listed once.

EXTENSIBILITY: extensibility(APPENDABLE)

PATTERN: Typedef

ORIGINAL TYPE: `sequence<EnergyStartStopLevel,0,8>`

B.5.2.76 `tms::EnergyStartStopLevel`

PURPOSE: Start/stop operating level of a SRC or STOR device.

EXTENSIBILITY: extensibility(APPENDABLE)

PATTERN: Enumeration

VALUES:

Name	Description
`ESSL_UNKNOWN`	Unknown value. This is a generic placeholder representing the case in which an appropriate value could not be determined.
`ESSL_ANY`	Wildcard value. Allows the EnergyStartStopRequest.fromLevel to match any EnergyStartStopState.presentLevel. Invalid for other uses.
`ESSL_OFF`	Device controller is operational. Energy and power component temperatures are ambient. Energy component is disabled. Power component is disabled. Synchronization is not maintained. Power port switch is open.
`ESSL_WARM`	Device controller is operational. Energy and power component temperatures are regulated. Energy component is disabled. Power component is disabled. Synchronization is not maintained. Power port switch is open.
`ESSL_IDLE`	Device controller is operational. Energy and power component temperatures are regulated. Energy component is enabled but not ready to operate at rated capacity. Power component is disabled. Synchronization is not maintained. Power port switch is open.
`ESSL_READY`	Device controller is operational. Energy and power component temperatures are regulated. Energy component is enabled and ready to operate at rated capcity. Power component is enabled. Synchronization is not maintained. Power port switch is open.
`ESSL_READY_SYNCED`	Device controller is operational. Energy and power component temperatures are regulated. Energy component is enabled and ready to operate at rated capcity. Power component is enabled. Synchronization is maintained. Power port switch is open.
`ESSL_OPERATIONAL`	Device controller is operational. Energy and power component temperatures are regulated. Energy component is enabled and ready to operate at rated capcity. Power component is enabled. Synchronization is maintained. Power port switch is closed.

B.5.2.77 `tms::StorageInfo`

PURPOSE: Design information for power storage devices.

TOPIC USAGE: Nested

EXTENSIBILITY: extensibility(APPENDABLE)

PATTERN: Structure

ATTRIBUTES:

Name	Type and Description
`features`	`StorageFeatureSequence` A set of features this device supports.
`maxChargeEnergy`	`float32` Rated energy at full state of charge. Annotations: *units*=joule (J)
`loadSharing`	`LoadSharingInfo` Load sharing capabilities of this device.
`supportedEnergyStartStopLevels`	`EnergyStartStopLevelSequence` List of the EnergyStartStopLevels supported by this device for use in EnergyStartStopState or EnergyStartStopRequest.

B.5.2.78 `tms::StorageFeatureSequence`

PURPOSE: A sequence of StorageFeature.

EXTENSIBILITY: extensibility(FINAL)

PATTERN: Typedef

ORIGINAL TYPE: `sequence<StorageFeature,0,5>`

B.5.2.79 `tms::StorageFeature`

PURPOSE: Describes a feature of a specific energy storage device.

EXTENSIBILITY: extensibility(APPENDABLE)

PATTERN: Enumeration

VALUES:

Name	Description
`STORF_UNKNOWN`	Unknown value. This is a generic placeholder representing the case in which an appropriate value could not be determined.
`STORF_GRID`	Grid attached storage, does not require pass-through from source to load.
`STORF_SUBCYCLE_UPS`	Can provide fast voltage support to recover from a grid fault condition.
`STORF_CHARGING`	Integrated into a charging station, as for electric vehicles.
`STORF_VEHICLE`	Integrated into an electric vehicle power train.

B.5.2.80 `tms::LoadInfo`

PURPOSE: Design information for load devices.

TOPIC USAGE: Nested

EXTENSIBILITY: extensibility(APPENDABLE)

PATTERN: Structure

ATTRIBUTES:

Name	Type and Description
`features`	`LoadFeatureSequence` A set of features this device supports.
`maxRealPower`	`float32` Maximum real power utilization from this Load. Annotations: *units*=watt (W)
`maxReactivePower`	`float32` Maximum reactive power utilization from this Load. Annotations: *units*=volt ampere reactive (var)

B.5.2.81 `tms::LoadFeatureSequence`

PURPOSE: A sequence of LoadFeature.

EXTENSIBILITY: extensibility(FINAL)

PATTERN: Typedef

ORIGINAL TYPE: `sequence<LoadFeature,0,4>`

B.5.2.82 `tms::LoadFeature`

PURPOSE: Describes a feature on a specific energy load.

EXTENSIBILITY: extensibility(APPENDABLE)

PATTERN: Enumeration

VALUES:

Name	Description
`LOADF_UNKNOWN`	Unknown value. This is a generic placeholder representing the case in which an appropriate value could not be determined.
`LOADF_DEMAND_RESPONSE`	Device can shed load to provide grid support.
`LOADF_CHANGE_NOTIFICATION`	Device can notify the microgrid before causing a large load transient.
`LOADF_SOFT_START`	Device can smoothly ramp up and down to minimize load transients.

B.6 Nickname

Every TMS device is identified by a machine-readable `Identity`. This `Identity` may be long with no clear meaning to the operator. `IdentityNicknameState` is a human-readable name for the `Identity` intended for display on the device and MD. The nickname may be configured on the device or MD. It should be persisted across device reboots.

A nickname may be assigned to a device or platform `Identity`. When operating within a platform, only the platform representative [see Section A.6.3] may publish a `IdentityNicknameState` associated with a platform `Identity`.

B.6.1 Data Flows

Figure B.15 illustrates all the topics in the data flow.

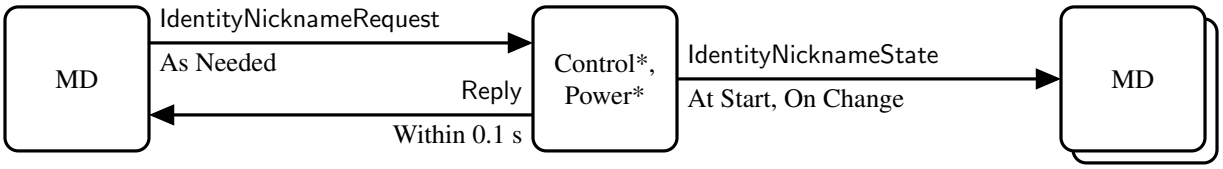


Figure B.15: Nickname data flow

The nickname is only for operator convenience. It does not impact the operation of the microgrid because all TMS device communications use the `Identity`. The nickname shall not be used for automatic control.

There are three ways that a nickname can be assigned to the device:

1. Stored in memory so it can be announced at startup.
2. Directly assigned by an operator. The device will publish an `IdentityNicknameState` after the nickname is assigned.
3. Issued a new name by the MD. If the MD's `IdentityNicknameRequest` is successful, the device nickname is updated and the `IdentityNicknameState` is published.

Table B.81 provides an overview of how each topic is used.

Table B.81: Description of the Nickname topics.

Topic	Description
IdentityNicknameState	Device and Platform Identity Nickname.
IdentityNicknameRequest	Request to change a Device or Platform Identity Nickname.
Reply	The reply to a request.

Table B.82 specifies the publishers and subscribers for each topic as well as usage requirements as described in Section A.8.7.

Table B.82: Participants of the Nickname topics.

Topic	Usage	MD	MC	MON	SRC	STOR	DIST	LOAD	CONV
IdentityNicknameState	Conditional	Pub	Pub		Pub	Pub	Pub	Pub	Pub
		Sub		Sub					
IdentityNicknameRequest	Optional	Pub							
		Sub	Sub	Sub	Sub	Sub	Sub	Sub	Sub
Reply	Required	Pub	Pub		Pub	Pub	Pub	Pub	Pub
		Sub	Sub	Sub					

This participants table has the following interpretation.

1. IdentityNicknameState is conditional on SUPPORTS_REQUEST.

2. IdentityNicknameState shall be published by MD, MC, SRC, STOR, DIST, LOAD, and CONV devices that support the corresponding request topic.
3. IdentityNicknameState may be subscribed by MD and MON devices.
4. IdentityNicknameRequest shall be published by MD devices that have this topic listed in `TopicInfo.optionalTopics`.
5. IdentityNicknameRequest may be subscribed by MD, MC, MON, SRC, STOR, DIST, LOAD, and CONV devices.
6. Reply shall be published by MD, MC, SRC, STOR, DIST, LOAD, and CONV devices.
7. Reply may be subscribed by MD, MC, and MON devices.

Table B.83 specifies the timing for when sample updates are sent to each topic. These parameters are defined in Section A.8.5.

Table B.83: Timing of the Nickname topics.

`Topic`	Data Trigger	Rate Class	Burst Size	QoS Profile
IdentityNicknameState	At Start, On Change	10 s	1	PublishLast
IdentityNicknameRequest	As Needed	10 s	1	Command
Reply	Within 0.1 s	All	1	Reply

B.6.2 Data Types

Table B.84 specifies the data types that are used as sample values for each topic.

Table B.84: Data types of the Nickname topics.

Topic	`Data Type`
IdentityNicknameState	`tms::IdentityNicknameState`
IdentityNicknameRequest	`tms::IdentityNicknameRequest`
Reply	`tms::Reply`

Topic-level data types are defined in the order shown in Table B.84. Then nested data types are defined in the order of appearance. Cross-references are provided for data types that have already been defined.

B.6.2.1 `tms::IdentityNicknameState`

PURPOSE: Provide a short, human-readable name for an Identity.

TOPIC USAGE: IdentityNicknameState

EXTENSIBILITY: extensibility(APPENDABLE)

PATTERN: Structure

ATTRIBUTES:

Name	Type and Description
`deviceId`	`Identity` Identity of the device that sent this structure. Annotations: *keyval*
`identityId`	`Identity` Identity using the nickname. It must match a DeviceInfo.deviceId or DeviceInfo.product.platformId. Annotations: *keyval*
`nickname`	`String32` Nickname given to this identityId.

B.6.2.2 `tms::IdentityNicknameRequest`

PURPOSE: Change the nickname associated with a Identity.

TOPIC USAGE: IdentityNicknameRequest

EXTENSIBILITY: extensibility(APPENDABLE)

PATTERN: Structure

ATTRIBUTES:

Name	Type and Description
`requestId`	`DeviceRequest` Identity of this request. Annotations: *keyval*
`sequenceId`	`tms::RequestSequence` Request sequence data used to associate a request and returning reply.
`nickname`	`String32` New nickname to be used for this Identity.

B.6.2.3 `tms::Reply`

See definition in Section B.2.2.1.

B.6.2.4 `tms::Identity`

See definition in Section B.2.2.2.

B.6.2.5 `tms::String32`

See definition in Section B.2.2.8.

B.6.2.6 `tms::DeviceRequest`

PURPOSE: Key value for a request that targets one device.

TOPIC USAGE: Nested

EXTENSIBILITY: extensibility(APPENDABLE)

PATTERN: Structure

ATTRIBUTES:

Name	Type and Description
`requestingDeviceId`	`Identity` Identity of the device that sent this request.
`targetDeviceId`	`Identity` Identity of the device that should receive this request.

B.6.2.7 `tms::RequestSequence`

See definition in Section B.2.2.5.

B.6.2.8 `tms::ConfigId`

See definition in Section B.2.2.3.

B.6.2.9 `tms::PowerPortNumber`

See definition in Section B.2.2.4.

B.6.2.10 `tms::ReplyStatus`

See definition in Section B.2.2.6.

B.6.2.11 `tms::ReplyCode`

See definition in Section B.2.2.7.

B.7 Device Icon

The Device Icon allows a power device to provide a device or platform image for display on the MD. Otherwise the MD may use a stock image based on the device role, NSN, or other information in the DeviceInfo. This image is tied to a device type, not a specific device or its operational state.

B.7.1 Data Flows

Figure B.16 illustrates all the topics in the data flow.

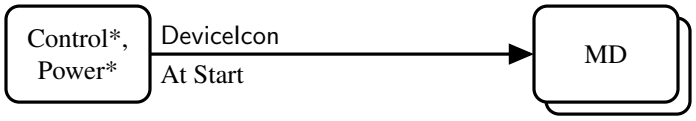


Figure B.16: Device Icon data flow

The MD may receive an image from one device of a given type and use it for all devices of the same type.

Table B.88 provides an overview of how each topic is used.

Table B.88: Description of the Device Icon topics.

Topic	Description
DeviceIcon	Device image as an icon.

Table B.89 specifies the publishers and subscribers for each topic as well as usage requirements as described in Section A.8.7.

Table B.89: Participants of the Device Icon topics.

Topic	Usage	MD	MC	MON	SRC	STOR	DIST	LOAD	CONV
DeviceIcon	Optional	Pub Sub	Pub	Sub	Pub	Pub	Pub	Pub	Pub

This participants table has the following interpretation.

1. DeviceIcon shall be published by MD, MC, SRC, STOR, DIST, LOAD, and CONV devices that have this topic listed in `TopicInfo.optionalTopics`.

2. DeviceIcon may be subscribed by MD and MON devices.

Table B.90 specifies the timing for when sample updates are sent to each topic. These parameters are defined in Section A.8.5.

Table B.90: Timing of the Device Icon topics.

Topic	Data Trigger	Rate Class	Burst Size	QoS Profile
DeviceIcon	At Start	Once	1	PublishLast

B.7.2 Data Types

Table B.91 specifies the data types that are used as sample values for each topic.

Table B.91: Data types of the Device Icon topics.

Topic	Data Type
DeviceIcon	`tms::DeviceIcon`

Topic-level data types are defined in the order shown in Table B.91. Then nested data types are defined in the order of appearance. Cross-references are provided for data types that have already been defined.

B.7.2.1 `tms::DeviceIcon`

PURPOSE: Small image representing a device or platform.

TOPIC USAGE: DeviceIcon

EXTENSIBILITY: extensibility(APPENDABLE)

PATTERN: Structure

ATTRIBUTES:

Name	Type and Description
`deviceId`	`Identity` The device or platform illustrated by this image. Only the platform representative may publish a platform icon. Annotations: *keyval* Change History: 1. *Version:* 2.0.3, *Change:* modify, *Interoperability:* Yes *Description:* Update the deviceId description to allow the platform representative to publish a platform icon.

`mimeType`

`String32`
Type of the image format. Currently, the only supported type is 'image/png'. See the IANA Media Type list - http://www.iana.org/assignments/media-types/media-types.xhtml.

`data`

`OctetSequence`
Contents of the image with a resolution of 64x64 and transparent background. The image should be clearly visible on a white or black background.

B.7.2.2 `tms::Identity`

See definition in Section B.2.2.2.

B.7.2.3 `tms::String32`

See definition in Section B.2.2.8.

B.7.2.4 `tms::OctetSequence`

PURPOSE: A sequence of Octet (byte).

EXTENSIBILITY: extensibility(FINAL)

PATTERN: Typedef

ORIGINAL TYPE: `sequence<octet,1,32768>`

B.8 Control Hardware

All TMS devices have control hardware that provides automation and the communications interface. This may be the entirety of a control device, or the device controller of a power device. This section provides information on the operating status of that control hardware.

B.8.1 Data Flows

Figure B.17 illustrates all the topics in the data flow.

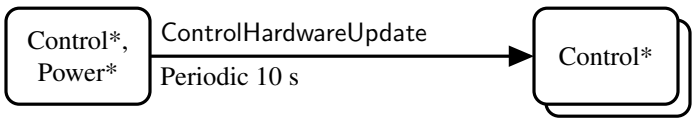


Figure B.17: Control Hardware data flow

Table B.93 provides an overview of how each topic is used.

Table B.93: Description of the Control Hardware topics.

Topic	Description
ControlHardwareUpdate	Reports dynamic state changes of control device hardware.

Table B.94 specifies the publishers and subscribers for each topic as well as usage requirements as described in Section A.8.7.

Table B.94: Participants of the Control Hardware topics.

Topic	Usage	MD	MC	MON	SRC	STOR	DIST	LOAD	CONV
ControlHardwareUpdate	Optional	Pub Sub	Pub Sub	Sub	Pub	Pub	Pub	Pub	Pub

This participants table has the following interpretation.

1. ControlHardwareUpdate shall be published by MD, MC, SRC, STOR, DIST, LOAD, and CONV devices that have this topic listed in `TopicInfo.optionalTopics`.
2. ControlHardwareUpdate may be subscribed by MD, MC, and MON devices.

Table B.95 specifies the timing for when sample updates are sent to each topic. These parameters are defined in Section A.8.5.

Table B.95: Timing of the Control Hardware topics.

`Topic`	Data Trigger	Rate Class	Burst Size	QoS Profile
ControlHardwareUpdate	Periodic 10 s	10 s	1	Slow

B.8.2 Data Types

Table B.96 specifies the data types that are used as sample values for each topic.

Table B.96: Data types of the Control Hardware topics.

Topic	`Data Type`
ControlHardwareUpdate	`tms::ControlHardwareUpdate`

Topic-level data types are defined in the order shown in Table B.96. Then nested data types are defined in the order of appearance. Cross-references are provided for data types that have already been defined.

B.8.2.1 `tms::ControlHardwareUpdate`

PURPOSE: Operating state for the control hardware in this device.

TOPIC USAGE: ControlHardwareUpdate

EXTENSIBILITY: extensibility(APPENDABLE)

PATTERN: Structure

ATTRIBUTES:

Name	Type and Description
`deviceId`	`Identity` The device described by this structure. Annotations: *keyval*
`timestamp`	`ClockMonotonic` Time of these measurements.
`processorUsage`	`optional<float32>` Central Processing Unit (CPU) usage, as a per unit measure of the whole capacity. Annotations: *min*=0, *max*=1, *units*=per unit (p.u.) Change History: 1. *Version:* 2.0.1, *Change:* modify, *Interoperability:* No-xtypes *Description:* Made the field optional.
`memoryUsage`	`optional<float32>` Memory utilization of the hardware, as a per unit measure of the whole capacity. Annotations: *min*=0, *max*=1, *units*=per unit (p.u.) Change History: 1. *Version:* 2.0.1, *Change:* modify, *Interoperability:* No-xtypes *Description:* Made the field optional.
`networkUsage`	`optional<float32>` Network utilization of the hardware, as a per unit measure of the whole capacity. If the device has multiple network interfaces, this field will describe the network interface utilized by TMS communication protocols. Annotations: *min*=0, *max*=1, *units*=per unit (p.u.) Change History: 1. *Version:* 2.0.1, *Change:* modify, *Interoperability:* No-xtypes *Description:* Made the field optional.
`storageUsage`	`optional<float32>` Disk storage utilization of the hardware, as a per unit measure of the whole capacity.

Annotations: *min*=0, *max*=1, *units*=per unit (p.u.)
Change History:

1. *Version:* 2.0.1, *Change:* modify, *Interoperability:* No-xtypes
   Description: Made the field optional.

`temperature`
`float32`
Temperature measurement of the control hardware.
Annotations: *units*=degree Celsius (degC)
Change History:

1. *Version:* 2.0.1, *Change:* modify, *Interoperability:* No-xtypes
   Description: Removed minimum and maximum range specification.

B.8.2.2 `tms::Identity`

See definition in Section B.2.2.2.

B.8.2.3 `tms::ClockMonotonic`

PURPOSE: Local oscillator timestamp, not affected by clock jumps.

TOPIC USAGE: Nested

EXTENSIBILITY: extensibility(APPENDABLE)

PATTERN: Structure

ATTRIBUTES:

Name	Type and Description
`seconds`	`uint32` integer portion of the timestamp.
`nanoseconds`	`uint32` fractional portion of the timestamp.

DESCRIPTION:

This clock is used to measure the relative time between events local to a device. The start time, t=0.0, of this clock is unspecified. This clock must be available for all messages, forward progress must track a faithful estimate of passing time, and negative jumps are prohibited.

For more information on the intended usage of this clock, see IEEE 1003.1-2012, Section 3.227 Monotonic Clock. See also IEEE 1003.13-2003, Section 6.6.1.13 Clocks and Timers.

B.9 Power Hardware

This section provides information on the operating status of the physical components in a power device. This information relates to the internal status of these components; external status and control is supported in other sections.

B.9.1 Data Flows

Figure B.18 illustrates all the topics in the data flow.

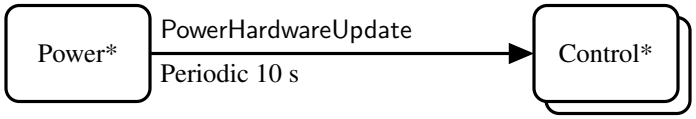


Figure B.18: Power Hardware data flow

Table B.99 provides an overview of how each topic is used.

Table B.99: Description of the Power Hardware topics.

Topic	Description
PowerHardwareUpdate	Reports dynamic state of power hardware components.

Table B.100 specifies the publishers and subscribers for each topic as well as usage requirements as described in Section A.8.7.

Table B.100: Participants of the Power Hardware topics.

Topic	Usage	MD	MC	MON	SRC	STOR	DIST	LOAD	CONV
PowerHardwareUpdate	Optional	Sub	Sub	Sub	Pub	Pub	Pub	Pub	Pub

This participants table has the following interpretation.

1. PowerHardwareUpdate shall be published by SRC, STOR, DIST, LOAD, and CONV devices that have this topic listed in `TopicInfo.optionalTopics`.
2. PowerHardwareUpdate may be subscribed by MD, MC, and MON devices.

Table B.101 specifies the timing for when sample updates are sent to each topic. These parameters are defined in Section A.8.5.

Table B.101: Timing of the Power Hardware topics.

Topic	Data Trigger	Rate Class	Burst Size	QoS Profile
PowerHardwareUpdate	Periodic 10 s	10 s	1	Slow

B.9.2 Data Types

Table B.102 specifies the data types that are used as sample values for each topic.

Table B.102: Data types of the Power Hardware topics.

Topic	Data Type
PowerHardwareUpdate	`tms::PowerHardwareUpdate`

Topic-level data types are defined in the order shown in Table B.102. Then nested data types are defined in the order of appearance. Cross-references are provided for data types that have already been defined.

B.9.2.1 `tms::PowerHardwareUpdate`

PURPOSE: Operating state of the power hardware.

TOPIC USAGE: PowerHardwareUpdate

EXTENSIBILITY: extensibility(APPENDABLE)

PATTERN: Structure

ATTRIBUTES:

Name	Type and Description
`deviceId`	`Identity` The device described by this structure. Annotations: *keyval*
`timestamp`	`ClockMonotonic` Time of these measurements.
`engine`	`optional<EngineState>` Operating state of the engine if present.
`fuel`	`optional<FuelState>` Fuel state, if present.
`generator`	`optional<GeneratorState>` Operating state of a generator, if present.
`energyStorage`	`optional<EnergyStorageState>` Operating state of energy storage if present.
`powerElectronics`	`optional<PowerElectronicsState>` Operating state of active components including inverters and rectifiers, if present.

`thermal`
: `optional<ThermalState>`
Thermal operating state including heating and cooling components, if present.

B.9.2.2 `tms::Identity`

See definition in Section B.2.2.2.

B.9.2.3 `tms::ClockMonotonic`

See definition in Section B.8.2.3.

B.9.2.4 `tms::EngineState`

PURPOSE: Report key parameters about the state of the engine.

TOPIC USAGE: Nested

EXTENSIBILITY: extensibility(APPENDABLE)

PATTERN: Structure

ATTRIBUTES:

Name — Type and Description

`operatingState`
: `EngineOperatingState`
Present operating state. Should be consistent with the EnergyStartStopState.

`oilPressure`
: `float32`
Oil pressure.
Annotations: *units*=pascal (Pa)

`coolantTemperature`
: `float32`
Coolant temperature.
Annotations: *units*=degree Celsius (degC)
Change History:

1. *Version:* 2.0.1, *Change:* modify, *Interoperability:* No-xtypes
Description: Removed minimum and maximum range specification.

`speed`
: `float32`
Speed.
Annotations: *units*=radian per second (rad/s)
Change History:

1. *Version:* 2.0.1, *Change:* modify, *Interoperability:* No-xtypes
Description: Removed minimum range specification.

`wetStack`

`optional<float32>`

wet stack per unit, mitigation required or indicated at 1.

Annotations: *min*=0, *max*=1, *units*=per unit (p.u.)

`runTime`

`float32`

Engine runtime.

Annotations: *units*=second (s)

Change History:

1. *Version:* 2.0.1, *Change:* modify, *Interoperability:* No-xtypes
   Description: Removed minimum range specification.
2. *Version:* 2.0.1, *Change:* modify, *Interoperability:* No-xtypes
   Description: Name changed from engineRuntime and units to second.

`fuelConsumptionRate`

`optional<float32>`

Fuel consumption rate at the present power level.

Annotations: *min*=0, *units*=liters per second (L/s)

B.9.2.5 `tms::EngineOperatingState`

PURPOSE: Indicate the operating state of an engine. Engines are not required to support all operating states.

EXTENSIBILITY: extensibility(APPENDABLE)

PATTERN: Enumeration

VALUES:

Name	Description
`EOS_UNKNOWN`	Unknown value. This is a generic placeholder representing the case in which an appropriate value could not be determined.
`EOS_STOPPED`	Engine is stopped.
`EOS_HEATING`	Engine is stopped and being heated for cold weather start.
`EOS_STARTING`	Engine is starting.
`EOS_IDLING`	Engine is started and running at a low, non-operational speed.
`EOS_WARMUP`	Engine is started and warming up to operational speed.
`EOS_RUNNING`	Engine is started and fully operational.

`EOS_COOLDOWN`
: Engine is started and cooling down in preparation to stop.

`EOS_STOPPING`
: Engine is stopping.

`EOS_DISABLED`
: Engine is disabled. It may be broken or undergoing maintenance.

B.9.2.6 `tms::FuelState`

PURPOSE: Provides dynamic state information regarding fuel levels and pumps.

TOPIC USAGE: Nested

EXTENSIBILITY: extensibility(APPENDABLE)

PATTERN: Structure

ATTRIBUTES:

Name	Type and Description
`fuelLevel`	`float32` The current fuel level of the device. Annotations: *units*=liter (L)
`fuelPumpRunning`	`boolean` True indicates that fuel is being pulled from external tank (fuel pump is on). False otherwise or if the device does not have a fuel pump.

B.9.2.7 `tms::GeneratorState`

PURPOSE: Report information on AVR and generator condition.

TOPIC USAGE: Nested

EXTENSIBILITY: extensibility(APPENDABLE)

PATTERN: Structure

ATTRIBUTES:

Name	Type and Description
`fieldCurrent`	`float32` field current, as controlled by AVR. Annotations: *units*=ampere (A)
`statorTemperature`	`float32` stator temperature. Annotations: *units*=degree Celsius (degC)

B.9.2.8 `tms::EnergyStorageState`

PURPOSE: Operating state of the energy storage device or unit.

TOPIC USAGE: Nested

EXTENSIBILITY: extensibility(APPENDABLE)

PATTERN: Structure

ATTRIBUTES:

Name	Type and Description
`minTemperature`	`float32` Present temperature of the energy storage components, minimum value measured. Annotations: *units*=degree Celsius (degC) Change History: 1. *Version:* 2.0.1, *Change:* modify, *Interoperability:* No-xtypes *Description:* Removed minimum and maximum range specification.
`averageTemperature`	`float32` Present temperature of the energy storage components, average of values measured. Annotations: *units*=degree Celsius (degC) Change History:

1. *Version:* 2.0.1, *Change:* modify, *Interoperability:* No-xtypes
   Description: Removed minimum and maximum range specification.

`maxTemperature`

`float32`

Present temperature of the energy storage components, maximum value measured.
Annotations: *units*=degree Celsius (degC)
Change History:

1. *Version:* 2.0.1, *Change:* modify, *Interoperability:* No-xtypes
   Description: Removed minimum and maximum range specification.

`runTime`

`float32`

Cumulative time that the energy storage has been actively charging or discharging. Usually displayed as an hour meter.
Annotations: *units*=second (s)

`storageCells`

`StorageCellStateSequence`

Information on each storage cell.

B.9.2.9 `tms::StorageCellStateSequence`

PURPOSE: A sequence of StorageCellState.

EXTENSIBILITY: extensibility(FINAL)

PATTERN: Typedef

ORIGINAL TYPE: `sequence<StorageCellState,0,MAXLEN_storageCells>`

B.9.2.10 `tms::StorageCellState`

PURPOSE: Power measurement for direct current devices.

TOPIC USAGE: Nested

EXTENSIBILITY: extensibility(APPENDABLE)

PATTERN: Structure

ATTRIBUTES:

Name	Type and Description
`cellNumber`	`StorageCellNumber` Identifies the energy storage cell being reported on within a device.
`cycleCounter`	`float32` Cumulative estimate of the energy storage wear, measured as the number of operating charge/discharge cycles. Set to NaN if not available.
`stateOfCharge`	`optional<float32>` Estimate of present charge as a fraction of rated capacity. 0=empty and 1=full. Annotations: *min*=0, *max*=1, *units*=per unit (p.u.) Change History: 1. *Version:* 2.0.1, *Change:* modify, *Interoperability:* No-xtypes *Description:* Made the field optional.
`temperature`	`float32` Temperature measurement of the individual storage cell. Set to NaN if not available. Annotations: *units*=degree Celsius (degC) Change History: 1. *Version:* 2.0.1, *Change:* modify, *Interoperability:* No-xtypes *Description:* Removed minimum and maximum range specification.
`voltage`	`float32` DC bus voltage across the energy storage cell terminals. Set to NaN if not available. Annotations: *units*=volt (V)

B.9.2.11 `tms::StorageCellNumber`

PURPOSE: Identifies an energy storage cell within a device.

EXTENSIBILITY: extensibility(APPENDABLE)

PATTERN: Typedef

ORIGINAL TYPE: `uint16`

CONSTANTS:

Name	Type, Value, and Description
`ONLY_CELL`	`StorageCellNumber`, 0 Number to use for a single-cell device.
`MAXLEN_storageCells`	`StorageCellNumber`, 50 Maximum number of cells that a device may report.

B.9.2.12 `tms::PowerElectronicsState`

PURPOSE: Operating state of the power electronics hardware.

TOPIC USAGE: Nested

EXTENSIBILITY: extensibility(APPENDABLE)

PATTERN: Structure

ATTRIBUTES:

Name	Type and Description
`temperature`	`float32` Present temperature. Annotations: *units*=degree Celsius (degC) Change History: 1. *Version:* 2.0.1, *Change:* modify, *Interoperability:* No-xtypes *Description:* Removed minimum and maximum range specification.

B.9.2.13 `tms::ThermalState`

PURPOSE: Provides operating information of the thermal management in this device.

TOPIC USAGE: Nested

EXTENSIBILITY: extensibility(APPENDABLE)

PATTERN: Structure

ATTRIBUTES:

Name	Type and Description
`thermalLoad`	`ThermalLoadSequence` Load for each thermalZone, in same order published in tms.ThermalInfo.

B.9.2.14 `tms::ThermalLoadSequence`

PURPOSE: A sequence of float32. Estimate of temperature control load as a fraction of capacity. 0=no load, 1=full rated load (hot or cold). Values above 1 indicate overload conditions.

EXTENSIBILITY: extensibility(FINAL)

PATTERN: Typedef

ORIGINAL TYPE: `sequence<float32,0,5>`

B.10 Energy Storage

This section provides updates on the state of charge of energy storage devices. A future version of this standard may provide requests to control the state of charge.

B.10.1 Data Flows

Figure B.19 illustrates all the topics in the data flow.

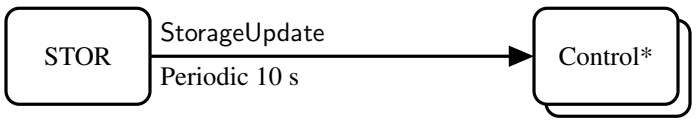


Figure B.19: Energy Storage data flow

Table B.113 provides an overview of how each topic is used.

Table B.113: Description of the Energy Storage topics.

Topic	Description
StorageUpdate	The dynamic state of a Storage device.

Table B.114 specifies the publishers and subscribers for each topic as well as usage requirements as described in Section A.8.7.

Table B.114: Participants of the Energy Storage topics.

Topic	Usage	MD	MC	MON	SRC	STOR	DIST LOAD	CONV
StorageUpdate	Required	Sub	Sub	Sub	Pub			

This participants table has the following interpretation.

1. StorageUpdate shall be published by STOR devices.
2. StorageUpdate may be subscribed by MD, MC, and MON devices.

Table B.115 specifies the timing for when sample updates are sent to each topic. These parameters are defined in Section A.8.5.

Table B.115: Timing of the Energy Storage topics.

Topic	Data Trigger	Rate Class	Burst Size	QoS Profile
StorageUpdate	Periodic 10 s	10 s	1	Slow

B.10.2 Data Types

Table B.116 specifies the data types that are used as sample values for each topic.

Table B.116: Data types of the Energy Storage topics.

Topic	Data Type
StorageUpdate	`tms::StorageUpdate`

Topic-level data types are defined in the order shown in Table B.116. Then nested data types are defined in the order of appearance. Cross-references are provided for data types that have already been defined.

B.10.2.1 `tms::StorageUpdate`

PURPOSE: Report the state of a storage device.

TOPIC USAGE: StorageUpdate

EXTENSIBILITY: extensibility(APPENDABLE)

PATTERN: Structure

ATTRIBUTES:

Name	Type and Description
`deviceId`	`Identity` The device described by this structure. Annotations: *keyval*
`internalVoltage`	`float32` Measurement indicating the DC bus voltage between the energy storage and power converter. Annotations: *units*=volt (V)
`stateOfCharge`	`optional<float32>` Estimate of present charge as a fraction of rated capacity. 0=empty and 1=full. Annotations: *min*=0, *max*=1 Change History: 1. *Version:* 2.0.1, *Change:* modify, *Interoperability:* No-xtypes *Description:* Made the field optional.
`availableEnergy`	`float32` Estimate of present energy available for discharging to the microgrid. Annotations: *units*=joule (J)
`holdTime`	`float32` Estimate of how long the energy storage can operate at the present power rate. Annotations: *units*=second (s)
`chargeTime`	`Curve2D` Estimate of how long the energy storage can charge at different power levels, without exceeding the highStateOfCharge. Annotations: *units*=watt, second (W, s)
`dischargeTime`	`Curve2D` Estimate of how long the energy storage can discharge at different power levels, without exceeding the lowStateOfCharge. Annotations: *units*=watt, second (W, s)

`maxChargeTime`

`Curve2D`

Estimate of how long the energy storage can charge at different power levels, with battle short enabled. Must be greater than or equal to the chargeTime.

Annotations: *units*=watt, second (W, s)

`maxDischargeTime`

`Curve2D`

Estimate of how long the energy storage can discharge at different power levels, with battle short enabled. Must be greater than or equal to the dischargeTime.

Annotations: *units*=watt, second (W, s)

`maxChargeRate`

`optional<float32>`

The maximum rate of charge based on current temperatures and state of charge. 0 to Inf.

Annotations: *min*=0, *units*=watt (W)

Change History:

1. *Version:* 2.0.1, *Change:* modify, *Interoperability:* No-xtypes
   Description: Made the field optional.

`maxDischargeRate`

`optional<float32>`

The maximum rate of discharge based on current temperatures and state of charge. -Inf to 0.

Annotations: *max*=0, *units*=watt (W)

Change History:

1. *Version:* 2.0.1, *Change:* modify, *Interoperability:* No-xtypes
   Description: Made the field optional.

B.10.2.2 `tms::Identity`

See definition in Section B.2.2.2.

B.10.2.3 `tms::Curve2D`

See definition in Section B.5.2.58.

B.10.2.4 `tms::Point2DSequence`

See definition in Section B.5.2.59.

B.10.2.5 `tms::Point2D`

See definition in Section B.5.2.60.

B.11 Operator Input

Operator Input allows the MD operator to influence MC control over the microgrid. These requests can be used when the operator requires that a TMS device be in a specific state, such as off-line to perform maintenance, or to specify how a TMS device handles its functions. For example, an operator might enable load curtailment by assigning priorities to the LOAD devices and setting the minimum priority a device must have before it can be considered for load curtailment.

B.11.1 Data Flows

When the operator provides new input, the MD broadcasts an `OperatorIntentRequest` to all MCs. Since this is a `GridRequest`, the MD does not expect a reply from each individual MC. Instead, MCs provide feedback by updating the OperatorIntentState.

Figure B.20 illustrates all the topics in the data flow.

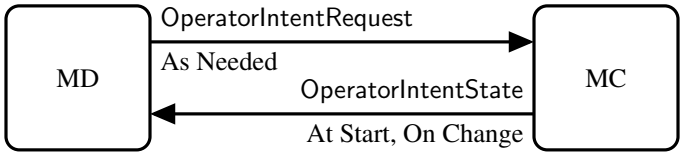


Figure B.20: Operator Input data flow

The MD and MC responsibilities for the Operator Input topic are:

- MD (Microgrid Dashboard)
  - Allow the operator to define and activate the `OperatorIntent` data type
  - Only a single operator intent can be active at a time
- MC (Microgrid Controller)
  - Transition from the present `OperatorIntent` to a new `OperatorIntent`
  - Set the default `OperatorIntent` if the MD has not set one
  - Persist the MD-activated `OperatorIntent` for use on re-start

Table B.118 provides an overview of how each topic is used.

Table B.118: Description of the Operator Input topics.

Topic	Description
OperatorIntentState	The last published OperatorIntent received by a Microgrid Controller
OperatorIntentRequest	Operator defined microgrid control directives that influence how the microgrid behaves.

Table B.119 specifies the publishers and subscribers for each topic as well as usage requirements as described in Section A.8.7.

Table B.119: Participants of the Operator Input topics.

Topic	Usage	MD	MC	MON	SRC	STOR	DIST	LOAD	CONV
OperatorIntentState	Conditional		Pub						
		Sub		Sub					
OperatorIntentRequest	Optional	Pub							
			Sub	Sub					

This participants table has the following interpretation.

1. OperatorIntentState is conditional on SUPPORTS_REQUEST.
2. OperatorIntentState shall be published by MC devices that support the corresponding request topic.
3. OperatorIntentState may be subscribed by MD and MON devices.
4. OperatorIntentRequest shall be published by MD devices that have this topic listed in `TopicInfo.optionalTopics`.
5. OperatorIntentRequest may be subscribed by MC and MON devices.

Table B.120 specifies the timing for when sample updates are sent to each topic. These parameters are defined in Section A.8.5.

Table B.120: Timing of the Operator Input topics.

Topic	Data Trigger	Rate Class	Burst Size	QoS Profile
OperatorIntentState	At Start, On Change	10 s	10	PublishLast
OperatorIntentRequest	As Needed	10 s	10	Command

B.11.2 Data Types

The operator input is provided at the following three control levels: microgrid, device, and power port. The microgrid level applies to overall coordination and thus all devices, the device level applies to individual devices, and the power port level applies to individual ports on multi-port devices (CONV and DIST).

Source dispatch and load shedding are coordinated features that span all three levels. The microgrid level controls the amount of spinning reserve and whether shedding is enabled. The device level prioritizes which SRC and STOR devices provide power, which LOAD devices are shed, and whether CONV devices import or export power. The power port level prioritizes which DIST power ports are shed.

The operator priorities may also be used to coordinate maintenance actions. Before maintenance begins, `OPT_NEVER_OPERATE` causes the MC to remove a device from service. The operator may then turn it off and disconnect it without disrupting other operations. After maintenance completes, `OPT_ALWAYS_OPERATE` or `OPT_NUMERIC_RANK` causes the MC to return the device to service, preserving any previously assigned numeric rank.

Table B.121 specifies the data types that are used as sample values for each topic.

Table B.121: Data types of the Operator Input topics.

Topic	Data Type
OperatorIntentState	`tms::OperatorIntentState`
OperatorIntentRequest	`tms::OperatorIntentRequest`

Topic-level data types are defined in the order shown in Table B.121. Then nested data types are defined in the order of appearance. Cross-references are provided for data types that have already been defined.

B.11.2.1 `tms::OperatorIntentState`

PURPOSE: The active OperatorIntent.

TOPIC USAGE: OperatorIntentState

EXTENSIBILITY: extensibility(APPENDABLE)

PATTERN: Structure

ATTRIBUTES:

Name	Type and Description
`deviceId`	`Identity` The device described by this structure. Annotations: *keyval*
`activeOperatorIntent`	`OperatorIntent` The complete active OperatorIntent.

B.11.2.2 `tms::OperatorIntentRequest`

PURPOSE: Used to request activation of an OperatorIntent.

TOPIC USAGE: OperatorIntentRequest

EXTENSIBILITY: extensibility(APPENDABLE)

PATTERN: Structure

ATTRIBUTES:

Name	Type and Description
`requestId`	`GridRequest` The device sending this request. Annotations: *keyval*
`sequenceId`	`tms::RequestSequence` Request sequence data used to associate a request and returning reply.
`desiredOperatorIntent`	`OperatorIntent` The OperatorIntent to activate.

B.11.2.3 `tms::Identity`

See definition in Section B.2.2.2.

B.11.2.4 `tms::OperatorIntent`

PURPOSE: Specifies a complete set of operator directives used by the MC to control the microgrid.

TOPIC USAGE: Nested

EXTENSIBILITY: extensibility(APPENDABLE)

PATTERN: Structure

ATTRIBUTES:

Name	Type and Description
`requestId`	`GridRequest` The device that defined this structure.
`intentType`	`OperatorIntentType` The operator intent type.
`microgrid`	`MicrogridIntent` The microgrid level directives.
`devices`	`DeviceIntentSequence` The device level directives.
`powerPorts`	`PowerPortIntentSequence` The power port level directives.

B.11.2.5 `tms::GridRequest`

PURPOSE: Key value for a request that targets all devices in the grid.

TOPIC USAGE: Nested

EXTENSIBILITY: extensibility(APPENDABLE)

PATTERN: Structure

ATTRIBUTES:

Name	Type and Description
`requestingDeviceId`	`Identity` Identity of the device that sent this request.

B.11.2.6 `tms::OperatorIntentType`

PURPOSE: Specify the type of Operator Intent.

EXTENSIBILITY: extensibility(APPENDABLE)

PATTERN: Enumeration

VALUES:

Name	Description
`OIT_UNKNOWN`	Unknown value. This is a generic placeholder representing the case in which an appropriate value could not be determined.
`OIT_DEFAULT_INTENT`	Microgrid Controller defined default operator intent.
`OIT_OPERATOR_DEFINED`	Microgrid Dashboard activated operator intent.

B.11.2.7 `tms::MicrogridIntent`

PURPOSE: Specify the microgrid level operator intent.

TOPIC USAGE: Nested

EXTENSIBILITY: extensibility(APPENDABLE)

PATTERN: Structure

ATTRIBUTES:

Name	Type and Description
`operatingModeValue`	`OperatingMode` Overall operating mode for the microgrid.
`loadSharingIntentValue`	`LoadSharingIntent` Load sharing Microgrid Controller directives.
`storageIntentValue`	`optional<StorageIntent>` Present if the operator provides Storage device Microgrid Controller directives.

B.11.2.8 `tms::OperatingMode`

PURPOSE: Indicate the microgrid level operating mode

EXTENSIBILITY: extensibility(APPENDABLE)

PATTERN: Enumeration

VALUES:

Name	Description
`OPM_UNKNOWN`	Unknown value. This is a generic placeholder representing the case in which an appropriate value could not be determined.
`OPM_NORMAL`	Normal operating mode.
`OPM_EMERGENCY`	Emergency operating mode.
`OPM_SILENT_WATCH`	Silent watch mode.

B.11.2.9 `tms::LoadSharingIntent`

PURPOSE: Specify the microgrid level load sharing intent.

TOPIC USAGE: Nested

EXTENSIBILITY: extensibility(APPENDABLE)

PATTERN: Structure

ATTRIBUTES:

Name	Type and Description
`increaseCapacityThreshold`	`float32` Per unit of used capacity that triggers the MC to add additional power source to the microgrid. Per unit is in the range 0 - 1. Annotations: *min*=0, *max*=1, *units*=per unit (p.u.)
`decreaseCapacityThreshold`	`float32` Per unit of used capacity that triggers the MC to remove a power source from the microgrid. Per unit is in the range 0 - 1 and must be less than increaseCapacityThreshold. Annotations: *min*=0, *max*=1, *units*=per unit (p.u.)
`decreaseCapacityDuration`	`float32` The time duration the decreaseCapacityThreshold must be met before the Microgrid Controller removes a power source from the microgrid. This duration prevents power sources from being added and removed in rapid succession. Annotations: *units*=second (s)
`priorityLoadCurtailmentAllowed`	`OperatorPriority` The priority above which the Microgrid Controller is authorized to trigger load curtailment. Load curtailment is accomplished by stopped a load device. A value of OPT_ALWAYS_OPERATE disables load curtailment.
`priorityPowerPortSheddingAllowed`	`OperatorPriority` The priority above which the Microgrid Controller is authorized to trigger port power load shedding. Port power load shedding is accomplished by opening a power port on a distribution device. A value of OPT_ALWAYS_OPERATE disables circuit load shedding.

B.11.2.10 `tms::OperatorPriority`

PURPOSE: Specifies the operator priority for a single device or power port.

TOPIC USAGE: Nested

EXTENSIBILITY: extensibility(APPENDABLE)

PATTERN: Structure

ATTRIBUTES:

Name	Type and Description
`priorityType`	`OperatorPriorityType` The operator priority type.
`numericRank`	`int16` Rank values indicate priority order, where 0 is the highest priority and increasing values are lower priority. The numericRank is used when priorityType is OPT_NUMERIC_RANK.

B.11.2.11 `tms::OperatorPriorityType`

PURPOSE: Specify the type of Operator priority.

EXTENSIBILITY: extensibility(APPENDABLE)

PATTERN: Enumeration

VALUES:

Name	Description
`OPT_UNKNOWN`	Unknown value. This is a generic placeholder representing the case in which an appropriate value could not be determined.
`OPT_NEVER_OPERATE`	Transition the device to the lowest supported EnergyStartStopLevel or open the power port.
`OPT_ALWAYS_OPERATE`	Transition the device to EnergyStartStopLevel ESSL_OPERATIONAL or close the power port.
`OPT_NUMERIC_RANK`	Priority based on OperatorPriority.numericRank.

B.11.2.12 `tms::StorageIntent`

PURPOSE: Specify the storage device intent.

TOPIC USAGE: Nested

EXTENSIBILITY: extensibility(APPENDABLE)

PATTERN: Structure

ATTRIBUTES:

Name	Type and Description
`reservedStateOfCharge`	`float32` The desired total state of charge to hold in reserve. Annotations: *min*=0, *max*=1, *units*=per unit (p.u.)
`timeTillReservedSoc`	`ClockMonotonic` Time till the reservedStateOfCharge is reached and maintained.
`priorityLoadSheddingAllowed`	`OperatorPriority` The priority above which the MC is authorized to trigger load shedding. Load shedding is accomplished by stopped a smart load device. A value of OPT_ALWAYS_OPERATE disables load curtailment.
`priorityPowerPortSheddingAllowed`	`OperatorPriority` The priority above which the MC is authorized to trigger power port load curtailment. Power port shedding is accomplished by opening a power port on a distribution device. A value of OPT_ALWAYS_OPERATE disables circuit load shedding.
`reservedDeviceIds`	`IdentitySequence` The list of storage devices that should reach the reservedStateOfCharge. At empty list indicates all available storage devices.

B.11.2.13 `tms::ClockMonotonic`

See definition in Section B.8.2.3.

B.11.2.14 `tms::IdentitySequence`

PURPOSE: A sequence of Identity.

EXTENSIBILITY: extensibility(FINAL)

PATTERN: Typedef

ORIGINAL TYPE: `sequence<Identity,0,MAX_PORTS>`

B.11.2.15 `tms::DeviceIntentSequence`

PURPOSE: A sequence of DeviceIntent.

EXTENSIBILITY: extensibility(FINAL)

PATTERN: Typedef

ORIGINAL TYPE: `sequence<DeviceIntent,0,200>`

B.11.2.16 `tms::DeviceIntent`

PURPOSE: Specify the device level operator intent.

TOPIC USAGE: Nested

EXTENSIBILITY: extensibility(APPENDABLE)

PATTERN: Structure

ATTRIBUTES:

Name	Type and Description
`deviceId`	`Identity` The device described by this structure.
`battleShort`	`boolean` The device Battle Short state.
`priority`	`OperatorPriority` The operator assigned priority of the device.

B.11.2.17 `tms::PowerPortIntentSequence`

PURPOSE: A sequence of PowerPortIntent.

EXTENSIBILITY: extensibility(FINAL)

PATTERN: Typedef

ORIGINAL TYPE: `sequence<PowerPortIntent,0,4000>`

B.11.2.18 `tms::PowerPortIntent`

PURPOSE: Specify the device power port level operator intent.

TOPIC USAGE: Nested

EXTENSIBILITY: extensibility(APPENDABLE)

PATTERN: Structure

ATTRIBUTES:

Name	Type and Description
`deviceId`	`Identity` The device described by this structure.
`portNumber`	`PowerPortNumber` Number shown on the device exterior to represent this port.
`priority`	`OperatorPriority` The operator assigned priority of this power port.

B.11.2.19 `tms::PowerPortNumber`

See definition in Section B.2.2.4.

B.11.2.20 `tms::RequestSequence`

See definition in Section B.2.2.5.

B.12 Diagnostics Reporting

Diagnostics reporting provides notifications ranging from notes and routine maintenance to warnings and alarms. In general, power devices report information about their internal health, and the MC and MD use this information to monitor them. Devices, especially the MC, may also report diagnostics about microgrid power or communications.

This functionality may signal the need for maintenance. It is not intended to replace technical manuals or internal interfaces used for diagnostics troubleshooting.

Each device publishes a list of its active diagnostic trouble codes that indicate conditions where performance is degraded or operator attention is required. Codes are removed from this list when they clear (are no longer active), and they should automatically clear or downgrade to lower severity when the condition is detected to no longer exist. An on-device user interface should be used to clear codes that do not automatically clear after appropriate maintenance activities. If too many codes are active at once, then only the most severe are reported and the rest are simply counted.

Each trouble code describes how the issue was detected, when it occurred, how severely this impacts the device, and a short textual hint. The MC should use the detection, time, and severity to determine whether a response is required. The MC should not rely on the hint. The MD should display the hint and provide access to the other information.

B.12.1 Data Flows

Figure B.21 illustrates all the topics in the data flow.

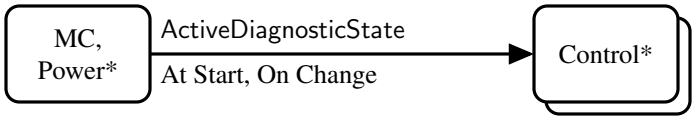


Figure B.21: Diagnostics Reporting data flow

This data flow does not define any response. The control devices should respond as necessary using the other functions defined in this standard.

For example, if a generator set reports that its engine is overheating, the MC may choose one of the following actions.

- No action. There may be reason to believe the load will decrease and clear the alarm, or potential equipment damage may be acceptable (battle short conditions).
- Disconnect. Request the generator set to stop load sharing and cool down.
- Dispatch. Request another device to produce more or consume less power, thus reducing the load on this generator set.

Table B.135 provides an overview of how each topic is used.

Table B.135: Description of the Diagnostics Reporting topics.

Topic	Description
ActiveDiagnosticState	Reports a device diagnostics state including information and alarm states.

Table B.136 specifies the publishers and subscribers for each topic as well as usage requirements as described in Section A.8.7.

Table B.136: Participants of the Diagnostics Reporting topics.

Topic	Usage	MD	MC	MON	SRC	STOR	DIST	LOAD	CONV
ActiveDiagnosticState	Required	Sub	Pub Sub	Sub	Pub	Pub	Pub	Pub	Pub

This participants table has the following interpretation.

1. ActiveDiagnosticState shall be published by MC, SRC, STOR, DIST, LOAD, and CONV devices.

2. ActiveDiagnosticState may be subscribed by MD, MC, and MON devices.

Table B.137 specifies the timing for when sample updates are sent to each topic. These parameters are defined in Section A.8.5.

Table B.137: Timing of the Diagnostics Reporting topics.

Topic	Data Trigger	Rate Class	Burst Size	QoS Profile
ActiveDiagnosticState	At Start, On Change	10 s	10	PublishLast

B.12.2 Data Types

The `SuspectParameterNumber` and `FailureModeIndicator` data types were designed to be compatible with SAE J1939-73 Diagnostic Trouble Codes (DTCs). Other aspects of J1939, including wire formats and timing requirements, do not apply.

Table B.138 specifies the data types that are used as sample values for each topic.

Table B.138: Data types of the Diagnostics Reporting topics.

Topic	Data Type
ActiveDiagnosticState	`tms::ActiveDiagnosticState`

Topic-level data types are defined in the order shown in Table B.138. Then nested data types are defined in the order of appearance. Cross-references are provided for data types that have already been defined.

B.12.2.1 `tms::ActiveDiagnosticState`

PURPOSE: Active diagnostic messages.

TOPIC USAGE: ActiveDiagnosticState

EXTENSIBILITY: extensibility(APPENDABLE)

PATTERN: Structure

ATTRIBUTES:

Name	Type and Description
`deviceId`	`Identity` The device described by this structure. Annotations: *keyval*
`codes`	`DiagnosticTroubleCodeSequence` Active trouble codes. A given suspect parameter can only appear once in this sequence.
`overflow`	`uint32` Count of trouble codes that were truncated from the codes because the maximum sequence length was exceeded. A count of 0 indicates that all active trouble codes are listed.

B.12.2.2 `tms::Identity`

See definition in Section B.2.2.2.

B.12.2.3 `tms::DiagnosticTroubleCodeSequence`

PURPOSE: A sequence of DiagnosticTroubleCode.

EXTENSIBILITY: extensibility(FINAL)

PATTERN: Typedef

ORIGINAL TYPE: `sequence<DiagnosticTroubleCode,0,64>`

B.12.2.4 `tms::DiagnosticTroubleCode`

PURPOSE: Represent an individual alarm or fault condition originating from the indicated sensor or subsystem, as identified by the affectedIdentityId, portNumber, and suspectParameter.

TOPIC USAGE: Nested

EXTENSIBILITY: extensibility(APPENDABLE)

PATTERN: Structure

ATTRIBUTES:

Name	Type and Description
`portNumber`	`optional<PowerPortNumber>` Number shown on the device exterior to represent this port. Present when this condition pertains to a specific power port and absent when it pertains to the entire device.
`suspectParameter`	`SuspectParameterNumber` Parameter representing this DTC.
`failureMode`	`FailureModeIndicator` Type of fault for this parameter.
`occurrenceCount`	`uint32` Number of times this DTC was raised. This field increments on a state change from inactive to active. In order to increment, the cleared state must be sensed. The count must not increment simply due to restarts. The count should be stored in non-volatile memory in order to preserve it over power cycles. The count saturates at 4294967294 ($2^{32} - 2$). The value 4294967295 ($2^{32} - 1$) indicates that the count is not available.
`timeRaised`	`ClockMonotonic` Timestamp when this DTC was raised or activated.
`severity`	`DtcSeverity` Expected impact of the DTC on grid operation. Intended to support automatic handling of parameters that are unknown to the receiving system.
`estimateTime`	`optional<ClockMonotonic>` Estimate of when the DTC severity will change, assuming continued operation under present conditions.
`estimateSeverity`	`optional<DtcSeverity>` Expected DTC severity at the estimateTime.

`hint`

`String32`

Short, human-readable text description of the DTC. This should summarize to the operator what the issue is (e.g. "oil pressure" or "low oil pressure"). Intended to support human operators for parameters that are unknown to the receiving system.

`affectedIdentityId`

`optional<Identity>`

The device described by this structure. Present when the device or platform experiencing the alarm or fault condition is other than the device publishing the tms::ActiveDiagnosticState.

Change History:

1. *Version:* 2.0.3, *Change:* add, *Interoperability:* Yes
   Description: Allow a device to report DiagnosticTroubleCodes on behalf of another device.

B.12.2.5 `tms::PowerPortNumber`

See definition in Section B.2.2.4.

B.12.2.6 `tms::SuspectParameterNumber`

PURPOSE: Identifies the sensor or subsystem at issue or a particular event or condition.

EXTENSIBILITY: extensibility(APPENDABLE)

PATTERN: Typedef

ORIGINAL TYPE: `uint32`

CONSTANTS:

Name	Type, Value, and Description
`SPN_RANGE_J1939`	`uint32`, 0 First SPN in the range reserved for SAE J1939 standard parameters Change History: 1. *Version:* 2.0.3, *Change:* add, *Interoperability:* Yes *Description:* Added range start and SPN_J1939_* constants
`SPN_J1939_FUEL_LEVEL_1`	`uint32`, 96 Fuel Level 1 (tms::FuelState.fuelLevel)
`SPN_J1939_ENGINE_OIL_PRESSURE_1`	`uint32`, 100 Engine Oil Pressure (tms::EngineState.oilPressure)
`SPN_J1939_ENGINE_COOLANT_TEMPERATURE`	`uint32`, 110 Engine Coolant Temperature (tms::EngineState.coolantTemperature)
`SPN_J1939_KEY_SWITCH_BATTERY_POTENTIAL`	`uint32`, 158 Key Switch Battery Potential (TMS: internal starter battery for control and power hardware, not the main STOR pack)
`SPN_J1939_ENGINE_SHUTDOWN_OVERRIDE_SWITCH`	`uint32`, 1237 Engine Shutdown Override Switch (TMS: Battle Short enabled)
`SPN_J1939_EMERGENCY_SHUTDOWN_SWITCH`	`uint32`, 3585 Emergency Shutdown Switch (TMS: Emergency Stop enabled)
`SPN_RANGE_PROPRIETARY`	`uint32`, 516096 First SPN in the range reserved for vendor-specific parameters Change History: 1. *Version:* 2.0.3, *Change:* add, *Interoperability:* Yes *Description:* Added range start
`SPN_RANGE_TMS`	`uint32`, 524288 First SPN in the range reserved for TMS standard parameters

Change History:

1. *Version:* 2.0.3, *Change:* add, *Interoperability:* Yes
   Description: Added range start and SPN_TMS_* constants

`SPN_TMS_CONTROL_LOCKOUT_LOCAL`
: `uint32`, 524300
Local automatic control has been disabled, as for maintenance or troubleshooting activities (FMI_CONDITION_EXISTS)

`SPN_TMS_CONTROL_LOCKOUT_REMOTE`
: `uint32`, 524301
Remote automatic control has been disabled, including all TMS requests (FMI_CONDITION_EXISTS)

`SPN_TMS_TRIP_VOLTAGE`
: `uint32`, 524400
Indicates a protections trip (PSR_AUTO_PROTECTION) due to over-voltage (FMI_HIGH), under-voltage (FMI_LOW), or rate of change of voltage (FMI_ABNORMAL_ROC). Corresponds to ANSI C37.2 numbers 59* and 27*.

`SPN_TMS_TRIP_FREQUENCY`
: `uint32`, 524401
Indicates a protections trip (PSR_AUTO_PROTECTION) due to over-frequency (FMI_HIGH), under-frequency (FMI_LOW), or rate of change of frequency (FMI_ABNORMAL_ROC). Corresponds to ANSI C37.2 numbers 81O, 81U, and 81R.

`SPN_TMS_TRIP_AMPERAGE`
: `uint32`, 524402
Indicates a protections trip (PSR_AUTO_PROTECTION) due to over-amperage (FMI_HIGH), under-amperage (FMI_LOW), or rate of change of amperage (FMI_ABNORMAL_ROC). Corresponds to ANSI C37.2 numbers 51* and 37.

`SPN_TMS_TRIP_POWER`
: `uint32`, 524403
Indicates a protections trip (PSR_AUTO_PROTECTION) due to over-power (FMI_HIGH), under-power / reverse power (FMI_LOW), or rate of change of power (FMI_ABNORMAL_ROC). Corresponds to ANSI C37.2 numbers 32* and 37P.

`SPN_TMS_TRIP_ENERGY`
: `uint32`, 524404
Indicates a protections trip (PSR_AUTO_PROTECTION) due to over-energy (FMI_HIGH), under-energy (FMI_LOW), or rate of change of energy (FMI_ABNORMAL_ROC).

`SPN_TMS_TRIP_GROUND`
: `uint32`, 524405
Indicates a protections trip (PSR_AUTO_PROTECTION) due to a grounding fault. Corresponds to ANSI C37.2 numbers 64*

`SPN_TMS_TRIP_TEMPERATURE`
: `uint32`, 524406
Indicates a protections trip (PSR_AUTO_PROTECTION) due to over-temperature (FMI_HIGH) or under-temperature (FMI_LOW). Does not apply to thermal-based current or power protections.

This data type extends the SAE J1939 suspect parameter number (SPN) as defined in Table B.142.

Table B.142: Suspect Parameter Number value ranges

Name	Start	End	Description
SAE J1939	0	516096	SAE J1939 standard parameters
Proprietary	516096	524287	Vendor-specific parameters (range matches J1939)
TMS	524288	$2^{32} - 1$	TMS standard parameters

Any device that uses SAE J1939 SPN values shall use them in accordance with that standard. Any device that uses proprietary (vendor-specific) SPN values shall document every such SPN in a technical manual that is provided with the device. Any device that uses TMS SPN values shall use them in accordance with this standard. In particular, devices shall not publish SPNs in the TMS range unless there are matching constants defined in this standard.

Constants have been defined for selected SAE J1939 SPN values that may be applied to TMS and for all SPNs defined in this standard.

B.12.2.7 `tms::FailureModeIndicator`

PURPOSE: Describes the type of fault. See SAE J1939.

EXTENSIBILITY: extensibility(APPENDABLE)

PATTERN: Enumeration

VALUES:

Name
: Description

`FMI_UNKNOWN`
: No information is available, either due to design or malfunction.

`FMI_HIGH_MOST_SEVERE_HIGH`
: 0, Data valid but above normal operational range, most severe level.

`FMI_HIGH_MOST_SEVERE_LOW`
: 1, Data valid but below normal operational range, most severe level.

`FMI_DATA_ERRATIC`
: 2, Data erratic, intermittent or incorrect (rationality).

`FMI_VOLTAGE_HIGH`
: 3, Voltage above normal or shorted to high source.

`FMI_VOLTAGE_LOW`
: 4, Voltage below normal or shorted to low source.

`FMI_CURRENT_LOW`
: 5, Current below normal or open circuit.

`FMI_CURRENT_HIGH`
: 6, Current above normal or ground circuit.

`FMI_NOT_RESPONDING`
: 7, Mechanical system not responding or out of adjustment.

`FMI_ABNORMAL_FREQUENCY`
: 8, Abnormal frequency or pulse width or period.

`FMI_ABNORMAL_UPDATE_RATE`
: 9, Abnormal update rate.

`FMI_ABNORMAL_RATE_OF_CHARGE`
: 10, Abnormal rate of change.

`FMI_FAILURE_CODE_NOT_IDENTIFIABLE`
: 11, Failure code not identifiable.

`FMI_BAD_INTELLIGENT_DEVICE`
: 12, Bad intelligent device or component.

`FMI_OUT_OF_CALIBRATION`
: 13, Out of calibration.

`FMI_SPECIAL_INSTRUCTIONS`
: 14, Special instructions.

`FMI_HIGH_LEAST_SEVERE`
: 15, Data valid but above normal operational range, least severe level.

`FMI_HIGH_MODERATELY_SEVERE`
: 16, Data valid but above normal operational range, moderately severe level.

`FMI_LOW_LEAST_SEVERE`
: 17, Data valid but below normal operational range, least severe level.

`FMI_LOW_MODERATELY_SEVERE`
: 18, Data valid but below normal operational range, moderately severe level.

`FMI_RECEIVED_DATA_ERROR`
: 19, Received network data in error (multiplexed data).

`FMI_DATA_DRIFTED_HIGH`
: 20, Data drifted high (rationality high).

`FMI_DATA_DRIFTED_LOW`
: 21, Data drifted low (rationality low).

`FMI_CONDITION_EXISTS`
: 31, Condition exists.

B.12.2.8 `tms::ClockMonotonic`

See definition in Section B.8.2.3.

B.12.2.9 `tms::DtcSeverity`

PURPOSE: Describe how a DTC affects the availability of the device

EXTENSIBILITY: extensibility(APPENDABLE)

PATTERN: Enumeration

VALUES:

Name	Description
`SEV_UNKNOWN`	Unknown value. Never transmitted in this version of the standard.
`SEV_1_CLEAR`	DTC has cleared. Never transmitted in this version of the standard.
`SEV_2_INFORMATIVE`	Informational only. No performance impact is expected.
`SEV_3_PREVENTATIVE`	Preventative maintenance has been scheduled.
`SEV_4_DEGRADED`	Poor configuration or operation resulting in degraded performance, including reduced efficiency and increased emissions.
`SEV_5_WARNING`	Increased wear and premature failure likely.
`SEV_6_MINOR`	The device may be failing to meet operating performance criteria. The DTC may represent cause or effect.
`SEV_7_MAJOR`	This condition may cause protection logic to enter a controlled shutdown sequence. A corrective action may be required before the device can be restarted.
`SEV_8_CRITICAL`	A loss of essential functionality has been detected. Routine maintenance or repair may be required.
`SEV_9_FATAL`	A loss of essential functionality has been detected. Permanent damage may have occurred.
`SEV_10_HUMAN_SAFETY`	Potential user safety risk.

B.12.2.10 `tms::String32`

See definition in Section B.2.2.8.

B.13 Microgrid Controller Selection

All power devices follow the MC selection algorithm defined in Section A.7.5 and report the selected MC using ActiveMicrogridControllerState. A future version of this standard may

provide support for the MC to modify the selection algorithm to prefer MCs by identity, type, or another attribute.

B.13.1 Data Flows

Figure B.22 illustrates all the topics in the data flow.

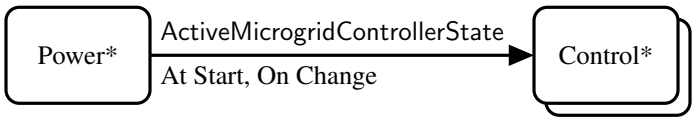


Figure B.22: Microgrid Controller Selection data flow

Table B.145 provides an overview of how each topic is used.

Table B.145: Description of the Microgrid Controller Selection topics.

Topic	Description
ActiveMicrogridControllerState	The selected master Microgrid Controller for a specific power device.

Table B.146 specifies the publishers and subscribers for each topic as well as usage requirements as described in Section A.8.7.

Table B.146: Participants of the Microgrid Controller Selection topics.

Topic	Usage	MD	MC	MON	SRC	STOR	DIST	LOAD	CONV
ActiveMicrogridControllerState	Required	Sub	Sub	Sub	Pub	Pub	Pub	Pub	Pub

This participants table has the following interpretation.

1. ActiveMicrogridControllerState shall be published by SRC, STOR, DIST, LOAD, and CONV devices.

2. ActiveMicrogridControllerState may be subscribed by MD, MC, and MON devices.

Table B.147 specifies the timing for when sample updates are sent to each topic. These parameters are defined in Section A.8.5.

Table B.147: Timing of the Microgrid Controller Selection topics.

Topic	Data Trigger	Rate Class	Burst Size	QoS Profile
ActiveMicrogridControllerState	At Start, On Change	1 s	1	PublishLast

B.13.2 Data Types

Table B.148 specifies the data types that are used as sample values for each topic.

Table B.148: Data types of the Microgrid Controller Selection topics.

Topic	Data Type
ActiveMicrogridControllerState	`tms::ActiveMicrogridControllerState`

Topic-level data types are defined in the order shown in Table B.148. Then nested data types are defined in the order of appearance. Cross-references are provided for data types that have already been defined.

B.13.2.1 `tms::ActiveMicrogridControllerState`

PURPOSE: Identify which :MC currently has write access to the active configuration.

TOPIC USAGE: ActiveMicrogridControllerState

EXTENSIBILITY: extensibility(APPENDABLE)

PATTERN: Structure

ATTRIBUTES:

Name	Type and Description
`deviceId`	`Identity` The device described by this structure. Annotations: *keyval*
`masterId`	`optional<Identity>` The MC that currently controls this device. Present if a MC is selected, absent otherwise.

DESCRIPTION:

This information is published by each controllable device.

B.13.2.2 `tms::Identity`

See definition in Section B.2.2.2.

B.14 Device Parameters

Device parameters provide a controlled mechanism for vendors to extend this data model with custom values. There are two types of device parameter. Control parameters are read/write values suitable for tuning operational behavior. Metric parameters are

read-only values suitable for reporting status. All parameters must be announced using `tms::ParameterMetadata` in `tms::DeviceInfo`. Additional documentation may be required to enable interoperability.

Device parameters follow a naming convention in order to maximize interoperability. All parameter names beginning with "tms" are reserved for definition by this standard. Proprietary parameter names should begin with a manufacturer-specific prefix.

This version of TMS defines the parameters given in Table B.150.

Table B.150: Device parameters defined by TMS.

Name	Units	Description
tms.avr.cutoffFrequency	Hz	Automatic voltage regulator low-pass filter cutoff frequency.
tms.avr.proportionalGain	p.u.	Automatic voltage regulator proportional gain.
tms.avr.integralGain	p.u. / s	Automatic voltage regulator integral gain.
tms.avr.derivativeGain	p.u. · s	Automatic voltage regulator derivative gain.
tms.gov.cutoffFrequency	Hz	Throttle governor low-pass filter cutoff frequency.
tms.gov.proportionalGain	p.u.	Throttle governor proportional gain.
tms.gov.integralGain	p.u. / s	Throttle governor integral gain.
tms.gov.derivativeGain	p.u. · s	Throttle governor derivative gain.

B.14.1 Data Flows

Figure B.23 illustrates all the topics in the data flow.

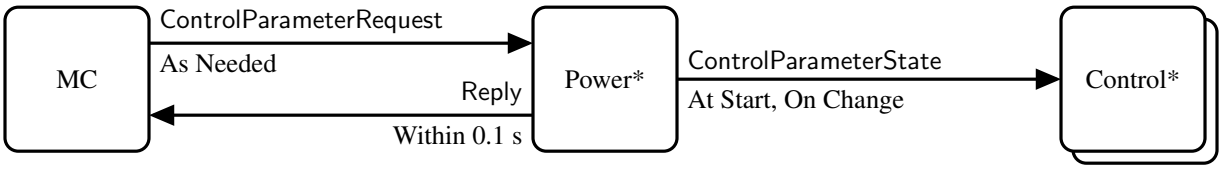


Figure B.23: Device Parameters data flow

Figure B.24 illustrates all the topics in the data flow.

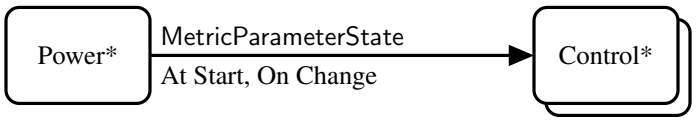


Figure B.24: Device Parameters data flow

Any power device that supports this mechanism shall declare the list of parameters in the Device Announcements, Section B.5.

Table B.151 provides an overview of how each topic is used.

Table B.151: Description of the Device Parameters topics.

Topic	Description
ControlParameterState	The active power device control parameters for this power device.
ControlParameterRequest	Request to change the device control parameters as defined by the DeviceInfo.controlParameters.
Reply	The reply to a request.
MetricParameterState	The active metric parameter values for this power device.

Table B.152 specifies the publishers and subscribers for each topic as well as usage requirements as described in Section A.8.7.

Table B.152: Participants of the Device Parameters topics.

Topic	Usage	MD	MC	MON	SRC	STOR	DIST	LOAD	CONV
ControlParameterState	Conditional				Pub	Pub	Pub	Pub	Pub
		Sub	Sub	Sub					
ControlParameterRequest	Optional		Pub						
				Sub	Sub	Sub	Sub	Sub	Sub
Reply	Required				Pub	Pub	Pub	Pub	Pub
		Sub	Sub	Sub					
MetricParameterState	Optional	Pub	Pub		Pub	Pub	Pub	Pub	Pub
		Sub	Sub	Sub					

This participants table has the following interpretation.

1. ControlParameterState is conditional on HAS_CONTROL_PARAMETERS.
2. ControlParameterState shall be published by SRC, STOR, DIST, LOAD, and CONV devices that have any `DeviceInfo.controlParameters` with non-zero length.
3. ControlParameterState may be subscribed by MD, MC, and MON devices.
4. ControlParameterRequest shall be published by MC devices that have this topic listed in `TopicInfo.optionalTopics`.
5. ControlParameterRequest may be subscribed by MON, SRC, STOR, DIST, LOAD, and CONV devices.
6. Reply shall be published by SRC, STOR, DIST, LOAD, and CONV devices.
7. Reply may be subscribed by MD, MC, and MON devices.
8. MetricParameterState shall be published by MD, MC, SRC, STOR, DIST, LOAD, and CONV devices that have this topic listed in `TopicInfo.optionalTopics`.

9. MetricParameterState may be subscribed by MD, MC, and MON devices.

Table B.153 specifies the timing for when sample updates are sent to each topic. These parameters are defined in Section A.8.5.

Table B.153: Timing of the Device Parameters topics.

Topic	Data Trigger	Rate Class	Burst Size	QoS Profile
ControlParameterState	At Start, On Change	10 s	10	PublishLast
ControlParameterRequest	As Needed	10 s	10	Command
Reply	Within 0.1 s	All	1	Reply
MetricParameterState	At Start, On Change	10 s	10	PublishLast

B.14.2 Data Types

Table B.154 specifies the data types that are used as sample values for each topic.

Table B.154: Data types of the Device Parameters topics.

Topic	Data Type
ControlParameterState	`tms::ControlParameterState`
ControlParameterRequest	`tms::ControlParameterRequest`
Reply	`tms::Reply`
MetricParameterState	`tms::MetricParameterState`

Topic-level data types are defined in the order shown in Table B.154. Then nested data types are defined in the order of appearance. Cross-references are provided for data types that have already been defined.

B.14.2.1 `tms::ControlParameterState`

PURPOSE: State of control parameters.

TOPIC USAGE: ControlParameterState

EXTENSIBILITY: extensibility(APPENDABLE)

PATTERN: Structure

ATTRIBUTES:

Name	Type and Description
`deviceId`	`Identity` The device described by this structure. Annotations: *keyval*
`config`	`ConfigId` Configuration that these values belong to.
`parameters`	`ParameterValueSequence` A full list of the parameters available on this device.

B.14.2.2 `tms::ControlParameterRequest`

PURPOSE: Request to change the device control parameters as defined by the DeviceInfo.controlParameters.

TOPIC USAGE: ControlParameterRequest

EXTENSIBILITY: extensibility(APPENDABLE)

PATTERN: Structure

ATTRIBUTES:

Name	Type and Description
`requestId`	`DeviceConfigRequest` Identity of this request. Annotations: *keyval*
`sequenceId`	`tms::RequestSequence` Request sequence data used to associate a request and returning reply.
`parameters`	`ParameterValueSequence` Selected parameter values to update (may be partial or full list).

DESCRIPTION:

The `parameters` must be validated by the subscribing device before the Reply is published. All the included parameters name/values must pass validation. If a validation error is detected, it should be reported in the Reply.ReplyStatus with an appropriate ReplyStatus.code and ReplyStatus.reason. The ReplyStatus.reason only allows 32 characters so the reason should include the parameter name that failed validation and a short explaination. If the validation fails because a required parameter is missing, report the first missing parameter name.

B.14.2.3 `tms::Reply`

See definition in Section B.2.2.1.

B.14.2.4 `tms::MetricParameterState`

PURPOSE: Latest values for device metric parameters.

TOPIC USAGE: MetricParameterState

EXTENSIBILITY: extensibility(APPENDABLE)

PATTERN: Structure

ATTRIBUTES:

Name	Type and Description
`deviceId`	`Identity` The device described by this structure. Annotations: *keyval*
`metricParameters`	`ParameterValueSequence` A full list of read-only parameter values on this device.

B.14.2.5 `tms::Identity`

See definition in Section B.2.2.2.

B.14.2.6 `tms::ConfigId`

See definition in Section B.2.2.3.

B.14.2.7 `tms::ParameterValueSequence`

PURPOSE: A sequence of ParameterValue.

EXTENSIBILITY: extensibility(FINAL)

PATTERN: Typedef

ORIGINAL TYPE: `sequence<ParameterValue,0,128>`

B.14.2.8 `tms::ParameterValue`

PURPOSE: Provide the value for a parameter.

TOPIC USAGE: Nested

EXTENSIBILITY: extensibility(APPENDABLE)

PATTERN: Structure

ATTRIBUTES:

Name	Type and Description
`name`	
	`String1_32`
	Name of this parameter.
`value`	
	`float32`
	Value of this parameter, with units according to ParameterMetadata.

B.14.2.9 `tms::String1-32`

See definition in Section B.5.2.20.

B.14.2.10 `tms::DeviceConfigRequest`

PURPOSE: Key value for a request that targets one power device configuration.

TOPIC USAGE: Nested

EXTENSIBILITY: extensibility(APPENDABLE)

PATTERN: Structure

ATTRIBUTES:

Name	Type and Description
`requestingDeviceId`	`Identity` Identity of the device that sent this request.
`targetDeviceId`	`Identity` Identity of the device that should receive this request.
`config`	`ConfigId` Power device configuration that this request belongs to.

B.14.2.11 `tms::RequestSequence`

See definition in Section B.2.2.5.

B.14.2.12 `tms::PowerPortNumber`

See definition in Section B.2.2.4.

B.14.2.13 `tms::ReplyStatus`

See definition in Section B.2.2.6.

B.14.2.14 `tms::ReplyCode`

See definition in Section B.2.2.7.

B.14.2.15 `tms::String32`

See definition in Section B.2.2.8.

B.15 Grounding Circuits

Grounding circuits are used to limit the voltage between power lines and the local earth ground. Most power devices have a single grounding circuit and all parts of the device are bonded to this circuit. Power conversion devices may have different grounding circuits for each power port. Most tactical microgrid equipment is designed to be solidly grounded.

Some DC microgrids may be high-resistance grounded.

B.15.1 Data Flows

Figure B.25 illustrates all the topics in the data flow.

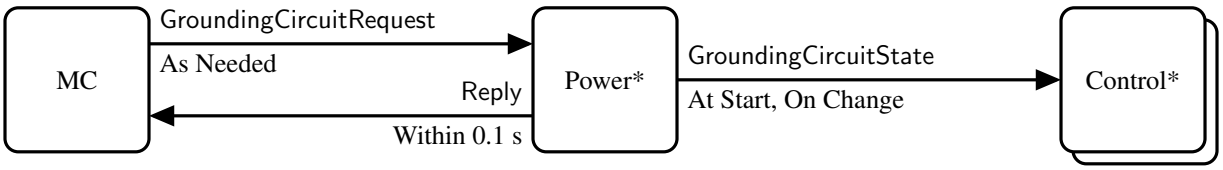


Figure B.25: Grounding Circuits data flow

All TMS power devices should report the status of their grounding circuits. Any power device having a switchable grounding circuit shall report their circuit status.

Table B.160 provides an overview of how each topic is used.

Table B.160: Description of the Grounding Circuits topics.

Topic	Description
GroundingCircuitState	Grounding Circuit states.
GroundingCircuitRequest	Grounding Circuit request.
Reply	The reply to a request.

Table B.161 specifies the publishers and subscribers for each topic as well as usage requirements as described in Section A.8.7.

Table B.161: Participants of the Grounding Circuits topics.

Topic	Usage	MD	MC	MON	SRC	STOR	DIST	LOAD	CONV
GroundingCircuitState	Conditional				Pub	Pub	Pub	Pub	Pub
		Sub	Sub	Sub					
GroundingCircuitRequest	Optional		Pub						
				Sub	Sub	Sub	Sub	Sub	Sub
Reply	Required				Pub	Pub	Pub	Pub	Pub
		Sub	Sub	Sub					

This participants table has the following interpretation.

1. GroundingCircuitState is conditional on HAS_DEVICE_GROUND_PORTS.
2. GroundingCircuitState shall be published by SRC, STOR, DIST, LOAD, and CONV devices that have `PowerDeviceInfo.grounds` with non-zero length.

3. GroundingCircuitState may be subscribed by MD, MC, and MON devices.

4. GroundingCircuitRequest shall be published by MC devices that have this topic listed in `TopicInfo.optionalTopics`.

5. GroundingCircuitRequest may be subscribed by MON, SRC, STOR, DIST, LOAD, and CONV devices.

6. Reply shall be published by SRC, STOR, DIST, LOAD, and CONV devices.

7. Reply may be subscribed by MD, MC, and MON devices.

Table B.162 specifies the timing for when sample updates are sent to each topic. These parameters are defined in Section A.8.5.

Table B.162: Timing of the Grounding Circuits topics.

`Topic`	Data Trigger	Rate Class	Burst Size	QoS Profile
GroundingCircuitState	At Start, On Change	10 s	10	PublishLast
GroundingCircuitRequest	As Needed	10 s	10	Command
Reply	Within 0.1 s	All	1	Reply

B.15.2 Data Types

Table B.163 specifies the data types that are used as sample values for each topic.

Table B.163: Data types of the Grounding Circuits topics.

Topic	`Data Type`
GroundingCircuitState	`tms::GroundingCircuitState`
GroundingCircuitRequest	`tms::GroundingCircuitRequest`
Reply	`tms::Reply`

Topic-level data types are defined in the order shown in Table B.163. Then nested data types are defined in the order of appearance. Cross-references are provided for data types that have already been defined.

B.15.2.1 `tms::GroundingCircuitState`

PURPOSE: Report the present state of all the ground circuits in a device.

TOPIC USAGE: GroundingCircuitState

EXTENSIBILITY: extensibility(APPENDABLE)

PATTERN: Structure

ATTRIBUTES:

Name	Type and Description
`deviceId`	`Identity` The device described by this structure. Annotations: *keyval*
`grounds`	`GroundingCircuitStateSequence` Information on each ground circuit.

B.15.2.2 `tms::GroundingCircuitRequest`

PURPOSE: Remotely transition a grounding circuit switch.

TOPIC USAGE: GroundingCircuitRequest

EXTENSIBILITY: extensibility(APPENDABLE)

PATTERN: Structure

ATTRIBUTES:

Name	Type and Description
`requestId`	`DeviceConfigRequest` Identity of this request. Annotations: *keyval*
`sequenceId`	`tms::RequestSequence` Request sequence data used to associate a request and returning reply.
`groundNumber`	`GroundingCircuitNumber` Grounding circuit containing the switch that should change state.
`control`	`DesiredCircuitContinuity` Desired continuity through the grounding control switch.
`pulse`	`DesiredCircuitContinuity` Desired continuity through the grounding pulse switch.

B.15.2.3 `tms::Reply`

See definition in Section B.2.2.1.

B.15.2.4 `tms::Identity`

See definition in Section B.2.2.2.

B.15.2.5 `tms::GroundingCircuitStateSequence`

PURPOSE: A sequence of Grounding Circuit Status.

EXTENSIBILITY: extensibility(FINAL)

PATTERN: Typedef

ORIGINAL TYPE: `sequence<GroundingCircuitStatus,0,MAX_GROUNDING_CIRCUITS>`

B.15.2.6 `tms::GroundingCircuitStatus`

PURPOSE: Describe the state of a grounding circuit.

TOPIC USAGE: Nested

EXTENSIBILITY: extensibility(APPENDABLE)

PATTERN: Structure

ATTRIBUTES:

Name	Type and Description
`groundNumber`	`GroundingCircuitNumber` Number used to represent this circuit.
`control`	`optional<PowerSwitchStatus>` Status of the grounding control switch.
`pulse`	`optional<PowerSwitchStatus>` Status of the grounding pulse switch.
`faultDetection`	`GroundFaultDetection` Status of the ground fault detector. GFD_NOT_APPLICABLE if no detector.

B.15.2.7 `tms::GroundingCircuitNumber`

See definition in Section B.5.2.49.

B.15.2.8 `tms::PowerSwitchStatus`

PURPOSE: Describe the present state and most recent transition of a power switch

TOPIC USAGE: Nested

EXTENSIBILITY: extensibility(APPENDABLE)

PATTERN: Structure

ATTRIBUTES:

Name	Type and Description
`presentContinuity`	`CircuitContinuity` Present circuit continuity through the switch. May be measured from the switch position or from electrical continuity. Continuity is unknown for manual switches with no monitoring capability. Continuity may be inconsistent for multi-phase switches.
`futureContinuity`	`CircuitContinuity` Continuity that the switch is transitioning to, or presentContinuity if no transition. May be initiated by a manual, automatic, or requested action.
`requestLock`	`boolean` This power switch is known to be locked and requests will get REPLY_PRECONDITION_FAILED. A device-level lock must be reported on all affected ports. Unknown locks may REPLY_OK and then PSTC_MALFUNCTION.
`presentContinuityReason`	`PowerSwitchReason` Type of action that caused the switch to be in its presentContinuity.
`presentContinuityActor`	`optional<Identity>` Device or operator that caused the switch to be in its presentContinuity. Present with an MC identity for REQUEST, operator identity for OPERATOR_LOGIN, and this deviceId for AUTO, AUTO_PROTECTION, AUTO_LOAD_SHED, AUTO_VOLTAGE_LOSS, and AUTO_COMMS_LOSS. Absent for all other PowerSwitchReason values.
`futureContinuityReason`	`PowerSwitchTransitionCode` Progress towards completing a transition, or final result of a requested transition.

B.15.2.9 `tms::CircuitContinuity`

PURPOSE: Indicate whether electricity can flow through a connection.

EXTENSIBILITY: extensibility(APPENDABLE)

PATTERN: Enumeration

VALUES:

Name	Description
`CC_UNKNOWN`	No information is available, either due to design or malfunction.
`CC_OPEN`	Electrical or mechanical sensors indicate that continuity has been broken.
`CC_CLOSED`	Electrical or mechanical sensors indicate that continuity has been established.
`CC_INCONSISTENT`	Electrical or mechanical sensors indicate that continuity has been established on some wires and broken on other wires.

B.15.2.10 `tms::PowerSwitchReason`

PURPOSE: Indicate why a power switch entered a continuity state.

EXTENSIBILITY: extensibility(APPENDABLE)

PATTERN: Enumeration

VALUES:

Name	Description
`PSR_UNKNOWN`	No information is available, either due to design or malfunction.
`PSR_STARTUP`	Initial state when the device started.
`PSR_MANUAL`	Changed by a local operator, either mechanical or digital. PSR_OPERATOR_LOGIN if the operator is authenticated and has an identity.
`PSR_AUTO`	Automatic operation initiated by the device and not covered by a more specific AUTO reason.
`PSR_REQUEST`	Responded to a MC request.
`PSR_OPERATOR_LOGIN`	Changed by a local operator who is logged in to an internal display and has an identity. PSR_MANUAL when not logged in or no operator identity.

PSR_AUTO_PROTECTION
: Activated by an internal protection function. Should report SPN_TMS_TRIP_* for additional details.

PSR_AUTO_LOAD_SHED
: Activated by internal load shed logic.

PSR_AUTO_VOLTAGE_LOSS
: Responded to an interruption in line voltage.

PSR_AUTO_COMMS_LOSS
: Responded to an interruption in microgrid communications.

PSR_MALFUNCTION
: Apparent internal malfunction.

B.15.2.11 tms::PowerSwitchTransitionCode

PURPOSE: Indicate the status of a pending power switch transition. Power switches are not required to support all transition codes.

EXTENSIBILITY: extensibility(APPENDABLE)

PATTERN: Enumeration

VALUES:

Name	Description
PSTC_UNKNOWN	Unknown value. This is a generic placeholder representing the case in which an appropriate value could not be determined.
PSTC_NONE	No transition was initiated.
PSTC_IN_PROGRESS	Transition is in progress and needs time to complete. No pending conditions. For example, IEEE C37.2 device function 62.
PSTC_PENDING_SYNC	Waiting for the synchronizer to close. For example, IEEE C37.2 device function 25.
PSTC_PENDING_SWITCH_CONDITIONS	Waiting for the PowerSwitchCloseConditions to be met.
PSTC_PENDING_INTERLOCK	Waiting for a sequencing condition to be met. For example, IEEE C37.2 device function 3.
PSTC_COMPLETE	The transition completed successfully.
PSTC_MANUAL_OVERRIDE	The last transition request was overridden (followed and replaced) by MANUAL action.

`PSTC_AUTO_OVERRIDE`
: The last transition request was overridden (followed and replaced) by AUTO logic.

`PSTC_AUTO_LOCK`
: The transition was attempted one or more times. Automatic logic locked out further attempts. For example, IEEE C37.2 device function 66. MC may re-try the request.

`PSTC_MALFUNCTION`
: The transition failed for an unknown reason.

B.15.2.12 `tms::GroundFaultDetection`

PURPOSE: Ground fault detection.

EXTENSIBILITY: extensibility(APPENDABLE)

PATTERN: Enumeration

VALUES:

Name	Description
`GFD_UNKNOWN`	Unknown value. This is a generic placeholder representing the case in which an appropriate value could not be determined.
`GFD_NOT_APPLICABLE`	No ground fault detector is present on the device.
`GFD_NONE`	No ground fault detected.
`GFD_ANY`	Ground fault detected. Multiple lines detected or specific line unknown.
`GFD_A`	Ground fault detected on phase A.
`GFD_B`	Ground fault detected on phase B.
`GFD_C`	Ground fault detected on phase C.
`GFD_DCPOS`	Ground fault detected on DC+.
`GFD_DCNEG`	Ground fault detected on DC-.

B.15.2.13 `tms::DeviceConfigRequest`

See definition in Section B.14.2.10.

B.15.2.14 `tms::ConfigId`

See definition in Section B.2.2.3.

B.15.2.15 `tms::RequestSequence`

See definition in Section B.2.2.5.

B.15.2.16 `tms::DesiredCircuitContinuity`

PURPOSE: Indicate the desired circuit continuity of a switch.

EXTENSIBILITY: extensibility(APPENDABLE)

PATTERN: Enumeration

VALUES:

Name	Description
`DCC_UNKNOWN`	Unknown value. This is a generic placeholder representing the case in which an appropriate value could not be determined.
`DCC_OPEN`	Transition to CC_OPEN.
`DCC_CLOSED`	Transition to CC_CLOSED. Predicated on AuthorizationToEnergizeRequest.
`DCC_SYNC_CLOSED`	Transition to CC_CLOSED when synchronization is achieved. May close to a dead circuit, predicated on AuthorizationToEnergizeRequest. May act as DCC_CLOSED_INTERNAL when the internal circuit is dead.
`DCC_CLOSED_INTERNAL`	Transition to CC_CLOSED. One side of the switch must be a circuit enclosed inside a device with protective guards.
`DCC_NO_CHANGE`	No transition. May be used instead of DCC_CLOSED in non-active settings groups.

B.15.2.17 `tms::PowerPortNumber`

See definition in Section B.2.2.4.

B.15.2.18 `tms::ReplyStatus`

See definition in Section B.2.2.6.

B.15.2.19 `tms::ReplyCode`

See definition in Section B.2.2.7.

B.15.2.20 `tms::String32`

See definition in Section B.2.2.8.

B.16 Source and Storage Start/Stop

Energy resources (SRC and STOR device roles) may take a long time to start and consume significant power when on standby, so the MC may request them to operate at one of the six levels defined in Table B.173. The SRC or STOR device controller and associated power supply and thermal regulation are not disabled by this request. All devices may be manually started and stopped by the operator, separate from any requests in the data model.

Table B.173: Definition of the `StartStopLevels`.

Operating Level	Temperature	Energy	Power	Synced	Switch
OFF	Ambient	Off	Off	No	Open
WARM	Controlled	Off	Off	No	Open
IDLE	Controlled	Idling	Off	No	Open
READY	Controlled	Rated	On	No	Open
READY_SYNCED	Controlled	Rated	On	Yes	Open
OPERATIONAL	Controlled	Rated	On	Yes	Closed

The entries in Table B.173 have the following interpretations.

1. Temperature. Describes the component that provides thermal regulation for the energy and power components. *Ambient* means it is disabled and *controlled* means it is enabled.

2. Energy. Describes the component that produces or stores energy. *Off* means it is has been disabled, *idling* means it is enabled at a low power state and not ready to operate at rated capacity, and *rated* means it is fully functional.

3. Power. Describes the component that provides power. *Off* means it is disabled and *on* means it is enabled and ready to provide power.

4. Synced. Describes the synchronization of voltage, frequency, and phase between the internal bus and the external connector. *No* means it is not maintained and *yes* means it is maintained.

5. Switch. Describes the circuit continuity of the power port switch.

SRC and STOR devices may implement a subset of these levels, as indicated in their respective `SourceInfo` and `StorageInfo`. SRC and STOR devices use the StartStopRequest to control their power switch instead of the usual PowerSwitchRequest (Section B.17.1).

Examples of more specific interpretations for these operating levels are given in Table B.174 and Table B.175.

Table B.174: Example operating levels for a diesel genset SRC.

Operating Level	Engine	Generator	Sync	Switch
OFF	Stopped	Off	No	Open
WARM	Stopped with block heater	Off	No	Open
IDLE	Running at low speed	Off	No	Open
READY	Running at rated speed	Energized	No	Open
READY_SYNCED	Running at rated speed	Energized	Yes	Open
OPERATIONAL	Running at rated speed	Energized	Yes	Closed

Table B.175: Example operating levels for a battery STOR.

Operating Level	Energy Storage	Power Conv.	Sync	Switch
OFF	Ambient temperature	Off	No	Open
WARM	Temperature controlled	Off	No	Open
IDLE	Not implemented			
READY	Temperature controlled	Energized	No	Open
READY_SYNCED	Temperature controlled	Energized	Yes	Open
OPERATIONAL	Temperature controlled	Energized	Yes	Closed

The WARM and IDLE levels are provided to support states that can start faster than OFF but consume less fuel or other sources of power than READY. The READY_SYNC level is provided to support devices that can maintain synchronization even when the power switch is open, thus starting slightly faster than READY.

When a power device receives an EnergyStartStopRequest, it may need to transition through one or more intermediate levels before reaching the desired level. It may also be able to skip over intermediate levels that are unnecessary given present operating conditions. Any such intermediate transitions have the following requirements.

1. Intermediate transitions shall be sequenced automatically by the power device, without requiring the MC to request them individually.
2. Intermediate transitions shall only include levels required for the nominal health and operation of the power device.
3. Intermediate transitions shall not include unnecessary delays between levels.

B.16.1 Data Flows

Figure B.26 illustrates all the topics in the data flow.

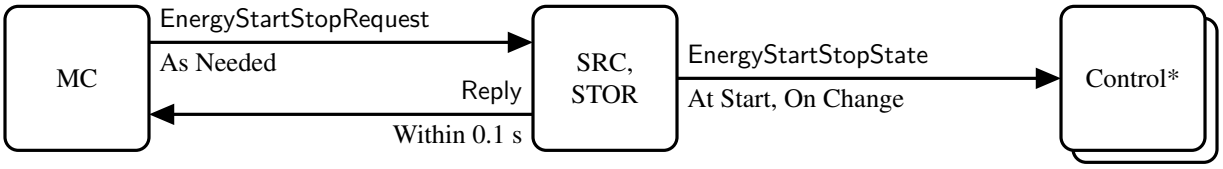


Figure B.26: Source and Storage Start/Stop data flow

Table B.176 provides an overview of how each topic is used.

Table B.176: Description of the Source and Storage Start/Stop topics.

Topic	Description
EnergyStartStopState	Reports the operational state of a Source or Storage device
EnergyStartStopRequest	Request to change the EnergyStartStopLevel of a Source or Storage device.
Reply	The reply to a request.

Table B.177 specifies the publishers and subscribers for each topic as well as usage requirements as described in Section A.8.7.

Table B.177: Participants of the Source and Storage Start/Stop topics.

Topic	Usage	MD	MC	MON	SRC	STOR	DIST	LOAD	CONV
EnergyStartStopState	Required				Pub	Pub			
		Sub	Sub	Sub					
EnergyStartStopRequest	Optional		Pub						
				Sub	Sub	Sub			
Reply	Required				Pub	Pub			
		Sub	Sub	Sub					

This participants table has the following interpretation.

1. EnergyStartStopState shall be published by SRC and STOR devices.
2. EnergyStartStopState may be subscribed by MD, MC, and MON devices.
3. EnergyStartStopRequest shall be published by MC devices that have this topic listed in `TopicInfo.optionalTopics`.
4. EnergyStartStopRequest may be subscribed by MON, SRC, and STOR devices.
5. Reply shall be published by SRC and STOR devices.
6. Reply may be subscribed by MD, MC, and MON devices.

Table B.178 specifies the timing for when sample updates are sent to each topic. These parameters are defined in Section A.8.5.

Table B.178: Timing of the Source and Storage Start/Stop topics.

`Topic`	Data Trigger	Rate Class	Burst Size	QoS Profile
EnergyStartStopState	At Start, On Change	10 s	10	PublishLast
EnergyStartStopRequest	As Needed	10 s	1	Command
Reply	Within 0.1 s	All	1	Reply

B.16.2 Data Types

Table B.179 specifies the data types that are used as sample values for each topic.

Table B.179: Data types of the Source and Storage Start/Stop topics.

Topic	Data Type
EnergyStartStopState	`tms::EnergyStartStopState`
EnergyStartStopRequest	`tms::EnergyStartStopRequest`
Reply	`tms::Reply`

Topic-level data types are defined in the order shown in Table B.179. Then nested data types are defined in the order of appearance. Cross-references are provided for data types that have already been defined.

B.16.2.1 `tms::EnergyStartStopState`

PURPOSE: Report the operating level of a SRC and STOR device, including the progress of any transitions.

TOPIC USAGE: EnergyStartStopState

EXTENSIBILITY: extensibility(APPENDABLE)

PATTERN: Structure

ATTRIBUTES:

Name	Type and Description
`deviceId`	`Identity` The device described by this structure. Annotations: *keyval*
`presentLevel`	`EnergyStartStopLevel` Present operating level.
`futureLevel`	`EnergyStartStopLevel` Level that the device is transitioning to, or presentLevel if no transition.
`requestLock`	`boolean` This device or power switch is known to be locked and requests will get REPLY_PRECONDITION_FAILED. A device-level lock must be reported on all affected ports. Unknown locks may REPLY_OK and then PSTC_MALFUNCTION.
`elapsedTime`	`float32` Elapsed time the device has been in the present transition. Remains at the total elapsed time when the transition is complete.

Annotations: *units*=second (s)

`remainingTime`
: `float32`
Estimates when the present transition will complete. It is zero (0.0) when the device is not in transition.
Annotations: *units*=second (s)

`presentLevelReason`
: `PowerSwitchReason`
Type of action that caused the device to be in its presentLevel.

`presentLevelActor`
: `optional<Identity>`
Device or user that caused the device to be in its presentLevel. Value may be a MC (REQUEST), this device (AUTO), or operator (MANUAL). Present when the identity is known, otherwise absent.

`futureLevelReason`
: `PowerSwitchTransitionCode`
Progress towards completing a transition, or final result of a requested transition.

B.16.2.2 `tms::EnergyStartStopRequest`

PURPOSE: Change the level of a SRC or STOR device.

TOPIC USAGE: EnergyStartStopRequest

EXTENSIBILITY: extensibility(APPENDABLE)

PATTERN: Structure

ATTRIBUTES:

Name	Type and Description
`requestId`	`DeviceConfigRequest` Identity of this request. Annotations: *keyval*
`sequenceId`	`RequestSequence` Request sequence data used to associate a request and returning reply.
`fromLevel`	`EnergyStartStopLevel` Reject this request if fromLevel does not match presentLevel. ESSL_ANY matches all levels. ESSL_UNKNOWN is invalid.
`toLevel`	`EnergyStartStopLevel` Desired operating level. ESSL_ANY and ESSL_UNKNOWN are invalid.

`switchConditions`
: `optional<PowerSwitchCloseConditions>`
Conditions that may delay switch closure. Only applies to ESSL_OPERATIONAL requests and power ports that support PSF_SWITCH_CONDITIONS.

B.16.2.3 `tms::Reply`

See definition in Section B.2.2.1.

B.16.2.4 `tms::Identity`

See definition in Section B.2.2.2.

B.16.2.5 `tms::EnergyStartStopLevel`

See definition in Section B.5.2.76.

B.16.2.6 `tms::PowerSwitchReason`

See definition in Section B.15.2.10.

B.16.2.7 `tms::PowerSwitchTransitionCode`

See definition in Section B.15.2.11.

B.16.2.8 `tms::DeviceConfigRequest`

See definition in Section B.14.2.10.

B.16.2.9 `tms::ConfigId`

See definition in Section B.2.2.3.

B.16.2.10 `tms::RequestSequence`

See definition in Section B.2.2.5.

B.16.2.11 `tms::PowerSwitchCloseConditions`

PURPOSE: The associated request to close a switch should wait until all of these conditions are met. The MC may use these to override any built-in device defaults, providing looser bounds to recover from abnormal conditions or tighter bounds to prevent a large transient. These conditions do not modify any built-in protections such as a synchronization check. Requires PSF_SWITCH_CONDITIONS in PowerSwitchInfo.features

TOPIC USAGE: Nested

EXTENSIBILITY: extensibility(APPENDABLE)

PATTERN: Structure

ATTRIBUTES:

Name	Type and Description
`minVoltage`	`float32` Minimum external RMS AC or DC voltage. Annotations: *units*=volt (V)
`maxVoltage`	`float32` Maximum external RMS AC or DC voltage. Annotations: *units*=volt (V)
`minFrequency`	`float32` Minimum external AC frequency. NaN for DC. Annotations: *units*=hertz (Hz)
`maxFrequency`	`float32` Maximum external AC frequency. NaN for DC. Annotations: *units*=hertz (Hz)

B.16.2.12 `tms::PowerPortNumber`

See definition in Section B.2.2.4.

B.16.2.13 `tms::ReplyStatus`

See definition in Section B.2.2.6.

B.16.2.14 `tms::ReplyCode`

See definition in Section B.2.2.7.

B.16.2.15 `tms::String32`

See definition in Section B.2.2.8.

B.17 Power Switches and Connectors

All power devices report the status of their power switches and connectors. All power devices inventory their power switches and connectors in the Device Announcements, Section B.5.

B.17.1 Data Flows

Figure B.27 illustrates all the topics in the data flow.

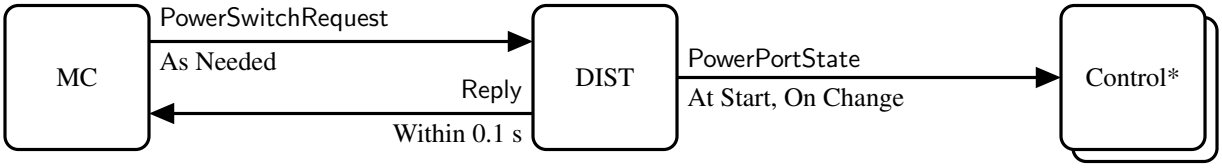


Figure B.27: Power Switches and Connectors data flow

Table B.183 provides an overview of how each topic is used.

Table B.183: Description of the Power Switches and Connectors topics.

Topic	Description
PowerPortState	Reports dynamic state changes for a power device ports.
PowerSwitchRequest	Request to change the circuit continuity of a single power port
Reply	The reply to a request.

Table B.184 specifies the publishers and subscribers for each topic as well as usage requirements as described in Section A.8.7.

Table B.184: Participants of the Power Switches and Connectors topics.

Topic	Usage	MD	MC	MON	SRC	STOR	DIST	LOAD	CONV
PowerPortState	Required				Pub	Pub	Pub	Pub	Pub
		Sub	Sub	Sub					
PowerSwitchRequest	Optional		Pub						
				Sub			Sub	Sub	Sub
Reply	Required						Pub	Pub	Pub
		Sub	Sub	Sub					

This participants table has the following interpretation.

1. PowerPortState shall be published by SRC, STOR, DIST, LOAD, and CONV devices.
2. PowerPortState may be subscribed by MD, MC, and MON devices.

3. PowerSwitchRequest shall be published by MC devices that have this topic listed in `TopicInfo.optionalTopics`.

4. PowerSwitchRequest may be subscribed by MON, DIST, LOAD, and CONV devices.

5. Reply shall be published by DIST, LOAD, and CONV devices.

6. Reply may be subscribed by MD, MC, and MON devices.

Any device that has remotely switchable ports shall implement the PowerSwitchRequest.

Table B.185 specifies the timing for when sample updates are sent to each topic. These parameters are defined in Section A.8.5.

Table B.185: Timing of the Power Switches and Connectors topics.

`Topic`	Data Trigger	Rate Class	Burst Size	QoS Profile
PowerPortState	At Start, On Change	1 s	1	PublishLast
PowerSwitchRequest	As Needed	10 s	10	Command
Reply	Within 0.1 s	All	1	Reply

B.17.2 Data Types

Table B.186 specifies the data types that are used as sample values for each topic.

Table B.186: Data types of the Power Switches and Connectors topics.

Topic	`Data Type`
PowerPortState	`tms::PowerPortState`
PowerSwitchRequest	`tms::PowerSwitchRequest`
Reply	`tms::Reply`

Topic-level data types are defined in the order shown in Table B.186. Then nested data types are defined in the order of appearance. Cross-references are provided for data types that have already been defined.

B.17.2.1 `tms::PowerPortState`

PURPOSE: Report the present state of all the power ports in a device.

TOPIC USAGE: PowerPortState

EXTENSIBILITY: extensibility(APPENDABLE)

PATTERN: Structure

ATTRIBUTES:

Name	Type and Description
`deviceId`	`Identity` The device described by this structure. Annotations: *keyval*
`powerPorts`	`PowerPortStatusSequence` Information on each power port.

B.17.2.2 `tms::PowerSwitchRequest`

PURPOSE: Remotely transition a power port switch.

TOPIC USAGE: PowerSwitchRequest

EXTENSIBILITY: extensibility(APPENDABLE)

PATTERN: Structure

ATTRIBUTES:

Name	Type and Description
`requestId`	`PowerPortConfigRequest` Identity of this request. Annotations: *keyval*
`sequenceId`	`tms::RequestSequence` Request sequence data used to associate a request and returning reply.
`continuity`	`DesiredCircuitContinuity` Desired continuity through the switch.
`switchConditions`	`optional<PowerSwitchCloseConditions>` Conditions that may delay switch closure. Can only be present for power ports that support PSF_SWITCH_CONDITIONS. Must be absent for DCC_OPEN or DCC_NO_CHANGE requests.

B.17.2.3 `tms::Reply`

See definition in Section B.2.2.1.

B.17.2.4 `tms::Identity`

See definition in Section B.2.2.2.

B.17.2.5 `tms::PowerPortStatusSequence`

PURPOSE: A sequence of PowerPortStatus.

EXTENSIBILITY: extensibility(FINAL)

PATTERN: Typedef

ORIGINAL TYPE: `sequence<PowerPortStatus,0,MAX_PORTS>`

B.17.2.6 `tms::PowerPortStatus`

PURPOSE: Describe the state of a power port.

TOPIC USAGE: Nested

EXTENSIBILITY: extensibility(APPENDABLE)

PATTERN: Structure

ATTRIBUTES:

Name	Type and Description
`portNumber`	`PowerPortNumber` Number shown on the device exterior to represent this port.
`cableStatus`	`CableSenseStatus` Indicates whether a power cable is plugged in.
`cableId`	`optional<Identity>` Indicates that a smart cable is attached. This value is present when a smart cable is connected and absent when it is disconnected.
`connection`	`optional<PowerPortId>` Indicates that this port is connected to a remote port.
`switchStatus`	`optional<PowerSwitchStatus>` Indicates the state of a switch associated with this port. Must be present when PowerPortInfo.hasSwitch is true and absent otherwise.

B.17.2.7 `tms::PowerPortNumber`

See definition in Section B.2.2.4.

B.17.2.8 `tms::CableSenseStatus`

PURPOSE: Indicate whether a cable has been sensed in a power port.

EXTENSIBILITY: extensibility(APPENDABLE)

PATTERN: Enumeration

VALUES:

Name	Description
`CS_UNKNOWN`	Cable presence is neither confirmed nor denied through active detection. Cable sense is not available, either due to device design or failure.
`CS_DISCONNECTED`	Sensing indicates that a cable is not connected. CST_MEASUREMENT devices must use CS_UNKNOWN instead of this value.
`CS_CONNECTED`	Sensing indicates that a cable is connected.

B.17.2.9 `tms::PowerPortId`

PURPOSE: Identify a power port (often remote)

TOPIC USAGE: Nested

EXTENSIBILITY: extensibility(APPENDABLE)

PATTERN: Structure

ATTRIBUTES:

Name	Type and Description
`deviceId`	`Identity` Device containing the port.
`portNumber`	`PowerPortNumber` Number shown on the device exterior to represent this port.

B.17.2.10 `tms::PowerSwitchStatus`

See definition in Section B.15.2.8.

B.17.2.11 `tms::CircuitContinuity`

See definition in Section B.15.2.9.

B.17.2.12 `tms::PowerSwitchReason`

See definition in Section B.15.2.10.

B.17.2.13 `tms::PowerSwitchTransitionCode`

See definition in Section B.15.2.11.

B.17.2.14 `tms::PowerPortConfigRequest`

PURPOSE: Key value for a request that targets one power port of a device.

TOPIC USAGE: Nested

EXTENSIBILITY: extensibility(APPENDABLE)

PATTERN: Structure

ATTRIBUTES:

Name	Type and Description
`requestingDeviceId`	`Identity` Identity of the device that sent this request.
`targetDeviceId`	`Identity` Identity of the device that should received this request.
`config`	`ConfigId` Power device configuration that this request belongs to.
`portNumber`	`PowerPortNumber` Power port number that should change state.

B.17.2.15 `tms::ConfigId`

See definition in Section B.2.2.3.

B.17.2.16 `tms::RequestSequence`

See definition in Section B.2.2.5.

B.17.2.17 `tms::DesiredCircuitContinuity`

See definition in Section B.15.2.16.

B.17.2.18 `tms::PowerSwitchCloseConditions`

See definition in Section B.16.2.11.

B.17.2.19 `tms::ReplyStatus`

See definition in Section B.2.2.6.

B.17.2.20 `tms::ReplyCode`

See definition in Section B.2.2.7.

B.17.2.21 `tms::String32`

See definition in Section B.2.2.8.

B.18 AC Measurements

AC measurements provide monitoring for alternating current power ports. These measurements may be used to monitor loads and power flows, balance sources, avoid thermal constraints before protections trip, and detect issues such as poor phase balance or grounding. Protections such as over-current and synchronization should rely on measurements internal to the device, not external communications.

The sign conventions are specified in Section A.9.5.

B.18.1 Data Flows

AC measurements are supported by two data flows. The AcMeasurementUpdate provides detailed information at a lower update rate suitable for most monitoring activities. The AcSummaryMeasurementUpdate provides summarized information at a higher update rate suitable for most power balance activities.

Figure B.28 illustrates all the topics in the data flow.

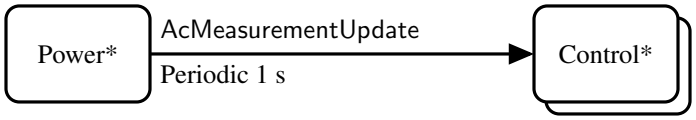


Figure B.28: AC Measurements data flow

Figure B.29 illustrates all the topics in the data flow.

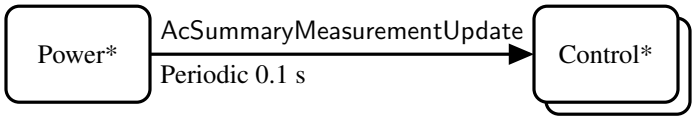


Figure B.29: AC Measurements data flow

Table B.193 provides an overview of how each topic is used.

Table B.193: Description of the AC Measurements topics.

Topic	Description
AcMeasurementUpdate	Reports power measurements for alternating current power device ports.
AcSummaryMeasurementUpdate	Summarized power measurements for AC. Fast updates useful for controlling power balance.

Table B.194 specifies the publishers and subscribers for each topic as well as usage requirements as described in Section A.8.7.

Table B.194: Participants of the AC Measurements topics.

Topic	Usage	MD	MC	MON	SRC	STOR	DIST	LOAD	CONV
AcMeasurementUpdate	Conditional	Sub	Sub	Sub	Pub	Pub	Pub	Pub	Pub
AcSummaryMeasurementUpdate	Conditional		Sub	Sub	Pub	Pub	Pub	Pub	Pub

This participants table has the following interpretation.

1. AcMeasurementUpdate is conditional on HAS_AC_METERS.
2. AcMeasurementUpdate shall be published by SRC, STOR, DIST, LOAD, and CONV devices that have any `PowerDeviceInfo.powerPorts` with at least one `PowerPortInfo.wiring` of AC type and `hasExternalMeter` or `hasInternalMeter` set true.
3. AcMeasurementUpdate may be subscribed by MD, MC, and MON devices.
4. AcSummaryMeasurementUpdate is conditional on HAS_AC_SUMMARY_METERS.
5. AcSummaryMeasurementUpdate shall be published by SRC, STOR, DIST, LOAD, and CONV devices that have any `PowerDeviceInfo.powerPorts` with at least one `PowerPortInfo.wiring` of AC type and `hasSummaryMeasurementUpdate` set true.
6. AcSummaryMeasurementUpdate may be subscribed by MC and MON devices.

Table B.195 specifies the timing for when sample updates are sent to each topic. These parameters are defined in Section A.8.5.

Table B.195: Timing of the AC Measurements topics.

Topic	Data Trigger	Rate Class	Burst Size	QoS Profile
AcMeasurementUpdate	Periodic 1 s	1 s	1	Medium
AcSummaryMeasurementUpdate	Periodic 0.1 s	0.1 s	1	Continuous

B.18.2 Data Types

Table B.196 specifies the data types that are used as sample values for each topic.

Table B.196: Data types of the AC Measurements topics.

Topic	Data Type
AcMeasurementUpdate	`tms::ac::MeasurementUpdate`
AcSummaryMeasurementUpdate	`tms::ac::MeasurementUpdate`

Topic-level data types are defined in the order shown in Table B.196. Then nested data types are defined in the order of appearance. Cross-references are provided for data types that have already been defined.

B.18.2.1 `tms::ac::MeasurementUpdate`

PURPOSE: Publish measurements for all alternating current (AC) power ports on a device.

TOPIC USAGE: AcMeasurementUpdate, AcSummaryMeasurementUpdate

EXTENSIBILITY: extensibility(APPENDABLE)

PATTERN: Structure

ATTRIBUTES:

Name	Type and Description
`deviceId`	`tms::Identity` The device described by this structure. Annotations: *keyval*
`timeMeasured`	`tms::ClockMonotonic` Time of these measurements.
`externalMeasurement`	`PowerPortMeasurementSequence` Measurements for all power ports where PowerPortInfo.hasExternalMeter is true.
`internalMeasurement`	`PowerPortMeasurementSequence` Measurements for all power ports where PowerPortInfo.hasInternalMeter is true.

DESCRIPTION:

Different values are used for each measurement topic, as described below.

The AcMeasurementUpdate topic uses both the `externalMeasurement` and `internalMeasurement` attributes. Also, the `tms::ac::PowerPortMeasurement` contains separate values for each individual line.

The AcSummaryMeasurementUpdate topic only uses the `externalMeasurement` for power ports that set `tms::PowerPortInfo.hasSummaryMeasurementUpdate` to true. This topic always has a zero-length `internalMeasurement` value. Also, the `tms::ac::PowerPortMeasurement` contains a single value summarized across all the lines.

If a device does not have direct external measurement capabilities, it is possible to use internal measurement values and other supporting data to publish deterministically derived data in accordance with Section A.9.10.1.

For example, consider the power port shown in Figure B.5 and suppose the device has an internal meter and can observe the switch state but has no external meter.

When the switch is closed, the internal and external lines are directly connected as a single electrical bus, so the two measurement locations provide equivalent values. Thus the device could justifiably report external metering data using values obtained from the internal meter.

When the switch is open, the internal and external lines are disconnected and form separate electrical buses, so the two measurement locations provide different values. The external voltage cannot be known using directly measured data and so the voltage and frequency are thus set to the default value, NaN. The external current is known to not be flowing through the open switch and so the amperage, phase offset, and power are thus set to zero.

B.18.2.2 `tms::Identity`

See definition in Section B.2.2.2.

B.18.2.3 `tms::ClockMonotonic`

See definition in Section B.8.2.3.

B.18.2.4 `tms::ac::PowerPortMeasurementSequence`

PURPOSE: A sequence of ac::PowerPortMeasurement.

EXTENSIBILITY: extensibility(APPENDABLE)

PATTERN: Typedef

ORIGINAL TYPE: `sequence<PowerPortMeasurement,0,tms::MAX_PORTS>`

B.18.2.5 `tms::ac::PowerPortMeasurement`

PURPOSE: Power measurement for an alternating current port.

TOPIC USAGE: Nested

EXTENSIBILITY: extensibility(APPENDABLE)

PATTERN: Structure

ATTRIBUTES:

Name	Type and Description
`portNumber`	`tms::PowerPortNumber` Port being measured.
`line`	`PowerLineMeasurementSequence` Measured values for each AC power line. Length and order of values are based on PowerPortInfo.wiring. WIRING_SINGLE=1, WIRING_SPLIT=2, WIRING_3WYE=4, and WIRING_3DELTA=3.
`phaseAngle`	`PhaseAngleSequence` Measured phase difference between the voltages for all applicable wiring pairs. Length and order of values are based on PowerPortInfo.wiring. Annotations: *units*=radian (rad)
`realPowerRateOfChange`	`float32` Time derivative of realPower, summed across all wires excluding neutral. Annotations: *units*=watt per second (W/s)
`reactivePowerRateOfChange`	`float32` Time derivative of reactivePower, summed across all wires excluding neutral. Annotations: *units*=volt ampere reactive per second (var/s)

DESCRIPTION:

Different values are used for the AcMeasurementUpdate and AcSummaryMeasurementUpdate topics, as described below.

The AcMeasurementUpdate topic uses separate values for each line in the power port. The order of values in `line` and `phaseAngle` must match the `tms::CircuitWiring` type reported in `tms::PowerPortInfo.wiring`, as specified in Table B.199, Table B.200, Table B.201, and Table B.202. The V_{NG} measurements are referenced to the protective chassis ground.

Values at the end of the `line` and `phaseAngle` sequences may be omitted when the device is not constructed to measure them. The number of values shall be constant and not change from sample to sample. Any "skipped" values before the end of the sequences shall be filled with NaN, so that later values in the sequence are not mis-interpreted.

Table B.199: AcMeasurementUpdate line order for `WIRING_AC_SINGLE`.

Voltage	Current	Angle
V	I	(zero length)

Table B.200: AcMeasurementUpdate line order for `WIRING_AC_SPLIT`.

Voltage	Current	Angle
V_{1N}	I_1	$\angle 12$
V_{2N}	I_2	
V_{NG}	I_N	

Table B.201: AcMeasurementUpdate line order for `WIRING_AC_3DELTA`.

Voltage	Current	Angle
V_{AB}	I_A	$\angle AB$
V_{BC}	I_B	$\angle BC$
V_{CA}	I_C	$\angle CA$

Table B.202: AcMeasurementUpdate line order for `WIRING_AC_3WYE`.

Voltage	Current	Angle
V_{AN}	I_A	$\angle AB$
V_{BN}	I_B	$\angle BC$
V_{CN}	I_C	$\angle CA$
V_{NG}	I_N	

Figure B.30 illustrates the phase angles for a 208/120V wye system.

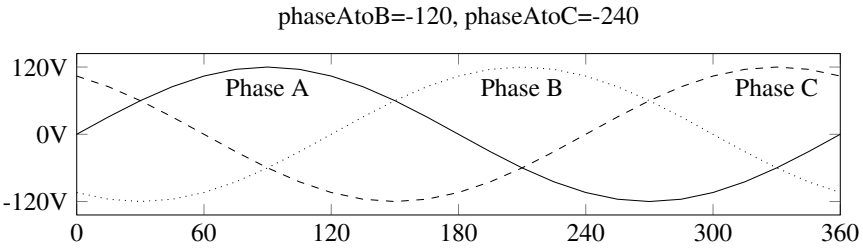


Figure B.30: Phase Angles for a 208/120V Wye System

The AcSummaryMeasurementUpdate topic uses a single value to summarize all lines in the power port. There is one value in `line` and no values in `phaseAngle`, as specified in Table B.203. The calculations used to summarize the lines are specified in Table B.204.

Table B.203: AcSummaryMeasurementUpdate line order.

Voltage	Current	Angle
$V_{summary}$	$I_{summary}$	(zero length)

Table B.204: `PowerLineMeasurement` values for AcSummaryMeasurementUpdate.

Attribute	Summary Value
`voltage`	Average across lines, excluding neutral to ground.
`frequency`	Average across lines, excluding neutral.
`amperage`	Sum across lines, excluding neutral.
`phaseOffset`	Average across lines, excluding neutral.
`realPower`	Sum across lines, excluding neutral.
`reactivePower`	Sum across lines, excluding neutral.

B.18.2.6 `tms::PowerPortNumber`

See definition in Section B.2.2.4.

B.18.2.7 `tms::ac::PowerLineMeasurementSequence`

PURPOSE: A sequence of PowerLineMeasurement. Supports the AC circuits defined in PowerConnectorPhases.

EXTENSIBILITY: extensibility(APPENDABLE)

PATTERN: Typedef

ORIGINAL TYPE: `sequence<PowerLineMeasurement,1,4>`

DESCRIPTION:

See `tms::ac::PowerPortMeasurement` for the order of values in this sequence.

B.18.2.8 `tms::ac::PowerLineMeasurement`

PURPOSE: Measurements for a single AC power line.

TOPIC USAGE: Nested

EXTENSIBILITY: extensibility(APPENDABLE)

PATTERN: Structure

ATTRIBUTES:

Name	Type and Description
`voltage`	`float32` RMS voltage. Annotations: *units*=volt (V)
`frequency`	`float32` Dominant line frequency. Annotations: *units*=hertz (Hz) Change History: 1. *Version:* 2.0.1, *Change:* modify, *Interoperability:* No-xtypes *Description:* Removed minimum range specification.
`amperage`	`float32` RMS amperage. Annotations: *units*=ampere (A)
`phaseOffset`	`optional<float32>` The angle (phase offset) by which the current lags the voltage. Annotations: *min*=-RADIAN_MAX, *max*=RADIAN_MAX, *units*=radian (rad) Change History: 1. *Version:* 2.0.1, *Change:* modify, *Interoperability:* No-xtypes *Description:* Made the field optional.
`realPower`	`float32` Active RMS power. Annotations: *units*=watt (W)
`reactivePower`	`float32` Reactive RMS power. Annotations: *units*=volt ampere reactive (var)

DESCRIPTION:

These measurements represent a single line (power, neutral, or summary) in a power port. Different values are used for each measurement topic, as described in `tms::ac::PowerPortMeasurement`.

These values should be time-averaged over regular intervals so that each update contains new information with no gaps in coverage between updates. The AcMeasurementUpdate values are averaged over 1 s intervals, and the AcSummaryMeasurementUpdate values are over 0.1 s intervals.

Some of these values are usually very similar across lines. Separate values for each line enable the detection of load imbalances, poor connections, and other power issues. Measurements of the neutral line may be compared across different devices to detect certain grounding issues.

Some of these values can be calculated from each other using assumptions about how the microgrid is constructed and operating. The "redundant" values may better reflect the actual instrumentation, provide better performance during changes, or be used to detect faults common in distributed systems.

Under normal synchronous operation, the frequency on all lines will be common throughout the grid. A frequency drift from one location to another may indicate grid partitioning and help identify which devices are in each grid segment. Frequency and voltage measurements can also be used to bring segments close enough together that a synchronizer can join them. A divergence in frequencies between the lines at a single location may indicate inverter regulation problems or another fault.

Values that are not measured, either due to design or malfunction, are filled with NaN. Default, nominal, previous, and projected values must not be used.

B.18.2.9 `tms::ac::PhaseAngleSequence`

PURPOSE: A sequence of floats representing phase angles.

EXTENSIBILITY: extensibility(FINAL)

PATTERN: Typedef

ORIGINAL TYPE: `sequence<float32,0,3>`

DESCRIPTION:

See `tms::ac::PowerPortMeasurement` for the order of values in this sequence.

B.19 DC Measurements

DC measurements provide monitoring for direct current power ports. These measurements may be used to monitor loads and power flows, balance sources, avoid thermal constraints before protections trip, and detect issues such as poor phase balance or grounding. Protections such as over-current and synchronization should rely on measurements internal to the device, not external communications.

The sign conventions are specified in Section A.9.5.

B.19.1 Data Flows

DC measurements are supported by two data flows. The DcMeasurementUpdate provides detailed information at a lower update rate suitable for most monitoring activities. The DcSummaryMeasurementUpdate provides summarized information at a higher update rate suitable for most power balance activities.

Figure B.31 illustrates all the topics in the data flow.

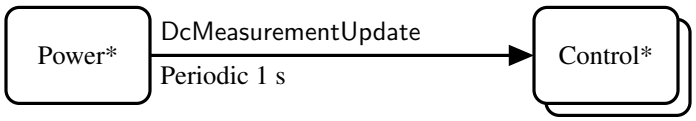


Figure B.31: DC Measurements data flow

Figure B.32 illustrates all the topics in the data flow.

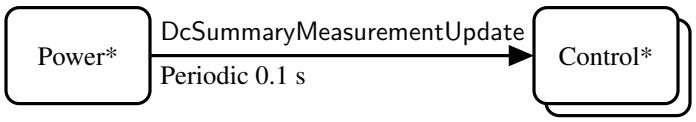


Figure B.32: DC Measurements data flow

Table B.206 provides an overview of how each topic is used.

Table B.206: Description of the DC Measurements topics.

Topic	Description
DcMeasurementUpdate	Reports power measurements for direct current power device ports.
DcSummaryMeasurementUpdate	Summarized power measurements for DC. Fast updates useful for controlling power balance.

Table B.207 specifies the publishers and subscribers for each topic as well as usage requirements as described in Section A.8.7.

Table B.207: Participants of the DC Measurements topics.

Topic	Usage	MD	MC	MON	SRC	STOR	DIST	LOAD	CONV
DcMeasurementUpdate	Conditional	Sub	Sub	Sub	Pub	Pub	Pub	Pub	Pub
DcSummaryMeasurementUpdate	Conditional		Sub	Sub	Pub	Pub	Pub	Pub	Pub

This participants table has the following interpretation.

1. DcMeasurementUpdate is conditional on HAS_DC_METERS.
2. DcMeasurementUpdate shall be published by SRC, STOR, DIST, LOAD, and CONV devices that have any `PowerDeviceInfo.powerPorts` with at least one `PowerPortInfo.wiring` of DC type and `hasExternalMeter` or `hasInternalMeter` set true.
3. DcMeasurementUpdate may be subscribed by MD, MC, and MON devices.
4. DcSummaryMeasurementUpdate is conditional on HAS_DC_SUMMARY_METERS.
5. DcSummaryMeasurementUpdate shall be published by SRC, STOR, DIST, LOAD, and CONV devices that have any `PowerDeviceInfo.powerPorts` with at least one `PowerPortInfo.wiring` of DC type and `hasSummaryMeasurementUpdate` set true.
6. DcSummaryMeasurementUpdate may be subscribed by MC and MON devices.

Table B.208 specifies the timing for when sample updates are sent to each topic. These parameters are defined in Section A.8.5.

Table B.208: Timing of the DC Measurements topics.

Topic	Data Trigger	Rate Class	Burst Size	QoS Profile
DcMeasurementUpdate	Periodic 1 s	1 s	1	Medium
DcSummaryMeasurementUpdate	Periodic 0.1 s	0.1 s	1	Continuous

B.19.2 Data Types

Table B.209 specifies the data types that are used as sample values for each topic.

Table B.209: Data types of the DC Measurements topics.

Topic	Data Type
DcMeasurementUpdate	`tms::dc::MeasurementUpdate`
DcSummaryMeasurementUpdate	`tms::dc::MeasurementUpdate`

Topic-level data types are defined in the order shown in Table B.209. Then nested data types are defined in the order of appearance. Cross-references are provided for data types that have already been defined.

B.19.2.1 `tms::dc::MeasurementUpdate`

PURPOSE: Publish measurements for all direct current (DC) power ports on a device.

TOPIC USAGE: DcMeasurementUpdate, DcSummaryMeasurementUpdate

EXTENSIBILITY: extensibility(APPENDABLE)

PATTERN: Structure

ATTRIBUTES:

Name	Type and Description
`deviceId`	`tms::Identity` The device described by this structure. Annotations: *keyval*
`timeMeasured`	`tms::ClockMonotonic` Time of these measurements.
`externalMeasurement`	`PowerPortMeasurementSequence` Measurements for all power ports where PowerPortInfo.hasExternalMeter is true.
`internalMeasurement`	`PowerPortMeasurementSequence` Measurements for all power ports where PowerPortInfo.hasInternalMeter is true.

DESCRIPTION:

Different values are used for each measurement topic, as described below.

The DcMeasurementUpdate topic uses both the `externalMeasurement` and `internalMeasurement` attributes. Also, the `tms::dc::PowerPortMeasurement` contains separate values for each individual wire.

The DcSummaryMeasurementUpdate topic only uses the `externalMeasurement` for power ports that set `tms::PowerPortInfo.hasSummaryMeasurementUpdate` to true. This topic always has a zero-length `internalMeasurement` value. Also, the `tms::dc::PowerPortMeasurement` contains a single value summarized across all the wires.

B.19.2.2 `tms::Identity`

See definition in Section B.2.2.2.

B.19.2.3 `tms::ClockMonotonic`

See definition in Section B.8.2.3.

B.19.2.4 `tms::dc::PowerPortMeasurementSequence`

PURPOSE: A sequence of dc:PowerPortMeasurement.

EXTENSIBILITY: extensibility(APPENDABLE)

PATTERN: Typedef

ORIGINAL TYPE: `sequence<PowerPortMeasurement,0,tms::MAX_PORTS>`

B.19.2.5 `tms::dc::PowerPortMeasurement`

PURPOSE: Power measurement for a direct current power port.

TOPIC USAGE: Nested

EXTENSIBILITY: extensibility(APPENDABLE)

PATTERN: Structure

ATTRIBUTES:

Name	Type and Description
`portNumber`	`tms::PowerPortNumber` Port being measured.
`line`	`PowerLineMeasurementSequence` Measured values for each DC line. Length and order of values are based on PowerPortInfo.wiring. PHASE_DC=2, PHASE_DC_3WIRE=3
`powerRateOfChange`	`float32` Time derivative of power, summed across all wires excluding neutral. Annotations: *units*=watt per second (W/s)

DESCRIPTION:

Different values are used for the DcMeasurementUpdate and DcSummaryMeasurementUpdate topics, as described below.

The DcMeasurementUpdate topic uses separate values for each line in the power port. The order of values in `line` must match the `tms::CircuitWiring` type reported in `tms::PowerPortInfo.wiring`, as specified in Table B.212 and Table B.213. The V_{NG} measurements are referenced to the protective chassis ground.

Values at the end of the `line` sequences may be omitted when the device is not constructed to measure them. The number of values shall be constant and not change from sample to sample. Any "skipped" values before the end of the sequences shall be filled with NaN, so that later values in the sequence are not mis-interpreted.

Table B.212: DcMeasurementUpdate line order for `WIRING_DC`.

Voltage	Current
V_{+G}	I_{+}
V_{-G}	I_{-}

Table B.213: DcMeasurementUpdate line order for `WIRING_DC_3WIRE`.

Voltage	Current
V_{+N}	I_{+}
V_{-N}	I_{-}
V_{NG}	I_{N}

The DcSummaryMeasurementUpdate topic uses a single value to summarize all lines in the power port. There is one value in `line`, as specified in Table B.214. The calculations used to summarize the lines are specified in Table B.215.

Table B.214: DcSummaryMeasurementUpdate line order.

Voltage	Current
$V_{summary}$	$I_{summary}$

Table B.215: `PowerLineMeasurement` values for DcSummaryMeasurementUpdate.

Attribute	Summary Value
`voltage`	Sum across lines, excluding neutral to ground.
`amperage`	Average absolute value across lines, excluding neutral.
`power`	Sum across lines, excluding neutral.

B.19.2.6 `tms::PowerPortNumber`

See definition in Section B.2.2.4.

B.19.2.7 `tms::dc::PowerLineMeasurementSequence`

PURPOSE: A sequence of PowerLineMeasurement. Supports the DC circuits defined in PowerConnectorPhases.

EXTENSIBILITY: extensibility(APPENDABLE)

PATTERN: Typedef

ORIGINAL TYPE: `sequence<PowerLineMeasurement,1,3>`

DESCRIPTION:

See `tms::dc::PowerPortMeasurement` for the order of values in this sequence.

B.19.2.8 `tms::dc::PowerLineMeasurement`

PURPOSE: Measurements for a single DC power line.

TOPIC USAGE: Nested

EXTENSIBILITY: extensibility(APPENDABLE)

PATTERN: Structure

ATTRIBUTES:

Name	Type and Description
`voltage`	`float32` Averaged DC voltage. Annotations: *units*=volt (V)
`amperage`	`float32` Averaged DC amperage. Annotations: *units*=ampere (A)
`power`	`float32` Averaged DC power. Annotations: *units*=watt (W)

DESCRIPTION:

These measurements represent a single line (power, neutral, or summary) in a power port. Different values are used for each measurement topic, as described in `tms::dc::PowerPortMeasurement`.

These values should be time-averaged over regular intervals so that each update contains new information with no gaps in coverage between updates. The DcMeasurementUpdate values

are averaged over 1 s intervals, and the DcSummaryMeasurementUpdate values are over 0.1 s intervals.

Some of these values are usually very similar across lines. Separate values for each line enable the detection of load imbalances, poor connections, and other power issues. Measurements of the neutral line may be compared across different devices to detect certain grounding issues.

Some of these values can be calculated from each other using assumptions about how the microgrid is constructed and operating. The "redundant" values may better reflect the actual instrumentation, provide better performance during changes, or be used to detect faults common in distributed systems.

Values that are not measured, either due to design or malfunction, are filled with NaN. Default, nominal, previous, and projected values must not be used.

B.20 Power Topology

Power topology describes how power devices are connected to create a microgrid. Topology may be reported by the operator, a power device, or the microgrid controller.

The topology is represented as lists of connections. Each connection links a single power port on one device to a single power port on another device. A "T" cable or other splitter may be used to connect one port to multiple other ports, and this is represented by multiple connections in the list.

B.20.1 Data Flows

Figure B.33 illustrates all the topics in the data flow.

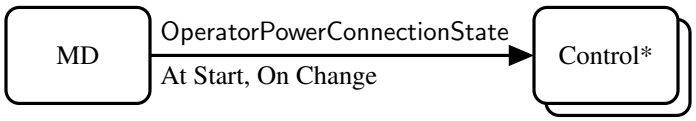


Figure B.33: Power Topology data flow

Figure B.34 illustrates all the topics in the data flow.

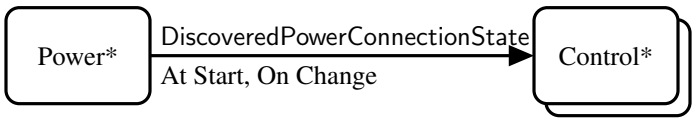


Figure B.34: Power Topology data flow

Figure B.35 illustrates all the topics in the data flow.

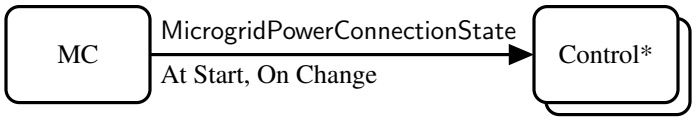


Figure B.35: Power Topology data flow

Table B.217 provides an overview of how each topic is used.

Table B.217: Description of the Power Topology topics.

Topic	Description
OperatorPowerConnectionState	Power connections manually entered by a microgrid operator.
DiscoveredPowerConnectionState	Power connections discovered by a power device, usually through a smart cable or operator entry.
MicrogridPowerConnectionState	Power connections used by the microgrid controller. May include operator connections, discovered connections, and other connections inferred by the MC.

Table B.218 specifies the publishers and subscribers for each topic as well as usage requirements as described in Section A.8.7.

Table B.218: Participants of the Power Topology topics.

Topic	Usage	MD	MC	MON	SRC	STOR	DIST	LOAD	CONV
OperatorPowerConnectionState	Optional	Pub							
			Sub	Sub					
DiscoveredPowerConnectionState	Optional				Pub	Pub	Pub	Pub	Pub
			Sub	Sub					
MicrogridPowerConnectionState	Optional		Pub						
		Sub		Sub					

This participants table has the following interpretation.

1. OperatorPowerConnectionState shall be published by MD devices that have this topic listed in `TopicInfo.optionalTopics`.
2. OperatorPowerConnectionState may be subscribed by MC and MON devices.
3. DiscoveredPowerConnectionState shall be published by SRC, STOR, DIST, LOAD, and CONV devices that have this topic listed in `TopicInfo.optionalTopics`.
4. DiscoveredPowerConnectionState may be subscribed by MC and MON devices.
5. MicrogridPowerConnectionState shall be published by MC devices that have this topic listed in `TopicInfo.optionalTopics`.

6. MicrogridPowerConnectionState may be subscribed by MD and MON devices.

Table B.219 specifies the timing for when sample updates are sent to each topic. These parameters are defined in Section A.8.5.

Table B.219: Timing of the Power Topology topics.

Topic	Data Trigger	Rate Class	Burst Size	QoS Profile
OperatorPowerConnectionState	At Start, On Change	10 s	1	PublishLast
DiscoveredPowerConnectionState	At Start, On Change	10 s	1	PublishLast
MicrogridPowerConnectionState	At Start, On Change	10 s	1	PublishLast

B.20.2 Data Types

Table B.220 specifies the data types that are used as sample values for each topic.

Table B.220: Data types of the Power Topology topics.

Topic	Data Type
OperatorPowerConnectionState	`tms::PowerConnectionState`
DiscoveredPowerConnectionState	`tms::PowerConnectionState`
MicrogridPowerConnectionState	`tms::PowerConnectionState`

Topic-level data types are defined in the order shown in Table B.220. Then nested data types are defined in the order of appearance. Cross-references are provided for data types that have already been defined.

B.20.2.1 `tms::PowerConnectionState`

PURPOSE: List all power connections known to a device or user

TOPIC USAGE: OperatorPowerConnectionState, DiscoveredPowerConnectionState, MicrogridPowerConnectionState

EXTENSIBILITY: extensibility(APPENDABLE)

PATTERN: Structure

ATTRIBUTES:

Name	Type and Description
`deviceId`	`Identity` The reporting device or user. Annotations: *keyval*
`connections`	`PowerConnectionSequence` List of power connections being reported by this device or user.

B.20.2.2 `tms::Identity`

See definition in Section B.2.2.2.

B.20.2.3 `tms::PowerConnectionSequence`

PURPOSE: A sequence of PowerConnection.

EXTENSIBILITY: extensibility(FINAL)

PATTERN: Typedef

ORIGINAL TYPE: `sequence<PowerConnection,0,200>`

B.20.2.4 `tms::PowerConnection`

PURPOSE: Identify links between devices

TOPIC USAGE: Nested

EXTENSIBILITY: extensibility(APPENDABLE)

PATTERN: Structure

ATTRIBUTES:

Name	Type and Description
`connectionId`	`PowerConnectionId` Connection described by this object.
`detectionType`	`PowerConnectionDetectionType` Technique used to detect this connection.
`detectionSource`	`Identity` Device or user that detected this connection.
`detectionConfidence`	`float32` Confidence in the link detection. -1 = confident the link does not exist, 0 = no opinion, 1 = confident the link exists. Annotations: *min*=-1, *max*=1

DESCRIPTION:

All devices should monitor local connections for a change in state. Loss active monitoring may trigger a transition to PCDT_HISTORY. This loss may occur due to device restart, cable sense indicating no cable (CS_NONE), or loss of line voltage when cable sense is not available.

B.20.2.5 `tms::PowerConnectionId`

PURPOSE: Identify a connection between power ports

TOPIC USAGE: Nested

EXTENSIBILITY: extensibility(APPENDABLE)

PATTERN: Structure

ATTRIBUTES:

Name	Type and Description
`portA`	`PowerPortId` One end of the connection.
`portB`	`optional<PowerPortId>` The other end of the connection. Absent if not available. See PowerConnectionDetectionType.

B.20.2.6 `tms::PowerPortId`

See definition in Section B.17.2.9.

B.20.2.7 `tms::PowerPortNumber`

See definition in Section B.2.2.4.

B.20.2.8 `tms::PowerConnectionDetectionType`

PURPOSE: Indicate how a power connection was detected.

EXTENSIBILITY: extensibility(APPENDABLE)

PATTERN: Enumeration

VALUES:

Name	Description
`PCDT_UNKNOWN`	Unknown value. This is a generic placeholder representing the case in which an appropriate value could not be determined.
`PCDT_OPERATOR`	Operator entry. PowerConnectionId portA and portB are present.
`PCDT_CABLE_ID`	Identification embedded in a smart cable. PowerConnectionId portA is present. PowerConnectionId portB is absent for power device connections. PortB is present for MC power connections. PowerPortStatus.cableId must be present

`PCDT_PROBE`
: Active probe signal sent down the line. PowerConnectionId portA is present, portB is absent.

`PCDT_CORRELATION`
: Correlation of power measurements was used. PowerConnectionId portA and portB are present.

`PCDT_COMMUNICATIONS`
: Power line carrier or similar network communications was used. PowerConnectionId portA and portB are present.

`PCDT_HISTORY`
: Device reset or other event has lost continuous monitoring of the connection. This connection was detected before the event and has neither been confirmed nor denied since.

DESCRIPTION:

For `PCDT_PROBE`, the `PowerConnectionId` is incomplete because the reporting power device sees power connectivity but does not know what is on the other end of the line. This may prompt the MD or MC to complete the connection using operator input or system-level information.

B.21 AC Load Sharing

AC load sharing allows multiple devices to share power on a common AC bus. Reference values and operating bounds are provided for regulating frequency, voltage, real power, and reactive power. A future version of this standard may provide support for load sharing groups that use summary power measurements in the control loop.

The load sharing capabilities of a device are announced using `tms::LoadSharingInfo`.

Control references are defined by two curves, as shown in Figure B.36. One curve defines the desired relationship between frequency and real power, and the other curve defines the desired relationship between voltage and reactive power. Control loops in each power device should keep the AC measurements on or near these reference curves.

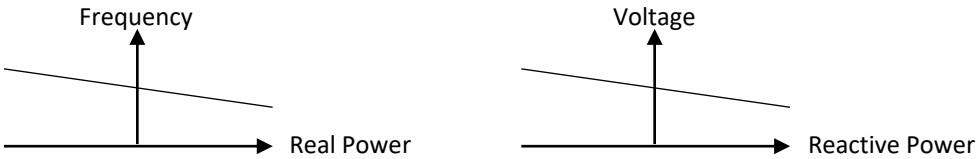


Figure B.36: Example AC Load Sharing Control References

Operating bounds contain minimum and maximum desired values for each measurement, as shown in Figure B.37. These bounds may be set to match the device's frequency, voltage, and power ratings. Looser bounds may have no effect, and tighter bounds may be used to keep the device within a narrower operating region. Positive power bounds produce or import power to the microgrid, negative bounds consume or export power from the microgrid, and positive and negative bounds do both.

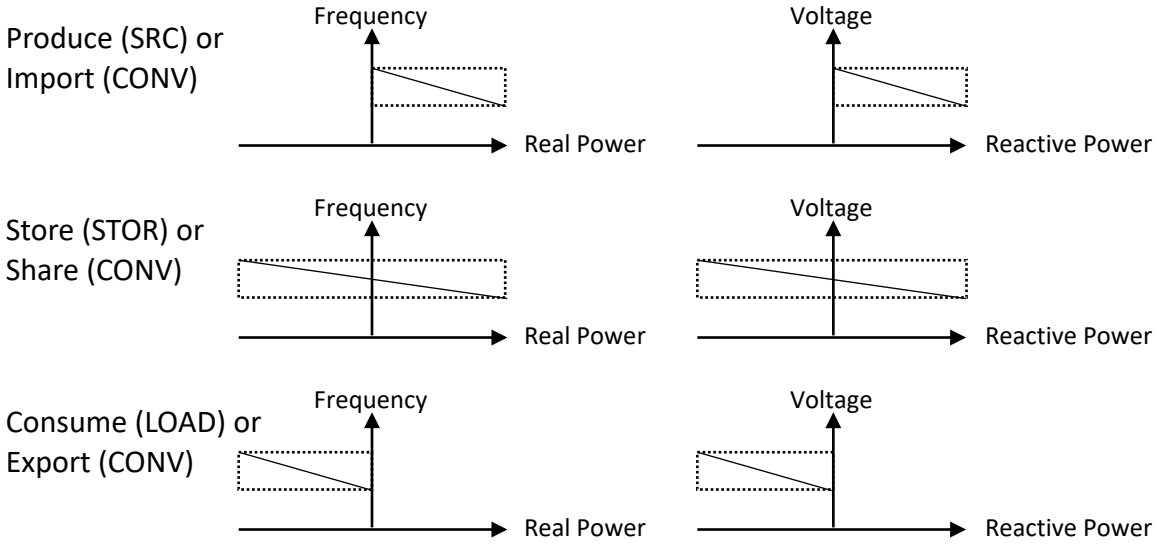


Figure B.37: Example AC Load Sharing Operating Bounds

The control references support three types of regulation, as shown in Figure B.38. Isochronous frequency and constant voltage uses reference values that do not change with power. Droop frequency and voltage uses reference values that do change with power. Constant power regulation uses operating bounds instead of reference values. Different types of regulation may be used for frequency and voltage at the same time.

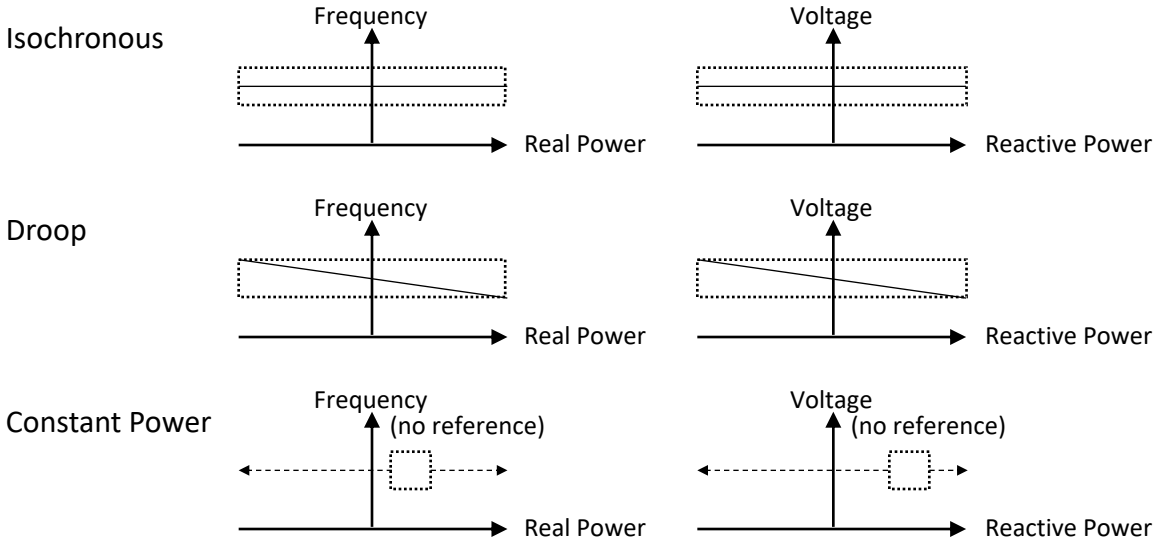


Figure B.38: Example AC Load Sharing Types

The control references support values that are defined by multiple connected line segments, as shown in Figure B.39. Multi-segment lines have uses including smooth transitions when a device nears it operating limits, frequency and voltage stiffness that varies with total load, and adjustments to which devices carry base or peak load.

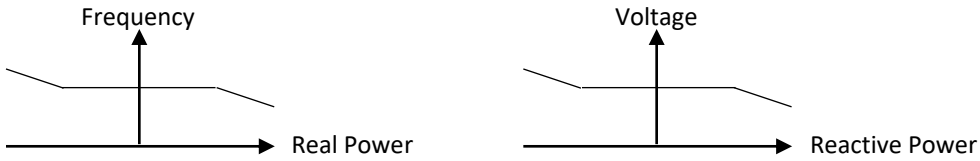


Figure B.39: Example Multi-Segment AC Load Sharing

The control references support minimum, maximum, and nominal values, as shown in Figure B.40. It is common to only use the nominal reference value. The minimum and maximum values have uses including robust control laws, thresholds for gain scheduling, and providing a hysteresis band to avoid poor control interactions between devices that respond at different speeds.

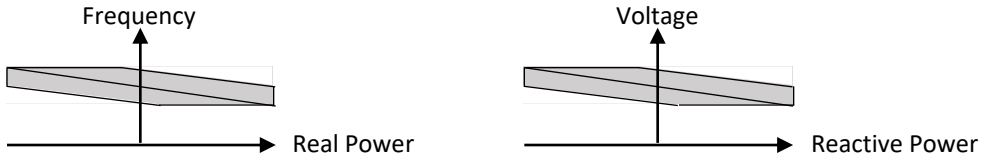


Figure B.40: Example AC Load Sharing Region

B.21.1 Data Flows

Figure B.41 illustrates all the topics in the data flow.

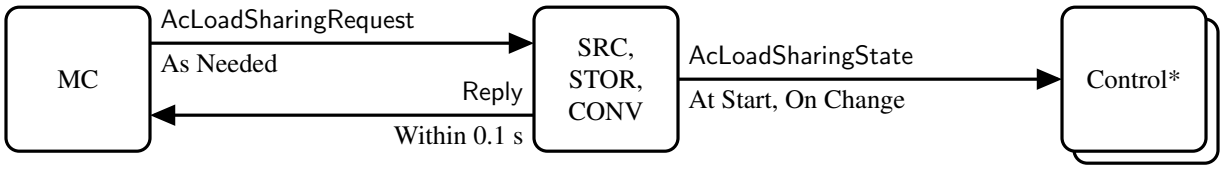


Figure B.41: AC Load Sharing data flow

Unlike many other data flows, the AcLoadSharingState contains the parameter values that were requested or set by manual or automatic action. The AC measurements are used to monitor actual operation.

Table B.225 provides an overview of how each topic is used.

Table B.225: Description of the AC Load Sharing topics.

Topic	Description
AcLoadSharingState	Reports the active load sharing status of an alternating current Source / Storage device.
AcLoadSharingRequest	Request to change the AC load sharing status of a Source / Storage device.
Reply	The reply to a request.

Table B.226 specifies the publishers and subscribers for each topic as well as usage requirements as described in Section A.8.7.

Table B.226: Participants of the AC Load Sharing topics.

Topic	Usage	MD	MC	MON	SRC	STOR	DIST LOAD	CONV
AcLoadSharingState	Conditional				Pub	Pub		Pub
		Sub	Sub	Sub				
AcLoadSharingRequest	Optional		Pub					
				Sub	Sub	Sub		Sub
Reply	Required				Pub	Pub		Pub
		Sub	Sub	Sub				

This participants table has the following interpretation.

1. AcLoadSharingState is conditional on HAS_AC_PORTS.
2. AcLoadSharingState shall be published by SRC, STOR, and CONV devices that have any `PowerDeviceInfo.powerPorts` with at least one `PowerPortInfo.wiring` of AC type.
3. AcLoadSharingState may be subscribed by MD, MC, and MON devices.
4. AcLoadSharingRequest shall be published by MC devices that have this topic listed in `TopicInfo.optionalTopics`.
5. AcLoadSharingRequest may be subscribed by MON, SRC, STOR, and CONV devices.
6. Reply shall be published by SRC, STOR, and CONV devices.
7. Reply may be subscribed by MD, MC, and MON devices.

Table B.227 specifies the timing for when sample updates are sent to each topic. These parameters are defined in Section A.8.5.

Table B.227: Timing of the AC Load Sharing topics.

`Topic`	Data Trigger	Rate Class	Burst Size	QoS Profile
AcLoadSharingState	At Start, On Change	10 s	10	PublishLast
AcLoadSharingRequest	As Needed	10 s	10	Command
Reply	Within 0.1 s	All	1	Reply

B.21.2 Data Types

Table B.228 specifies the data types that are used as sample values for each topic.

Table B.228: Data types of the AC Load Sharing topics.

Topic	Data Type
AcLoadSharingState	`tms::ac::LoadSharingState`
AcLoadSharingRequest	`tms::ac::LoadSharingRequest`
Reply	`tms::Reply`

Topic-level data types are defined in the order shown in Table B.228. Then nested data types are defined in the order of appearance. Cross-references are provided for data types that have already been defined.

B.21.2.1 `tms::ac::LoadSharingState`

PURPOSE: Report the present value of the load sharing parameters for alternating current devices.

TOPIC USAGE: AcLoadSharingState

EXTENSIBILITY: extensibility(APPENDABLE)

PATTERN: Structure

ATTRIBUTES:

Name	Type and Description
`deviceId`	`tms::Identity` The device described by this structure. Annotations: *keyval*
`config`	`tms::ConfigId` Configuration that these values belong to.
`parameters`	`LoadSharingParameterSequence` LoadSharingParameters for each power port that supports AC load sharing.

B.21.2.2 `tms::ac::LoadSharingRequest`

PURPOSE: Sets load sharing parameters for AC power ports.

TOPIC USAGE: AcLoadSharingRequest

EXTENSIBILITY: extensibility(APPENDABLE)

PATTERN: Structure

ATTRIBUTES:

Name	Type and Description
`requestId`	`tms::PowerPortConfigRequest` Identity of this request. Annotations: *keyval*
`sequenceId`	`tms::RequestSequence` Request sequence data used to associate a request and returning reply.
`parameters`	`LoadSharingParameters` Desired parameters for controlling frequency, voltage, and power.

DESCRIPTION:

Both AcLoadSharingRequest and AcLoadSharingState use the same `LoadSharingParameters` data type so it can be simply copied from one to the other. However, the request is keyed on port number and the state is not, causing some redundancy between the `PowerPortConfigRequest.portNumber` and `LoadSharingParameters.portNumber` attributes. Both attributes must be set to the same value in the request.

B.21.2.3 `tms::Reply`

See definition in Section B.2.2.1.

B.21.2.4 `tms::Identity`

See definition in Section B.2.2.2.

B.21.2.5 `tms::ConfigId`

See definition in Section B.2.2.3.

B.21.2.6 `tms::ac::LoadSharingParameterSequence`

PURPOSE: A sequence of AC LoadSharingParameters.

EXTENSIBILITY: extensibility(APPENDABLE)

PATTERN: Typedef

ORIGINAL TYPE: `sequence<LoadSharingParameters,1,tms::MAX_PORTS>`

B.21.2.7 `tms::ac::LoadSharingParameters`

PURPOSE: Load sharing parameters for AC power ports.

TOPIC USAGE: Nested

EXTENSIBILITY: extensibility(APPENDABLE)

PATTERN: Structure

ATTRIBUTES:

Name	Type and Description
`portNumber`	`tms::PowerPortNumber` The power port number.
`referenceFrequency`	`tms::ControlCurve` Desired frequency as a function of output real power. Zero points disables frequency regulation and enables bounded or constant real power regulation. One point is invalid. Two points with the same frequency enables constant frequency (isochronous) regulation. Two or more points with different frequencies enables droop frequency regulation. Annotations: *units*=watt, hertz (W, Hz)
`referenceVoltage`	`tms::ControlCurve` Desired voltage as a function of output reactive power. Zero points disables voltage regulation and enables bounded or constant reactive power regulation. One point is invalid. Two points with the same voltage enables constant voltage regulation. Two or more points with different voltages enables droop voltage regulation. Annotations: *units*=volt ampere reactive, volt (var, V)
`minRealPower`	`float32` Minimum desired real power output. May limit the device's ability to regulate frequency. Annotations: *units*=watt (W)
`maxRealPower`	`float32` Maximum desired real power output. May limit the device's ability to regulate frequency. Annotations: *units*=watt (W)
`minFrequency`	`float32` Minimum desired voltage output. May limit the device's ability to regulate real power. Annotations: *units*=hertz (Hz)

`maxFrequency`
: `float32`
Maximum desired voltage output. May limit the device's ability to regulate real power.
Annotations: *units*=hertz (Hz)

`minReactivePower`
: `float32`
Minimum desired reactive power output. May limit the device's ability to regulate voltage.
Annotations: *units*=volt ampere reactive (var)

`maxReactivePower`
: `float32`
Maximum desired reactive power output. May limit the device's ability to regulate voltage.
Annotations: *units*=volt ampere reactive (var)

`minVoltage`
: `float32`
Minimum desired voltage output. May limit the device's ability to regulate reactive power.
Annotations: *units*=volt (V)

`maxVoltage`
: `float32`
Maximum desired voltage output. May limit the device's ability to regulate reactive power.
Annotations: *units*=volt (V)

B.21.2.8 `tms::PowerPortNumber`

See definition in Section B.2.2.4.

B.21.2.9 `tms::ControlCurve`

PURPOSE: Specify output variable y as a function of input variable x.

TOPIC USAGE: Nested

EXTENSIBILITY: extensibility(APPENDABLE)

PATTERN: Structure

ATTRIBUTES:

Name	Type and Description
`minimum`	`Curve2D` Minimum acceptable value of y. Devices should operate above this curve.
`nominal`	`Curve2D` Nominal value of y. Used as the set-point for PID controls.
`maximum`	`Curve2D` Maximum acceptable value of y. Devices should operate below this curve.

The `ControlCurve` contains minimum, nominal, and maximum reference values, as shown in Figure B.42. Valid `ControlCurve` values have `minimum.y` $\leq$ `nominal.y` $\leq$ `maximum.y` for every horizontal coordinate `x`.

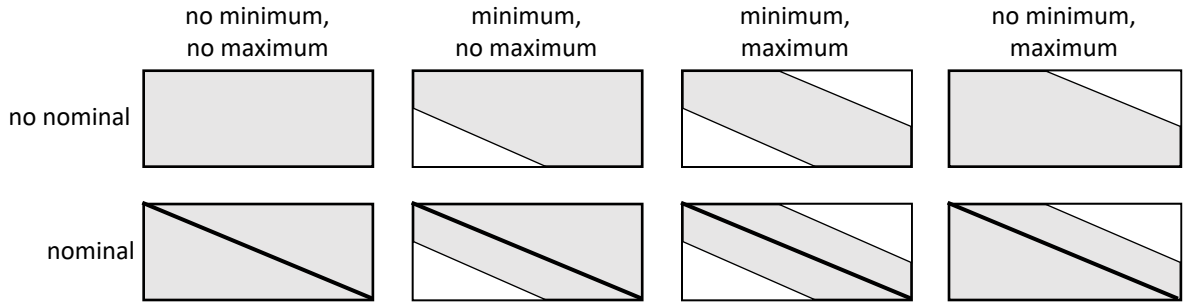


Figure B.42: Interpreting ControlCurve minimum, nominal, and maximum.

Control loops should keep the measured value above the minimum, near the nominal, and below the maximum. These values may be used to define bounded regions with or without a nominal value. The bounds may implemented in many ways including hysteresis curves, thresholds for gain scheduling, and a target set for robust bounded-input bounded-output control.

Devices are not required to support all combinations of the minimum, nominal, and maximum curves. Devices may `REPLY_BAD_REQUEST` to missing or unsupported values.

B.21.2.10 `tms::Curve2D`

See definition in Section B.5.2.58.

B.21.2.11 `tms::Point2DSequence`

See definition in Section B.5.2.59.

B.21.2.12 `tms::Point2D`

See definition in Section B.5.2.60.

B.21.2.13 `tms::PowerPortConfigRequest`

See definition in Section B.17.2.14.

B.21.2.14 `tms::RequestSequence`

See definition in Section B.2.2.5.

B.21.2.15 `tms::ReplyStatus`

See definition in Section B.2.2.6.

B.21.2.16 `tms::ReplyCode`

See definition in Section B.2.2.7.

B.21.2.17 `tms::String32`

See definition in Section B.2.2.8.

B.22 DC Load Sharing

DC load sharing allows multiple devices to share power on a common DC bus. Reference values and operating bounds are provided for regulating voltage and amperage. A future version of this standard may provide support for load sharing groups that use summary power measurements in the control loop.

The load sharing capabilities of a device are announced using `tms::LoadSharingInfo`.

Control references are defined by a single curve, as shown in Figure B.43. This curve defines the desired relationship between voltage and amperage. Control loops in each power device should keep the DC measurements on or near this reference curve.

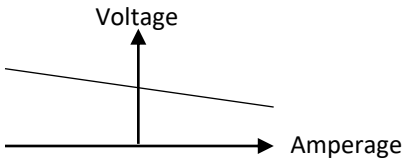


Figure B.43: Example DC Load Sharing Control References

Operating bounds contain minimum and maximum desired values for each measurement, as shown in Figure B.44. These bounds may be set to match the device's voltage and amperage ratings. Looser bounds may have no effect, and tighter bounds may be used to keep the device within a narrower operating region. Positive amperage bounds produce or import power to the microgrid, negative bounds consume or export power from the microgrid, and positive and negative bounds do both.

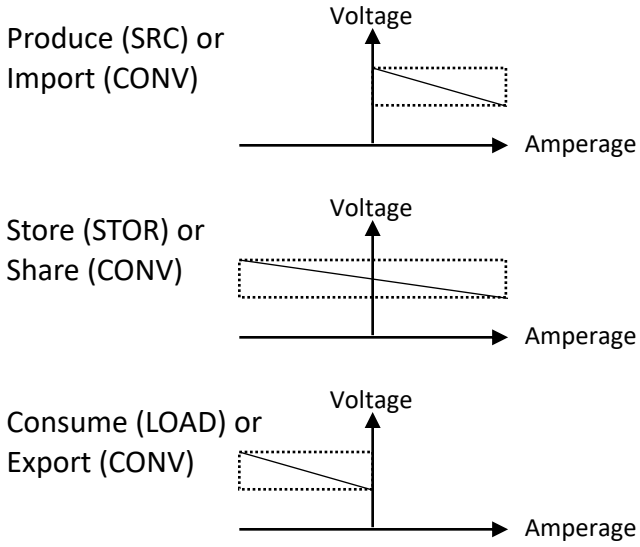


Figure B.44: Example DC Load Sharing Operating Bounds

The control references support three types of regulation, as shown in Figure B.45. Constant voltage uses reference values that do not change with power. Droop voltage uses reference values that do change with power. Constant power regulation uses operating bounds instead of reference values.

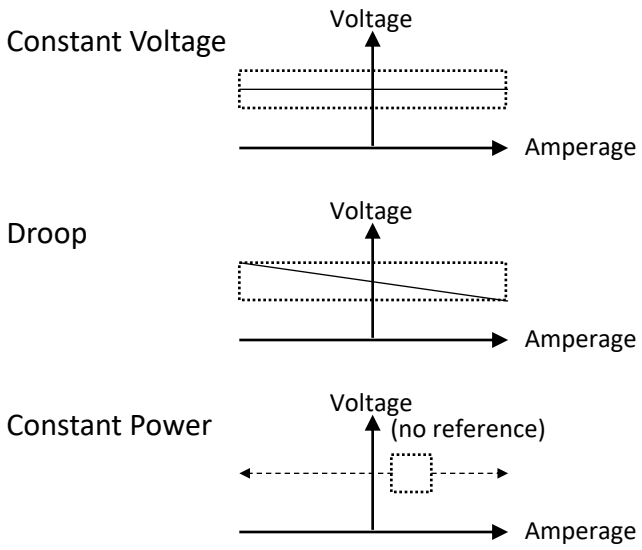


Figure B.45: Example DC Load Sharing Types

The control reference supports values that are defined by multiple connected line segments, as shown in Figure B.46. Multi-segment lines have uses including smooth transitions when a device nears it operating limits, frequency and voltage stiffness that varies with total load, and adjustments to which devices carry base or peak load.

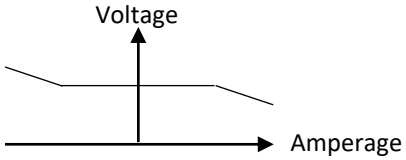


Figure B.46: Example Multi-Segment DC Load Sharing

The control reference supports minimum, maximum, and nominal values, as shown in Figure B.47. It is common to only use the nominal reference value. The minimum and maximum values have uses including robust control laws, thresholds for gain scheduling, and providing a hysteresis band to avoid poor control interactions between devices that respond at different speeds.

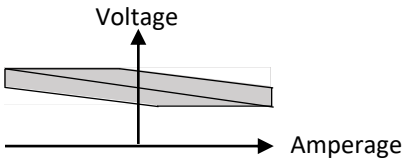


Figure B.47: Example DC Load Sharing Region

B.22.1 Data Flows

Figure B.48 illustrates all the topics in the data flow.

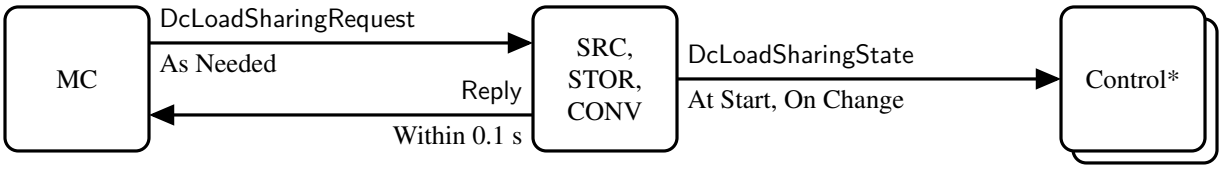


Figure B.48: DC Load Sharing data flow

Unlike many other data flows, the DcLoadSharingState contains the parameter values that were requested or set by manual or automatic action. The DC measurements are used to monitor actual operation.

Table B.233 provides an overview of how each topic is used.

Table B.233: Description of the DC Load Sharing topics.

Topic	Description
DcLoadSharingState	Reports the active direct current load sharing status of a device.
DcLoadSharingRequest	Request to change the DC load sharing status of a device.
Reply	The reply to a request.

Table B.234 specifies the publishers and subscribers for each topic as well as usage requirements as described in Section A.8.7.

Table B.234: Participants of the DC Load Sharing topics.

Topic	Usage	MD	MC	MON	SRC	STOR	DIST LOAD	CONV
DcLoadSharingState	Conditional				Pub	Pub		Pub
		Sub	Sub	Sub				
DcLoadSharingRequest	Optional		Pub					
				Sub	Sub	Sub		Sub
Reply	Required				Pub	Pub		Pub
		Sub	Sub	Sub				

This participants table has the following interpretation.

1. DcLoadSharingState is conditional on HAS_DC_PORTS.
2. DcLoadSharingState shall be published by SRC, STOR, and CONV devices that have any `PowerDeviceInfo.powerPorts` with at least one `PowerPortInfo.wiring` of DC type.

3. DcLoadSharingState may be subscribed by MD, MC, and MON devices.
4. DcLoadSharingRequest shall be published by MC devices that have this topic listed in `TopicInfo.optionalTopics`.
5. DcLoadSharingRequest may be subscribed by MON, SRC, STOR, and CONV devices.
6. Reply shall be published by SRC, STOR, and CONV devices.
7. Reply may be subscribed by MD, MC, and MON devices.

Table B.235 specifies the timing for when sample updates are sent to each topic. These parameters are defined in Section A.8.5.

Table B.235: Timing of the DC Load Sharing topics.

Topic	Data Trigger	Rate Class	Burst Size	QoS Profile
DcLoadSharingState	At Start, On Change	10 s	10	PublishLast
DcLoadSharingRequest	As Needed	10 s	10	Command
Reply	Within 0.1 s	All	1	Reply

B.22.2 Data Types

Table B.236 specifies the data types that are used as sample values for each topic.

Table B.236: Data types of the DC Load Sharing topics.

Topic	Data Type
DcLoadSharingState	`tms::dc::LoadSharingState`
DcLoadSharingRequest	`tms::dc::LoadSharingRequest`
Reply	`tms::Reply`

Topic-level data types are defined in the order shown in Table B.236. Then nested data types are defined in the order of appearance. Cross-references are provided for data types that have already been defined.

B.22.2.1 `tms::dc::LoadSharingState`

PURPOSE: Report the present value of the load sharing parameters for direct current devices.

TOPIC USAGE: DcLoadSharingState

EXTENSIBILITY: extensibility(APPENDABLE)

PATTERN: Structure

ATTRIBUTES:

Name	Type and Description
`deviceId`	`tms::Identity` The device described by this structure. Annotations: *keyval*
`config`	`tms::ConfigId` Configuration that these values belong to.
`parameters`	`LoadSharingParameterSequence` LoadSharingParameters for each power port that support DC load sharing.

B.22.2.2 `tms::dc::LoadSharingRequest`

PURPOSE: Sets load sharing parameters for DC power ports.

TOPIC USAGE: DcLoadSharingRequest

EXTENSIBILITY: extensibility(APPENDABLE)

PATTERN: Structure

ATTRIBUTES:

Name	Type and Description
`requestId`	`tms::PowerPortConfigRequest` Identity of this request. Annotations: *keyval*
`sequenceId`	`tms::RequestSequence` Request sequence data used to associate a request and returning reply.
`parameters`	`LoadSharingParameters` Desired parameters for controlling voltage and power.

DESCRIPTION:

Both DcLoadSharingRequest and DcLoadSharingState use the same `LoadSharingParameters` data type so it can be simply copied from one to the other. However, the request is keyed on port number and the state is not, causing some redundancy between the `PowerPortConfigRequest.portNumber` and `LoadSharingParameters.portNumber` attributes. Both attributes must be set to the same value in the request.

B.22.2.3 `tms::Reply`

See definition in Section B.2.2.1.

B.22.2.4 `tms::Identity`

See definition in Section B.2.2.2.

B.22.2.5 `tms::ConfigId`

See definition in Section B.2.2.3.

B.22.2.6 `tms::dc::LoadSharingParameterSequence`

PURPOSE: A sequence of DC LoadSharingParameters.

EXTENSIBILITY: extensibility(APPENDABLE)

PATTERN: Typedef

ORIGINAL TYPE: `sequence<LoadSharingParameters,1,tms::MAX_PORTS>`

B.22.2.7 `tms::dc::LoadSharingParameters`

PURPOSE: Load sharing parameters for DC power ports.

TOPIC USAGE: Nested

EXTENSIBILITY: extensibility(APPENDABLE)

PATTERN: Structure

ATTRIBUTES:

Name	Type and Description
`portNumber`	`tms::PowerPortNumber` The power port number.
`referenceVoltage`	`tms::ControlCurve` Desired voltage as a function of output amperage. Zero points disables voltage regulation and enables bounded or constant amperage regulation. One point is invalid. Two points with the same voltage enables constant voltage regulation. Two or more points with different voltages enables droop voltage regulation. Annotations: *units*=ampere, volt (A, V)
`minAmperage`	`float32` Minimum desired amperage output. May limit the device's ability to regulate voltage. Annotations: *units*=ampere (A)
`maxAmperage`	`float32` Maximum desired amperage output. May limit the device's ability to regulate voltage. Annotations: *units*=ampere (A)
`minVoltage`	`float32` Minimum desired voltage output. May limit the device's ability to regulate amperage. Annotations: *units*=volt (V)
`maxVoltage`	`float32` Maximum desired voltage output. May limit the device's ability to regulate amperage. Annotations: *units*=volt (V)

B.22.2.8 `tms::PowerPortNumber`

See definition in Section B.2.2.4.

B.22.2.9 tms::ControlCurve

See definition in Section B.21.2.9.

B.22.2.10 tms::Curve2D

See definition in Section B.5.2.58.

B.22.2.11 tms::Point2DSequence

See definition in Section B.5.2.59.

B.22.2.12 tms::Point2D

See definition in Section B.5.2.60.

B.22.2.13 tms::PowerPortConfigRequest

See definition in Section B.17.2.14.

B.22.2.14 tms::RequestSequence

See definition in Section B.2.2.5.

B.22.2.15 tms::ReplyStatus

See definition in Section B.2.2.6.

B.22.2.16 tms::ReplyCode

See definition in Section B.2.2.7.

B.22.2.17 tms::String32

See definition in Section B.2.2.8.

B.23 Clock Status

Clock status provides information on the realtime clock, how it was set, and how it relates to the monotonic clock. Support for a realtime clock is indicated in tms::ControlHardwareInfo. A future version of this standard may provide support for requesting clock synchronization.

B.23.1 Data Flows

Figure B.49 illustrates all the topics in the data flow.

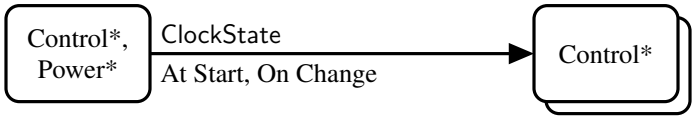


Figure B.49: Clock Status data flow

Table B.240 provides an overview of how each topic is used.

Table B.240: Description of the Clock Status topics.

Topic	Description
ClockState	Reports changes to a device clock(s). A device must support relative time and can optionally support absolute time.

Table B.241 specifies the publishers and subscribers for each topic as well as usage requirements as described in Section A.8.7.

Table B.241: Participants of the Clock Status topics.

Topic	Usage	MD	MC	MON	SRC	STOR	DIST	LOAD	CONV
ClockState	Optional	Pub Sub	Pub Sub	Sub	Pub	Pub	Pub	Pub	Pub

This participants table has the following interpretation.

1. ClockState shall be published by MD, MC, SRC, STOR, DIST, LOAD, and CONV devices that have this topic listed in `TopicInfo.optionalTopics`.
2. ClockState may be subscribed by MD, MC, and MON devices.

Table B.242 specifies the timing for when sample updates are sent to each topic. These parameters are defined in Section A.8.5.

Table B.242: Timing of the Clock Status topics.

Topic	Data Trigger	Rate Class	Burst Size	QoS Profile
ClockState	At Start, On Change	10 s	1	PublishLast

B.23.2 Data Types

Table B.243 specifies the data types that are used as sample values for each topic.

Table B.243: Data types of the Clock Status topics.

Topic	Data Type
ClockState	`tms::ClockState`

Topic-level data types are defined in the order shown in Table B.243. Then nested data types are defined in the order of appearance. Cross-references are provided for data types that have already been defined.

B.23.2.1 `tms::ClockState`

PURPOSE: Simultaneous measurement of both system clocks, and synchronization status for .

TOPIC USAGE: ClockState

EXTENSIBILITY: extensibility(APPENDABLE)

PATTERN: Structure

ATTRIBUTES:

Name	Type and Description
`deviceId`	`Identity` The device described by this structure. Annotations: *keyval*
`monotonic`	`ClockMonotonic` Measured monotonic time, sampled simultaneously with realtime.
`monotonicPersistence`	`ClockMonotonicPersistenceType` Indicate whether the monotonic clock initializes to 0 or accumulates operating time across restarts.
`realtime`	`ClockRealtime` Measured real time, sampled simultaneously with monotonic.
`realtimeSource`	`ClockSource` Source of the realtime clock, as enumerated for timeSource in IEEE-1588-2008 Table 7.
`realtimeSetTime`	`ClockRealtime` Last time the realtime clock was updated by the source.

DESCRIPTION:

The `realtimeSetTime` is set to the calibration time for `CLOCK_ATOMIC`, entry time for `CLOCK_HAND`, or the time when synchronization is acquired for other clock sources. Once set, this value only updates when a different offset between the `monotonic` and `realtime` clocks causes a significant change to device operation. The `realtimeSetTime` is t=00.0 when a valid time is not available.

B.23.2.2 `tms::Identity`

See definition in Section B.2.2.2.

B.23.2.3 `tms::ClockMonotonic`

See definition in Section B.8.2.3.

B.23.2.4 `tms::ClockMonotonicPersistenceType`

PURPOSE: Indicate how the monotonic clock is initialized.

EXTENSIBILITY: extensibility(APPENDABLE)

PATTERN: Enumeration

VALUES:

Name	Description
`CMPT_UNKNOWN`	Fault detection indicates that initialization may be inconsistent.
`CMPT_RESET`	Clock initializes to 0.
`CMPT_ACCUMULATE`	Clock initializes to a value stored shortly before shutdown. This is effectively an aggregate runtime meter.
`CMPT_OTHER`	Another clock initialization scheme is used.

B.23.2.5 tms::ClockRealtime

PURPOSE: Clock synchronized to a global time base.

TOPIC USAGE: Nested

EXTENSIBILITY: extensibility(APPENDABLE)

PATTERN: Structure

ATTRIBUTES:

Name	Type and Description
epoch	uint16 Epoch.
seconds	uint32 integer portion of the timestamp.
nanoseconds	uint32 fractional portion of the timestamp.

DESCRIPTION:

This clock is used to measure the absolute time of events. The start time, t=00.0, of this clock is as specified in IEEE 1588-2008, Precision Time Protocol (PTP). The structure defined is based on the PTP Timestamp type. The nanoseconds is equivalent to the PTP nanosecondsField. The seconds is equivalent to the least-significant 32 bits of the PTP secondsField. The epoch is the most-significant 16 bits of the PTP secondsField.

B.23.2.6 tms::ClockSource

PURPOSE: Source of the realtime clock, as enumerated for timeSource in IEEE-1588-2008 Table 7.

EXTENSIBILITY: extensibility(APPENDABLE)

PATTERN: Enumeration

VALUES:

Name	Description
CLOCK_UNKNOWN	No information is available, either due to design or malfunction.
CLOCK_ATOMIC	Direct connection to a calibrated atomic clock.
CLOCK_GPS	Satellite-based system.

`CLOCK_RADIO`
: Terrestrial radio system.

`CLOCK_PTP`
: PTP, IEEE-1588.

`CLOCK_NTP`
: NTP, RFC 5905 or SNTP, RFC 4330.

`CLOCK_HAND`
: User input.

`CLOCK_OTHER`
: Other source.

`CLOCK_INTERNAL`
: Arbitrary or unknown epoch

B.24 Authorization To Energize

Authorization to energize allows operators to supervise the SRC, STOR, DIST, and CONV device roles when they energize a circuit by automatic action or requested action such as EnergyStartStopRequest, LoadSharingRequest, or PowerSwitchRequest. Proper safety equipment and procedures are necessary to prevent injury or death to people in the vicinity of a microgrid or any other large electric power system. Programmable safety logic like authorization to energize must always be backed by hardware safety mechanisms and safe operating procedures.

A circuit is *live* when it is at or near rated voltage, and it is *dead* when it is at or near zero voltage. A power device may *energize* a circuit from dead to live or *de-energize* a circuit from live to dead.

The authorization to energize protocol allows an operator to accept or deny that a dead circuit can be safely energized. This protocol is based on the following principles.

1. Manual actions are always authorized and should not be automatically reversed.
2. Automatic (power device) and requested (MC) actions must be supervised for safety.
3. Live circuits that go dead for less than three minutes may automatically energize to restore power.
4. Dead circuits must not automatically energize after five minutes. People in the area may believe that touch-safe precautions are not required.
5. Operators must "walk the line", check for hazards, and notify everyone in the area before energizing circuits.
6. Operators must disable equipment, disconnect circuits, and perform safety procedures such lock out / tag out (LOTO) to verify that wires cannot be energized before performing maintenance activities that are not inherently touch safe.
7. Power device logic must rely on internal measurements and timers in case network communications are not available.

The timing requirements for authorization to energize are shown in Figure B.50.

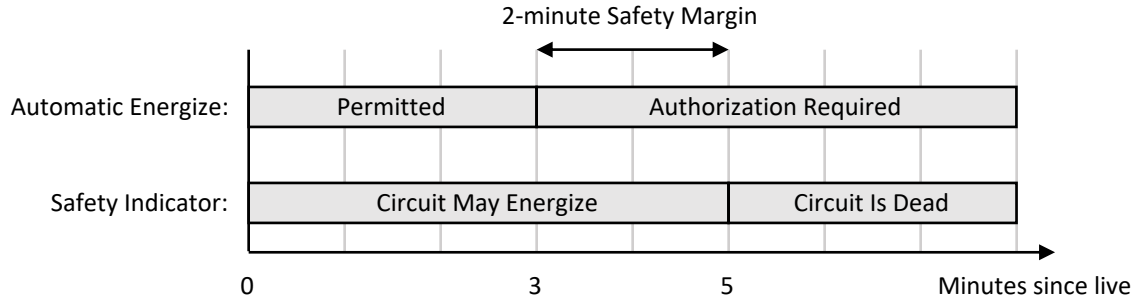


Figure B.50: Authorization To Energize Timing

A power device requires authorization to energize a circuit in the following circumstances.

1. A circuit was dead when the device initialized.
2. A circuit has been dead for at least three minutes.
3. A circuit is de-energized by a manual action on the device.

A power device does not require authorization to energize a circuit in the following circumstances.

1. A circuit is energized manually. Manual actions are implicitly authorized.
2. A circuit is de-energized for a reason external to the device, and the device attempts to energize the same circuit within three minutes.
3. The device de-energizes a circuit by an automatic or requested action and then attempts to energize the same circuit within three minutes.

Authorization may be given in one of the following ways.

1. An operator accepts the action through an on-device display.
2. An operator accepts the action through the authorization to energize data flow.

Any authorization has immediate effect. Authorization cannot be saved for future use. Authorization data samples and associated safety information should be logged for traceability.

The authorization to energize protocol is shown in Figure B.51. When a power device detects that an automatic or requested action would energize a circuit, it first sends an AuthorizationToEnergizeRequest and waits for a reply. The device should wait at least 15 seconds before canceling the request due to timeout, allowing an operator some time to respond. When an AuthorizationToEnergizeReply is received, the power device validates the reply and then cancels a denied request or completes an accepted request. When the action is canceled or completed for any reason, the power device finishes by sending an AuthorizationToEnergizeResult.

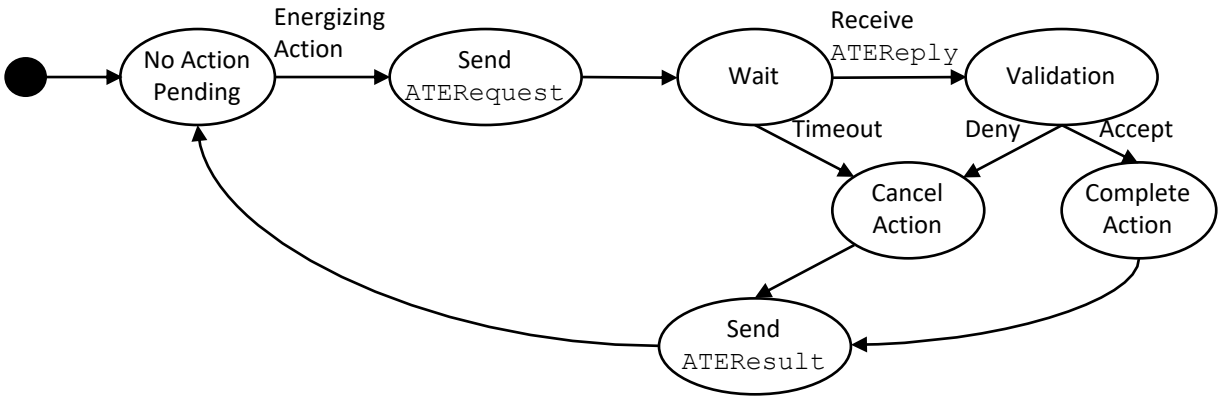


Figure B.51: Authorization To Energize Protocol

The state machine shown in Figure B.52 is recommended for tracking when authorization to energize is required.

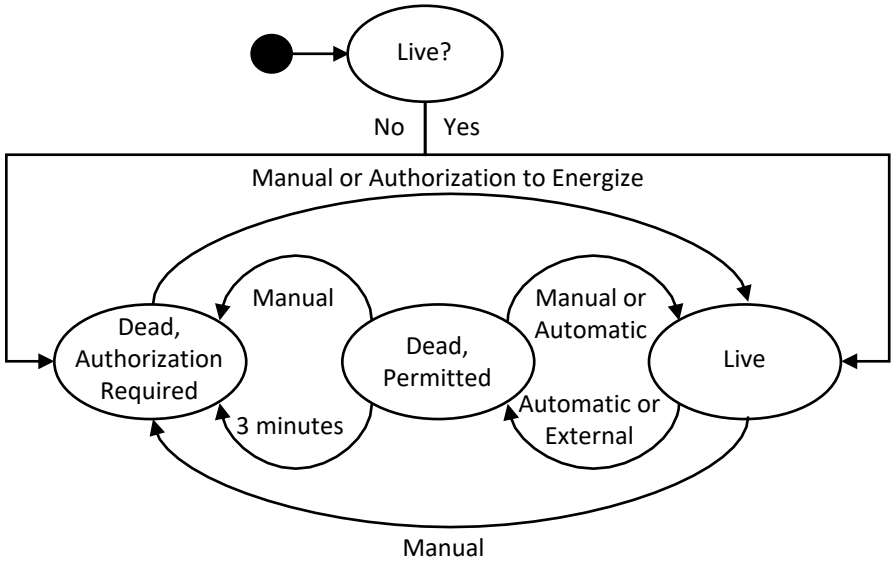


Figure B.52: Energizing Action State Diagram

The state machine shown in Figure B.53 is recommended for safety indicators. A safety indicator may indicate a circuit is live with a solid light on, dead but may energize with a blinking light, and dead with a solid light off.

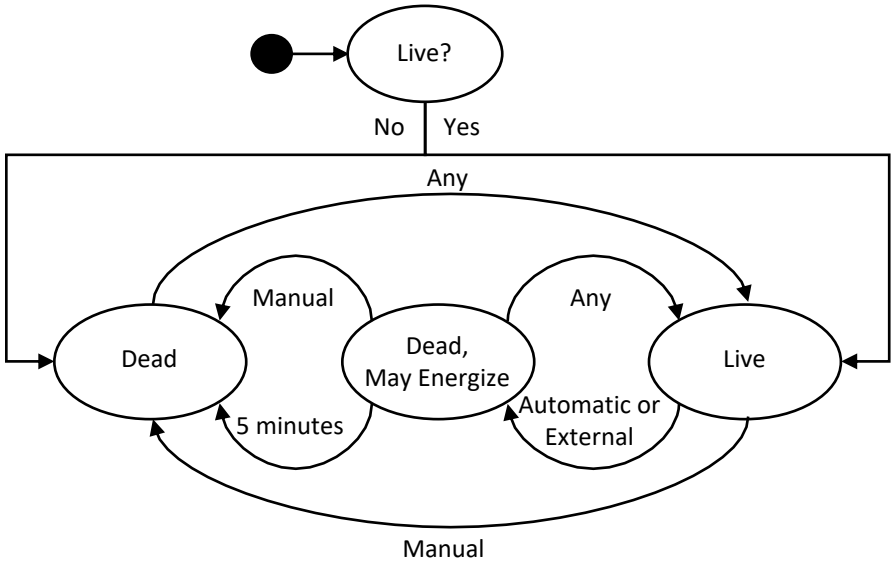


Figure B.53: Safety Indicator State Diagram

B.24.1 Data Flows

Figure B.54 illustrates all the topics in the data flow.

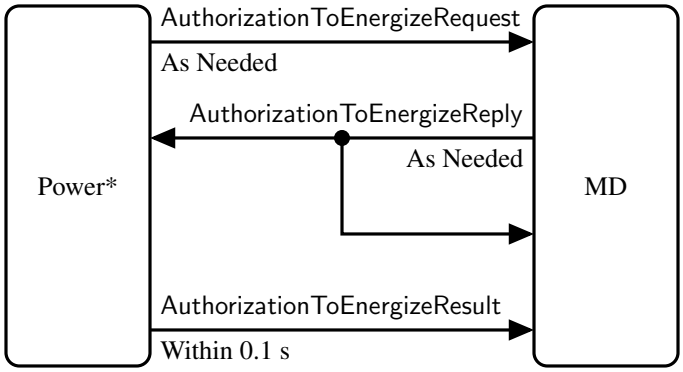


Figure B.54: Authorization To Energize data flow

Table B.248 provides an overview of how each topic is used.

Table B.248: Description of the Authorization To Energize topics.

Topic	Description
AuthorizationToEnergizeRequest	Request authorization to execute a request when a black start is detected.
AuthorizationToEnergizeReply	Provide the operator's response to an authorization request.
AuthorizationToEnergizeResult	Report the outcome of a request that required authorization.

Table B.248 specifies the publishers and subscribers for each topic as well as usage requirements as described in Section A.8.7.

Table B.249: Participants of the Authorization To Energize topics.

Topic	Usage	MD	MC	MON	SRC	STOR	DIST	LOAD	CONV
AuthorizationToEnergizeRequest	Optional				Pub	Pub	Pub		Pub
		Sub		Sub					
AuthorizationToEnergizeReply	Conditional	Pub							
		Sub		Sub	Sub	Sub	Sub		Sub
AuthorizationToEnergizeResult	Optional				Pub	Pub	Pub		Pub
		Sub	Sub	Sub					

This participants table has the following interpretation.

1. AuthorizationToEnergizeRequest shall be published by SRC, STOR, DIST, and CONV devices that have this topic listed in `TopicInfo.optionalTopics`.

2. AuthorizationToEnergizeRequest may be subscribed by MD and MON devices.

3. AuthorizationToEnergizeReply is conditional on SUPPORTS_REQUEST.

4. AuthorizationToEnergizeReply shall be published by MD devices that support the corresponding request topic.

5. AuthorizationToEnergizeReply may be subscribed by MD, MON, SRC, STOR, DIST, and CONV devices.

6. AuthorizationToEnergizeResult shall be published by SRC, STOR, DIST, and CONV devices that have this topic listed in `TopicInfo.optionalTopics`.

7. AuthorizationToEnergizeResult may be subscribed by MD, MC, and MON devices.

Table B.250 specifies the timing for when sample updates are sent to each topic. These parameters are defined in Section A.8.5.

Table B.250: Timing of the Authorization To Energize topics.

`Topic`	Data Trigger	Rate Class	Burst Size	QoS Profile
AuthorizationToEnergizeRequest	As Needed	10 s	1	Command
AuthorizationToEnergizeReply	As Needed	10 s	1	Response
AuthorizationToEnergizeResult	Within 0.1 s	10 s	1	Response

B.24.2 Data Types

Table B.251 specifies the data types that are used as sample values for each topic.

Table B.251: Data types of the Authorization To Energize topics.

Topic	`Data Type`
AuthorizationToEnergizeRequest	`tms::AuthorizationToEnergizeRequest`
AuthorizationToEnergizeReply	`tms::AuthorizationToEnergizeReply`
AuthorizationToEnergizeResult	`tms::AuthorizationToEnergizeResult`

Topic-level data types are defined in the order shown in Table B.251. Then nested data types are defined in the order of appearance. Cross-references are provided for data types that have already been defined.

B.24.2.1 `tms::AuthorizationToEnergizeRequest`

PURPOSE: Request authorization for a manual black start command.

TOPIC USAGE: AuthorizationToEnergizeRequest

EXTENSIBILITY: extensibility(APPENDABLE)

PATTERN: Structure

ATTRIBUTES:

Name	Type and Description
`requestId`	`GridRequest` Identity of this request. Annotations: *keyval*
`sequenceId`	`tms::RequestSequence` GridRequest sequence data used to associate a request and returning reply.
`energizeRequestingDeviceId`	`Identity` Copy of the requestingDeviceId from the request that requires authorization.
`energizeSequenceId`	`tms::RequestSequence` Copy of the sequenceId from the request that requires authorization.
`powerPorts`	`PowerPortNumberSequence` All power ports that would be energized by executing this request.

DESCRIPTION:

The `deviceId` and `ports` fully capture the location of the requested event. This request must be approved by an authorized operator.

B.24.2.2 `tms::AuthorizationToEnergizeReply`

PURPOSE: Provide the operator's response to an authorization request.

TOPIC USAGE: AuthorizationToEnergizeReply

EXTENSIBILITY: extensibility(APPENDABLE)

PATTERN: Structure

ATTRIBUTES:

Name	Type and Description
`deviceId`	`Identity` Identity of the device that processed the corresponding AuthorizationToEnergizeRequest and sent this response.
`relatedRequestId`	`GridRequest` Copy of the corresponding AuthorizationToEnergizeRequest.requestId. Annotations: *keyval*
`relatedSequenceId`	`RequestSequence` Copy of the corresponding AuthorizationToEnergizeRequest.sequenceId.
`energizeRequestingDeviceId`	`Identity` Copy of the corresponding AuthorizationToEnergizeRequest.energizeRequestingDeviceId.
`energizeSequenceId`	`tms::RequestSequence` Copy of the corresponding AuthorizationToEnergizeRequest.energizeSequenceId.
`powerPorts`	`PowerPortNumberSequence` Copy of the AuthorizationToEnergizeRequest.powerPorts.
`userId`	`optional<Identity>` Identity of the user that provided this response. Absent if not available.
`accept`	`boolean` True if the request is authorized to proceed.
`deny`	`boolean` True if the request must not be executed.
`timeOfReview`	`ClockRealtime` Time the user provided this response.

DESCRIPTION:

The black start policy specifies that a human operator needs to take positive steps to ensure the safety of everyone in the area when manual black start requirements are in effect. In accepting a request, the responding operator also confirms that appropriate safeguards are in place. A request may be denied for any reason.

The receiving device must compare `deviceId` and `ports` to the original request and any value mismatch shall cause this authorization to be rejected.

If both `accept` and `deny` are true in a single message or if multiple responses are received before the request is completed, then any `deny` takes precedence and the request must not be executed.

In the user interface, `accept` should be red as a warning and `deny` should be green.

For validation, the `operatorId` may be compared to an access control list and the `authorizationTime` may be compared to the local clock.

This object must be published regardless of where the authorization occurs. For example, authorization may occur at the MC, at another remote user interface, or at a local user interface.

B.24.2.3 `tms::AuthorizationToEnergizeResult`

PURPOSE: Report the result of a request that required authorization.

TOPIC USAGE: AuthorizationToEnergizeResult

EXTENSIBILITY: extensibility(APPENDABLE)

PATTERN: Structure

ATTRIBUTES:

Name	Type and Description
`relatedRequestId`	`GridRequest` Copy of the corresponding AuthorizationToEnergizeRequest.requestId. Annotations: *keyval*
`relatedSequenceId`	`RequestSequence` Copy of the corresponding AuthorizationToEnergizeRequest.sequenceId.
`authorizationDeviceId`	`optional<Identity>` Copy of AuthorizationToEnergizeReply.deviceId if a response was received, else absent.
`energizeRequestingDeviceId`	`Identity` Copy of the corresponding AuthorizationToEnergizeRequest.energizeRequestingDeviceId.

`energizeSequenceId`
: `tms::RequestSequence`
Copy of the corresponding AuthorizationToEnergizeRequest.energizeSequenceId.

`userId`
: `optional<Identity>`
Copy of the AuthorizationToEnergizeReply.userId if a response was received and the field was present, else absent.

`accepted`
: `boolean`
True if the authorization was accepted, the request will be executed, and both relatedRequestId and relatedSequenceId are valid.

`responseReceived`
: `boolean`
True confirms that an AuthorizationToEnergizeReply was received. False indicates a response due to authorization timeout, and therefore authorizationDeviceId is absent.

`energizeRequestValid`
: `boolean`
True confirms that energizeRequestingDeviceId, energizeSequenceId, and powerPorts in the AuthorizationToEnergizeReply matched a pending energize request. False otherwise.

`authorizationReviewValid`
: `boolean`
True confirms that the AuthorizationToEnergizeReply.userId and AuthorizationToEnergizeReply.timeOfReview are valid. False otherwise.

`outcomeMonotonic`
: `ClockMonotonic`
Local monotonic clock when the outcome decision was made.

`outcomeRealtime`
: `ClockRealtime`
Local realtime clock when the outcome decision was made.

DESCRIPTION:

This object is sent in response to an authorization request or the lack thereof. It documents key details surrounding the authorization and resulting action. If the authorization is `accepted`, it documents that all required checks passed.

B.24.2.4 `tms::GridRequest`

See definition in Section B.11.2.5.

B.24.2.5 `tms::Identity`

See definition in Section B.2.2.2.

B.24.2.6 `tms::RequestSequence`

See definition in Section B.2.2.5.

B.24.2.7 `tms::PowerPortNumberSequence`

See definition in Section B.5.2.51.

B.24.2.8 `tms::PowerPortNumber`

See definition in Section B.2.2.4.

B.24.2.9 `tms::ClockRealtime`

See definition in Section B.23.2.5.

B.24.2.10 `tms::ClockMonotonic`

See definition in Section B.8.2.3.

This page intentionally left blank.

Sections B.25 through B.99 have been reserved for future versions of this standard.

B.100 Data Type Index

This section contains alphabetical listings of all data types, with references to their definition. It contains one table for all namespaces, followed by separate tables for each namespace. Listings of all constants are also provided after the data types.

B.100.1 Data Types in All Namespaces

Each entry is of the format "type, namespace, page".

ActiveConversionInfo, tms, 163
ActiveDiagnosticState, tms, 216
ActiveMicrogridControllerState, tms, 225
AuthorizationToEnergizeReply, tms, 308
AuthorizationToEnergizeRequest, tms, 307
AuthorizationToEnergizeResult, tms, 309

CableSenseStatus, tms, 254
CircuitContinuity, tms, 239
CircuitWiring, tms, 153
ClockMonotonic, tms, 188
ClockMonotonicPersistenceType, tms, 299
ClockRealtime, tms, 300
ClockSource, tms, 300
ClockState, tms, 298
ConfigId, tms, 120
ControlCurve, tms, 287
ControlHardwareInfo, tms, 134
ControlHardwareUpdate, tms, 187
ControlParameterRequest, tms, 229
ControlParameterState, tms, 229
ControlServiceInfo, tms, 147
ConversionFeature, tms, 163
ConversionFeatureSequence, tms, 162
ConversionInfo, tms, 162
ConversionTapInfo, tms, 171
ConversionTapInfoSequence, tms, 170
Curve2D, tms, 165

DesiredCircuitContinuity, tms, 242
DeviceConfigRequest, tms, 232
DeviceIcon, tms, 184
DeviceInfo, tms, 129
DeviceIntent, tms, 212
DeviceIntentSequence, tms, 212
DeviceRequest, tms, 182
DeviceRole, tms, 130
DiagnosticTroubleCode, tms, 217
DiagnosticTroubleCodeSequence, tms, 216
DistributionFeature, tms, 172
DistributionFeatureSequence, tms, 172
DistributionInfo, tms, 172
DtcSeverity, tms, 223

EnergyStartStopLevel, tms, 175
EnergyStartStopLevelSequence, tms, 174
EnergyStartStopRequest, tms, 247
EnergyStartStopState, tms, 246
EnergyStorageInfo, tms, 139
EnergyStorageState, tms, 194
EngineInfo, tms, 136
EngineOperatingState, tms, 192
EngineState, tms, 191
EnumLabelSequence, tms, 147

FailureModeIndicator, tms, 221
FuelInfo, tms, 138
FuelState, tms, 193

GeneratorInfo, tms, 138
GeneratorState, tms, 194
GlobalTradeItemNumber, tms, 132
GridRequest, tms, 206
GroundFaultDetection, tms, 241
GroundingCircuitNumber, tms, 161
GroundingCircuitRequest, tms, 236
GroundingCircuitState, tms, 235
GroundingCircuitStateSequence, tms, 237
GroundingCircuitStatus, tms, 237
GroundingDesignType, tms, 161

GroundingInfo, tms, 160
GroundingInfoSequence, tms, 159

Heartbeat, tms, 126

Identity, tms, 120
IdentityNicknameRequest, tms, 181
IdentityNicknameState, tms, 181
IdentitySequence, tms, 212

LoadFeature, tms, 178
LoadFeatureSequence, tms, 178
LoadInfo, tms, 177
LoadSharingInfo, tms, 164
LoadSharingInfoSequence, tms, 164
LoadSharingIntent, tms, 209
LoadSharingParameters, tms::ac, 285
LoadSharingParameters, tms::dc, 295
LoadSharingParameterSequence, tms::ac, 284
LoadSharingParameterSequence, tms::dc, 294
LoadSharingRequest, tms::ac, 283
LoadSharingRequest, tms::dc, 293
LoadSharingState, tms::ac, 282
LoadSharingState, tms::dc, 293
LoadStepResponse, tms, 168
LoadStepResponseSequence, tms, 167

MeasurementUpdate, tms::ac, 258
MeasurementUpdate, tms::dc, 267
MetricParameterState, tms, 230
MicrogridControllerFeature, tms, 148
MicrogridControllerFeatureSequence, tms, 148
MicrogridControllerInfo, tms, 147
MicrogridDashboardFeature, tms, 149
MicrogridDashboardFeatureSequence, tms, 149
MicrogridDashboardInfo, tms, 149
MicrogridIntent, tms, 207

NatoStockNumber, tms, 132

OctetSequence, tms, 185
OperatingMode, tms, 208
OperatorIntent, tms, 206
OperatorIntentRequest, tms, 205
OperatorIntentState, tms, 204
OperatorIntentType, tms, 207
OperatorPriority, tms, 210
OperatorPriorityType, tms, 210

ParameterMetadata, tms, 145
ParameterMetadataSequence, tms, 144
ParameterValue, tms, 231
ParameterValueSequence, tms, 231
PassiveConversionInfo, tms, 169
PhaseAngleSequence, tms::ac, 264
Point2D, tms, 167
Point2DSequence, tms, 167
PowerConnection, tms, 275
PowerConnectionDetectionType, tms, 276
PowerConnectionId, tms, 276
PowerConnectionSequence, tms, 274
PowerConnectionState, tms, 274
PowerConnectorFeature, tms, 155
PowerConnectorFeatureSequence, tms, 155
PowerConnectorInfo, tms, 154
PowerConnectorPolarity, tms, 157
PowerConnectorType, tms, 156
PowerDeviceInfo, tms, 150
PowerElectronicsInfo, tms, 140
PowerElectronicsState, tms, 197
PowerHardwareInfo, tms, 135
PowerHardwareUpdate, tms, 190
PowerLineMeasurement, tms::ac, 263
PowerLineMeasurement, tms::dc, 270
PowerLineMeasurementSequence, tms::ac, 262
PowerLineMeasurementSequence, tms::dc, 270
PowerPortConfigRequest, tms, 255
PowerPortConversionInfo, tms, 170
PowerPortConversionInfoSequence, tms, 170
PowerPortDirectionality, tms, 154
PowerPortId, tms, 254
PowerPortInfo, tms, 151
PowerPortInfoSequence, tms, 151
PowerPortIntent, tms, 213

PowerPortIntentSequence, tms, 213
PowerPortMeasurement, tms::ac, 260
PowerPortMeasurement, tms::dc, 268
PowerPortMeasurementSequence, tms::ac, 259
PowerPortMeasurementSequence, tms::dc, 268
PowerPortNumber, tms, 121
PowerPortNumberSequence, tms, 162
PowerPortState, tms, 252
PowerPortStatus, tms, 253
PowerPortStatusSequence, tms, 253
PowerSwitchCloseConditions, tms, 249
PowerSwitchFeature, tms, 158
PowerSwitchFeatureSequence, tms, 158
PowerSwitchInfo, tms, 157
PowerSwitchReason, tms, 239
PowerSwitchRequest, tms, 252
PowerSwitchStatus, tms, 238
PowerSwitchTransitionCode, tms, 240
ProductInfo, tms, 131

Reply, tms, 118
ReplyCode, tms, 122
ReplyStatus, tms, 122
RequestSequence, tms, 121
SourceFeature, tms, 174
SourceFeatureSequence, tms, 173
SourceInfo, tms, 173
StorageCellNumber, tms, 197
StorageCellState, tms, 196
StorageCellStateSequence, tms, 195
StorageFeature, tms, 177
StorageFeatureSequence, tms, 176
StorageInfo, tms, 176
StorageIntent, tms, 211
StorageUpdate, tms, 200
String1-32, tms, 141
String16, tms, 146
String32, tms, 124
SuspectParameterNumber, tms, 219
SynchronousMachineCoefficients, tms, 142

TapNumber, tms, 171
ThermalInfo, tms, 141
ThermalLoadSequence, tms, 198
ThermalState, tms, 198
ThermalZoneSequence, tms, 141
TopicInfo, tms, 133
TopicList, tms, 134
TopicName, tms, 134

B.100.2 Data Types in Namespace tms

ActiveConversionInfo, 163
ActiveDiagnosticState, 216
ActiveMicrogridControllerState, 225
AuthorizationToEnergizeReply, 308
AuthorizationToEnergizeRequest, 307
AuthorizationToEnergizeResult, 309

CableSenseStatus, 254
CircuitContinuity, 239
CircuitWiring, 153
ClockMonotonic, 188
ClockMonotonicPersistenceType, 299
ClockRealtime, 300
ClockSource, 300
ClockState, 298
ConfigId, 120
ControlCurve, 287
ControlHardwareInfo, 134
ControlHardwareUpdate, 187
ControlParameterRequest, 229
ControlParameterState, 229
ControlServiceInfo, 147
ConversionFeature, 163
ConversionFeatureSequence, 162
ConversionInfo, 162
ConversionTapInfo, 171
ConversionTapInfoSequence, 170
Curve2D, 165

DesiredCircuitContinuity, 242
DeviceConfigRequest, 232
DeviceIcon, 184

DeviceInfo, 129
DeviceIntent, 212
DeviceIntentSequence, 212
DeviceRequest, 182
DeviceRole, 130
DiagnosticTroubleCode, 217
DiagnosticTroubleCodeSequence, 216
DistributionFeature, 172
DistributionFeatureSequence, 172
DistributionInfo, 172
DtcSeverity, 223

EnergyStartStopLevel, 175
EnergyStartStopLevelSequence, 174
EnergyStartStopRequest, 247
EnergyStartStopState, 246
EnergyStorageInfo, 139
EnergyStorageState, 194
EngineInfo, 136
EngineOperatingState, 192
EngineState, 191
EnumLabelSequence, 147

FailureModeIndicator, 221
FuelInfo, 138
FuelState, 193

GeneratorInfo, 138
GeneratorState, 194
GlobalTradeItemNumber, 132
GridRequest, 206
GroundFaultDetection, 241
GroundingCircuitNumber, 161
GroundingCircuitRequest, 236
GroundingCircuitState, 235
GroundingCircuitStateSequence, 237
GroundingCircuitStatus, 237
GroundingDesignType, 161
GroundingInfo, 160
GroundingInfoSequence, 159

Heartbeat, 126

Identity, 120
IdentityNicknameRequest, 181
IdentityNicknameState, 181
IdentitySequence, 212

LoadFeature, 178
LoadFeatureSequence, 178
LoadInfo, 177
LoadSharingInfo, 164
LoadSharingInfoSequence, 164
LoadSharingIntent, 209
LoadStepResponse, 168
LoadStepResponseSequence, 167

MetricParameterState, 230
MicrogridControllerFeature, 148
MicrogridControllerFeatureSequence, 148
MicrogridControllerInfo, 147
MicrogridDashboardFeature, 149
MicrogridDashboardFeatureSequence, 149
MicrogridDashboardInfo, 149
MicrogridIntent, 207

NatoStockNumber, 132

OctetSequence, 185
OperatingMode, 208
OperatorIntent, 206
OperatorIntentRequest, 205
OperatorIntentState, 204
OperatorIntentType, 207
OperatorPriority, 210
OperatorPriorityType, 210

ParameterMetadata, 145
ParameterMetadataSequence, 144
ParameterValue, 231
ParameterValueSequence, 231
PassiveConversionInfo, 169
Point2D, 167
Point2DSequence, 167
PowerConnection, 275
PowerConnectionDetectionType, 276
PowerConnectionId, 276
PowerConnectionSequence, 274
PowerConnectionState, 274
PowerConnectorFeature, 155
PowerConnectorFeatureSequence, 155
PowerConnectorInfo, 154

PowerConnectorPolarity, 157
PowerConnectorType, 156
PowerDeviceInfo, 150
PowerElectronicsInfo, 140
PowerElectronicsState, 197
PowerHardwareInfo, 135
PowerHardwareUpdate, 190
PowerPortConfigRequest, 255
PowerPortConversionInfo, 170
PowerPortConversionInfoSequence, 170
PowerPortDirectionality, 154
PowerPortId, 254
PowerPortInfo, 151
PowerPortInfoSequence, 151
PowerPortIntent, 213
PowerPortIntentSequence, 213
PowerPortNumber, 121
PowerPortNumberSequence, 162
PowerPortState, 252
PowerPortStatus, 253
PowerPortStatusSequence, 253
PowerSwitchCloseConditions, 249
PowerSwitchFeature, 158
PowerSwitchFeatureSequence, 158
PowerSwitchInfo, 157
PowerSwitchReason, 239
PowerSwitchRequest, 252
PowerSwitchStatus, 238
PowerSwitchTransitionCode, 240
ProductInfo, 131
Reply, 118
ReplyCode, 122
ReplyStatus, 122
RequestSequence, 121

SourceFeature, 174
SourceFeatureSequence, 173
SourceInfo, 173
StorageCellNumber, 197
StorageCellState, 196
StorageCellStateSequence, 195
StorageFeature, 177
StorageFeatureSequence, 176
StorageInfo, 176
StorageIntent, 211
StorageUpdate, 200
String1-32, 141
String16, 146
String32, 124
SuspectParameterNumber, 219
SynchronousMachineCoefficients, 142

TapNumber, 171
ThermalInfo, 141
ThermalLoadSequence, 198
ThermalState, 198
ThermalZoneSequence, 141
TopicInfo, 133
TopicList, 134
TopicName, 134

B.100.3 Data Types in Namespace tms::ac

LoadSharingParameters, 285
LoadSharingParameterSequence, 284
LoadSharingRequest, 283
LoadSharingState, 282

MeasurementUpdate, 258

PhaseAngleSequence, 264
PowerLineMeasurement, 263
PowerLineMeasurementSequence, 262
PowerPortMeasurement, 260
PowerPortMeasurementSequence, 259

B.100.4 Data Types in Namespace tms::dc

LoadSharingParameters, 295
LoadSharingParameterSequence, 294
LoadSharingRequest, 293
LoadSharingState, 293

MeasurementUpdate, 267

PowerLineMeasurement, 270
PowerLineMeasurementSequence, 270
PowerPortMeasurement, 268
PowerPortMeasurementSequence, 268

B.100.5 Defined Constants

INVALID_GROUNDING_CIRCUIT_NUMBER, 161
INVALID_PORT_NUMBER, 121

MAX_GROUNDING_CIRCUITS, 161
MAX_PARAMETERS, 146
MAX_PORTS, 121
MAX_TAPS, 171
MAXLEN_storageCells, 197

ONLY_CELL, 197

SPN_J1939_EMERGENCY_SHUTDOWN_SWITCH, 219
SPN_J1939_ENGINE_COOLANT_TEMPERATURE, 219
SPN_J1939_ENGINE_OIL_PRESSURE_1, 219
SPN_J1939_ENGINE_SHUTDOWN_OVERRIDE_SWITCH, 219
SPN_J1939_FUEL_LEVEL_1, 219
SPN_J1939_KEY_SWITCH_BATTERY_POTENTIAL, 219
SPN_RANGE_J1939, 219
SPN_RANGE_PROPRIETARY, 219
SPN_RANGE_TMS, 219
SPN_TMS_CONTROL_LOCKOUT_LOCAL, 220
SPN_TMS_CONTROL_LOCKOUT_REMOTE, 220
SPN_TMS_TRIP_AMPERAGE, 220
SPN_TMS_TRIP_ENERGY, 220
SPN_TMS_TRIP_FREQUENCY, 220
SPN_TMS_TRIP_GROUND, 220
SPN_TMS_TRIP_POWER, 220
SPN_TMS_TRIP_TEMPERATURE, 221
SPN_TMS_TRIP_VOLTAGE, 220

C. Cybersecurity Requirements

This section reserved for future use. Refer to Section D.6 for DDS Security integration.

D. OMG DDS Requirements

These requirements were defined using the following Object Management Group standards.

Table D.1: Referenced OMG DDS Standards

Name	Acronym	Version	Adoption Date
Data Distribution Service	DDS	1.4	March 2015
DDS Interoperability Wire Protocol	DDSI-RTPS	2.5	April 2022
DDS Security	DDS-SECURITY	1.1	July 2018
DDS XML	DDS-XML	1.0	December 2018
Extensible and Dynamic Topic Types for DDS	DDS-XTypes	1.3	February 2020
Interface Definition Language	IDL	4.2	March 2018

OMG DDS products built using other versions of these standards may be compatible. The following sections explain how the data model is implemented using OMG DDS and what modifications may be made to support older implementations.

D.1 Network Interoperability

All OMG DDS participants shall implement the Minimal Network Interoperability Profile as specified in DDS-XTypes section 2.2.1, including Extended CDR Representation Encoding Version 2 (XCDR2).

All OMG DDS participants should implement the Basic Network Interoperability Profile as specified in DDS-XTypes section 2.2.2.

All OMG DDS participants shall set the IPv6 multicast address to `FF03::EFFF:1`.

All OMG DDS participants shall set the Type Consistency Enforcement QoS Policy as specified in DDS-XTypes section 7.6.3.4 to the values given in Table D.2.

Table D.2: TypeConsistencyEnforcementQosPolicy Settings

Option	Value
kind	`ALLOW_TYPE_COERCION`
ignore_sequence_bounds	`true`
ignore_string_bounds	`true`
ignore_member_names	`true`
prevent_type_widening	`false`
force_type_validation	`true`

All OMG DDS participants that implement Section D.6 shall implement the Builtin Plugin Interoperability as specified in DDS-SECURITY section 2.2.

ACK_XTYPES_WARNING Two IDL files are attached to this standard in Section E.1.

The default IDL is `mil-std-3071_data_model.idl` and contains the `@try_construct` annotations needed for Section A.9.11.1.

The alternative IDL is `mil-std-3071_data_model_xtypes_warning.idl` and removes the annotations for OMG DDS implementations that do not support them. Users of this file must disable the `ACK_XTYPES_WARNING` error within it and follow vendor-specific instructions to enable type extensibility for enums and sequences equivalent to `@try_construct(USE_DEFAULT)` and `@try_construct(TRIM)` in the default file.

Implementation Note: *Many interoperability settings cannot be configured by the machine-readable IDL and QoS XML files distributed by TMS. These settings must instead be configured using vendor-specific instructions. Please verify that your TMS device properly implements the checklist of OMG DDS interoperability settings in Figure D.1.*

1. OMG DDS interoperability standards conformance
2. IPv6 enabled on the appropriate external ethernet interface
3. IPv6 multicast address is `FF03::EFFF:1`
4. XCDR2 data representation
5. QoS `USER_DATA` is `deviceID` (see Section D.5)
6. QoS `OWNERSHIP_STRENGTH` is 1 or more (see Section D.5)
7. Memory buffer sizes large enough for data samples on all topics
8. For the alternative IDL, enable type extensibility for enums and sequences equivalent to `@try_construct(USE_DEFAULT)` and `@try_construct(TRIM)`

Figure D.1: Checklist of OMG DDS interoperability settings

D.2 Domains

All OMG DDS participants shall connect to domain ID 0 (zero) by default. A single domain participant may be used to create data readers and writers for all topics.

Other domain IDs may be used for development or future deployments. Any TMS device that can use a domain other than 0 shall provide a means for the operator to view and change the domain ID, without requiring external hardware or software.

Incorrect domain ID selection could prevent devices from communicating, even though they are on the same physical network.

D.3 Topics

OMG DDS participants shall use the topic names specified in Chapter B.

D.4 IDL

The OMG Interface Definition Language (IDL) is used to provide a machine-readable definition of the data types defined by this standard. All OMG DDS participants shall use the IDL definitions provided in Chapter E and described in Chapter B. Unless otherwise stated, the only allowed modifications are changes that do not impact interoperability. It is permissible to remove data types that are not referenced by any topics that a device implements.

D.4.1 Data Types

Table D.3 maps TMS data model types to OMG IDL version 3 and IDL version 4.2 syntax.

Table D.3: IDL Type Mapping

Data Model	IDL 3	IDL 4.2
boolean	boolean	boolean
char	char	char
octet	octet	octet
int8	char	int8
int16	short	int16
int32	long	int32
int64	long long	int64
uint8	octet	uint8
uint16	unsigned short	uint16
uint32	unsigned long	uint32
uint64	unsigned long long	uint64
float32	float	float
float64	double	double
Enumeration	enum	enum
Constant	const	const
Structure	struct	struct
`sequence<T,min,max>`	sequence<T,max>	sequence<T,max>
`sequence<char,min,max>`	string<max>	string<max>
`optional<T>`	*not supported*	@optional T
Typedef	typedef	typedef

The official TMS IDL file targets IDL 4.2 syntax. It may be modified to support IDL 3 or non-standard DDS vendor extensions as long as interoperability with devices using the official release file is preserved.

D.4.2 Annotations

OMG IDL provides a number of options for annotating data types. This section describes the annotations used by the standard TMS IDL file. It also describes what modifications may be made to support DDS implementations that do not support them.

D.4.2.1 Extensibility

OMG IDL provides multiple options for annotating types. The official TMS IDL file will contain the following annotations.

Table D.4: IDL Annodations for Extensibility

Annotation	Meaning
`@extensibility(FINAL)`	No changes
`@extensibility(APPENDABLE)`	Values may be appended
`@extensibility(MUTABLE)`	Values may be appended or modified

These annotations change the DDS serialization and must not be removed. Conformance testing will verify that devices interopate with appendable and mutable changes.

D.4.2.2 Topic Usage

IDL annotations are used to indicate whether each data type is used directly or indirectly by one or more topics.

Table D.5: IDL Annotations for Topic Usage

Annotation	Meaning
`@topic("Foo")`	Type is assigned to topic `Foo`.
`@nested`	Type is used indirectly by one or more topics.

It is expected that devices may remove these annotation with no impact to interoperability.

Note: The `@topic` annotation does not appear in IDL version 4.2. It is defined in DDS-XTypes, version 1.3, clause 7.3.1.2.1.16.

D.4.2.3 Key Values

The IDL `@key` annotation is used to indicate key values.

This annotation may be converted into equivalent vendor-specific syntax. Key values should not be removed entirely, as this may increase the DDS network traffic. Conformance testing will verify interoperability with devices that do use key values.

D.4.2.4 Bounded Values

IDL annotations are used to indicate when valid values are bounded by less than the full numeric range. The annotation `@min` indicates a lower bound, `@max` indicates an upper bound, and `@range` indicates both bounds.

Table D.6: IDL Annotations for Bounded Values

Annotation	Meaning
`@min(k1)`	$k_1 \leq X$
`@max(k2)`	$X \leq k_2$
`@range(min=k1, max=k2)`	$k_1 \leq X \leq k_2$

It is expected that devices may remove these annotation with no impact to interoperability. Conformance testing will verify that devices publish values within this range, and subscribers flag out-of-range values, but not whether these annotations are used.

D.4.2.5 Units of Measure

The IDL `@unit` annotation is used to indicate the units of measure for numeric values.

Note: The IDL standard recommends using standardized abbreviations defined by BIPM whenever applicable.

It is expected that devices may remove this annotation with no impact to interoperability. Conformance testing will verify that device use the proper units of measurement but not whether this annotation is used.

D.5 Quality of Service

All devices shall use the following Quality of Service (QoS) profile definitions. These specify values for all QoS policies in the OMG DDS standard. Procurement documents may specify additional implementation-specific values.

Conformance testing will verify that devices under test interoperate with other devices using the specified values.

OMG DDS implementation may require additional, vendor-specific QoS settings. These settings may be chosen to meet the needs of the specific TMS device as long as interoperability with other devices using the standard TMS QoS settings is preserved.

All QoS policies have a `USER_DATA` value of `deviceId` that is intended to simplify tracking of remote participants, especially before the first data sample is sent. This value is a copy of `tms::DeviceInfo.deviceId`. This value must be configured in all DataWriters, possibly by configuring it in the DomainParticipant to be inherited by DataWriters.

All QoS policies have an `OWNERSHIP_STRENGTH` value that is specified as increasing. Each device shall choose a value that is greater than or equal to 1, set this value in the QoS for all published topics, and increase this value each time the device restarts. This is necessary to ensure that the `EXCLUSIVE` ownership does not filter out sample updates that are published before subscribers time out the previous DataWriter or its DomainParticipant. It is recommended to store the strength as a persistent counter in a file, battery-backed memory, or non-volatile memory. Devices with battery-backed clocks may instead calculate strength from the time, as long as time changes and rollover do not cause the value to decrease.

Change Note: OWNERSHIP_STRENGTH was previous set to a fixed value of 1. This change to increasing values is backwards compatible.

Devices may use the OMG DDS XML file provided in Section E.2.

D.5.1 PublishLast Profile

Table D.7: OMG DDS QoS Policies for the PublishLast Profile

QoS Policy	Notes	Value
USER_DATA		copy of `tms::DeviceInfo.deviceId`
TOPIC_DATA	default	zero-length sequence
GROUP_DATA	default	zero-length sequence
DURABILITY	default, 1	TRANSIENT_LOCAL
DURABILITY_SERVICE	2	N/A
PRESENTATION	default	access_scope=INSTANCE
	default	coherent_access=false
	default	ordered_access=false
DEADLINE	default	INFINITE_SEC, INFINITE_NSEC
LATENCY_BUDGET	default	0
OWNERSHIP		EXCLUSIVE
OWNERSHIP_STRENGTH	3	≥ 1, increase on each restart
LIVELINESS	default, 4	kind=AUTOMATIC
	default, 4	lease_duration=INFINITE_SEC, INFINITE_NSEC
TIME_BASED_FILTER	default, 5	0
PARTITION	default	zero-length sequence
RELIABILITY	6	kind=RELIABLE
	7	max_blocking_time = application specific
TRANSPORT_PRIORITY	default	0
LIFESPAN	default	INFINITE_SEC, INFINITE_NSEC
DESTINATION_ORDER	default	BY_RECEPTION_TIMESTAMP
HISTORY	default	kind=KEEP_LAST
	default	depth=1
RESOURCE_LIMITS	default, 8	max_samples = UNLIMITED
	default, 8	max_instances = UNLIMITED
	default, 8	max_samples_per_instance = UNLIMITED
ENTITY_FACTORY	7	application specific
WRITER_DATA_LIFECYCLE	7	application specific, default TRUE
READER_DATA_LIFECYCLE	7	application specific

NOTES:

0. Default. This is the default value in the OMG DDS standard. Set it explicitly or verify that it is the default in your implementation.

1. DURABILITY. It is expected that the DataWriter lifespan will match the operating time of the device. DataWriters should not be created and destroyed for each sample update.

2. DURABILITY_SERVICE. This value is not used with the selected DURABILITY.

3. OWNERSHIP_STRENGTH. This value must increase every time the device starts, as specified in Section D.5.

4. LIVELINESS. The publisher may choose to use a MANUAL kind and a shorter lease duration without affecting interoperability.

5. TIME_BASED_FILTER. The subscriber may choose a longer minimum separation without affecting interoprability.

6. RELIABILITY kind. Subscribers may select BEST_EFFORT under some circumstances. For example, an MD may not require reliable delivery of power measurements.

7. RELIABILITY max blocking time, FACTORY, and LIFECYCLE. These values are application-specific and may be changed without affecting interoperability.

8. RESOURCE_LIMITS. These values may be decreased by the publisher or subscriber, subject to the requirements in Section A.10.

D.5.2 Rare Profile

Table D.8: OMG DDS QoS Policies for the Rare Profile

QoS Policy	Notes	Value
USER_DATA		copy of `tms::DeviceInfo.deviceId`
TOPIC_DATA	default	zero-length sequence
GROUP_DATA	default	zero-length sequence
DURABILITY	1	VOLATILE
DURABILITY_SERVICE	2	N/A
PRESENTATION	default	access_scope=INSTANCE
	default	coherent_access=false
	default	ordered_access=false
DEADLINE		2000 sec, 0 nanosec
LATENCY_BUDGET	default	0
OWNERSHIP		EXCLUSIVE
OWNERSHIP_STRENGTH	3	≥ 1, increase on each restart
LIVELINESS	default, 4	kind=AUTOMATIC
	default, 4	lease_duration=INFINITE_SEC, INFINITE_NSEC
TIME_BASED_FILTER	default, 5	0
PARTITION	default	zero-length sequence
RELIABILITY	6	kind=BEST_EFFORT
	7	max_blocking_time = application specific
TRANSPORT_PRIORITY	default	0
LIFESPAN	default	INFINITE_SEC, INFINITE_NSEC
DESTINATION_ORDER	default	BY_RECEPTION_TIMESTAMP
HISTORY	default	kind=KEEP_LAST
	default	depth=1
RESOURCE_LIMITS	default, 8	max_samples = UNLIMITED
	default, 8	max_instances = UNLIMITED
	default, 8	max_samples_per_instance = UNLIMITED
ENTITY_FACTORY	7	application specific
WRITER_DATA_LIFECYCLE	7	application specific, default TRUE
READER_DATA_LIFECYCLE	7	application specific

NOTES:

0. Default. This is the default value in the OMG DDS standard. Set it explicitly or verify that it is the default in your implementation.

1. DURABILITY. It is expected that the DataWriter lifespan will match the operating time of the device. DataWriters should not be created and destroyed for each sample update.

2. DURABILITY_SERVICE. This value is not used with the selected DURABILITY.

3. OWNERSHIP_STRENGTH. This value must increase every time the device starts, as specified in Section D.5.

4. LIVELINESS. The publisher may choose to use a MANUAL kind and a shorter lease duration without affecting interoperability.

5. TIME_BASED_FILTER. The subscriber may choose a longer minimum separation without affecting interoprability.

6. RELIABILITY kind. Publishers should not offer RELIABLE, as this may delay delivery of new sample updates.

7. RELIABILITY max blocking time, FACTORY, and LIFECYCLE. These values are application-specific and may be changed without affecting interoperability.

8. RESOURCE_LIMITS. These values may be decreased by the publisher or subscriber, subject to the requirements in Section A.10.

D.5.3 Slow Profile

Table D.9: OMG DDS QoS Policies for the Slow Profile

QoS Policy	Notes	Value
USER_DATA		copy of `tms::DeviceInfo.deviceId`
TOPIC_DATA	default	zero-length sequence
GROUP_DATA	default	zero-length sequence
DURABILITY	1	VOLATILE
DURABILITY_SERVICE	2	N/A
PRESENTATION	default	access_scope=INSTANCE
	default	coherent_access=false
	default	ordered_access=false
DEADLINE		20 sec, 0 nanosec
LATENCY_BUDGET	default	0
OWNERSHIP		EXCLUSIVE
OWNERSHIP_STRENGTH	3	≥ 1, increase on each restart
LIVELINESS	default, 4	kind=AUTOMATIC
	default, 4	lease_duration=INFINITE_SEC, INFINITE_NSEC
TIME_BASED_FILTER	default, 5	0
PARTITION	default	zero-length sequence
RELIABILITY	6	kind=BEST_EFFORT
	7	max_blocking_time = application specific
TRANSPORT_PRIORITY	default	0
LIFESPAN	default	INFINITE_SEC, INFINITE_NSEC
DESTINATION_ORDER	default	BY_RECEPTION_TIMESTAMP
HISTORY	default	kind=KEEP_LAST
	default	depth=1
RESOURCE_LIMITS	default, 8	max_samples = UNLIMITED
	default, 8	max_instances = UNLIMITED
	default, 8	max_samples_per_instance = UNLIMITED
ENTITY_FACTORY	7	application specific
WRITER_DATA_LIFECYCLE	7	application specific, default TRUE
READER_DATA_LIFECYCLE	7	application specific

NOTES:

0. Default. This is the default value in the OMG DDS standard. Set it explicitly or verify that it is the default in your implementation.

1. DURABILITY. It is expected that the DataWriter lifespan will match the operating time of the device. DataWriters should not be created and destroyed for each sample update.

2. DURABILITY_SERVICE. This value is not used with the selected DURABILITY.

3. OWNERSHIP_STRENGTH. This value must increase every time the device starts, as specified in Section D.5.

4. LIVELINESS. The publisher may choose to use a MANUAL kind and a shorter lease duration without affecting interoperability.

5. TIME_BASED_FILTER. The subscriber may choose a longer minimum separation without affecting interoprability.

6. RELIABILITY kind. Publishers should not offer RELIABLE, as this may delay delivery of new sample updates.

7. RELIABILITY max blocking time, FACTORY, and LIFECYCLE. These values are application-specific and may be changed without affecting interoperability.

8. RESOURCE_LIMITS. These values may be decreased by the publisher or subscriber, subject to the requirements in Section A.10.

D.5.4 Medium Profile

Table D.10: OMG DDS QoS Policies for the Medium Profile

QoS Policy	Notes	Value
USER_DATA		copy of `tms::DeviceInfo.deviceId`
TOPIC_DATA	default	zero-length sequence
GROUP_DATA	default	zero-length sequence
DURABILITY	1	VOLATILE
DURABILITY_SERVICE	2	N/A
PRESENTATION	default	access_scope=INSTANCE
	default	coherent_access=false
	default	ordered_access=false
DEADLINE		3 sec, 0 nanosec
LATENCY_BUDGET	default	0
OWNERSHIP		EXCLUSIVE
OWNERSHIP_STRENGTH	3	≥ 1, increase on each restart
LIVELINESS	default, 4	kind=AUTOMATIC
	default, 4	lease_duration=INFINITE_SEC, INFINITE_NSEC
TIME_BASED_FILTER	default, 5	0
PARTITION	default	zero-length sequence
RELIABILITY	6	kind=BEST_EFFORT
	7	max_blocking_time = application specific
TRANSPORT_PRIORITY	default	0
LIFESPAN	default	INFINITE_SEC, INFINITE_NSEC
DESTINATION_ORDER	default	BY_RECEPTION_TIMESTAMP
HISTORY	default	kind=KEEP_LAST
	default	depth=1
RESOURCE_LIMITS	default, 8	max_samples = UNLIMITED
	default, 8	max_instances = UNLIMITED
	default, 8	max_samples_per_instance = UNLIMITED
ENTITY_FACTORY	7	application specific
WRITER_DATA_LIFECYCLE	7	application specific, default TRUE
READER_DATA_LIFECYCLE	7	application specific

NOTES:

0. Default. This is the default value in the OMG DDS standard. Set it explicitly or verify that it is the default in your implementation.

1. DURABILITY. It is expected that the DataWriter lifespan will match the operating time of the device. DataWriters should not be created and destroyed for each sample update.

2. DURABILITY_SERVICE. This value is not used with the selected DURABILITY.

3. OWNERSHIP_STRENGTH. This value must increase every time the device starts, as specified in Section D.5.

4. LIVELINESS. The publisher may choose to use a MANUAL kind and a shorter lease duration without affecting interoperability.

5. TIME_BASED_FILTER. The subscriber may choose a longer minimum separation without affecting interoprability.

6. RELIABILITY kind. Publishers should not offer RELIABLE, as this may delay delivery of new sample updates.

7. RELIABILITY max blocking time, FACTORY, and LIFECYCLE. These values are application-specific and may be changed without affecting interoperability.

8. RESOURCE_LIMITS. These values may be decreased by the publisher or subscriber, subject to the requirements in Section A.10.

D.5.5 Continuous Profile

Table D.11: OMG DDS QoS Policies for the Continuous Profile

QoS Policy	Notes	Value
USER_DATA		copy of `tms::DeviceInfo.deviceId`
TOPIC_DATA	default	zero-length sequence
GROUP_DATA	default	zero-length sequence
DURABILITY	1	VOLATILE
DURABILITY_SERVICE	2	N/A
PRESENTATION	default	access_scope=INSTANCE
	default	coherent_access=false
	default	ordered_access=false
DEADLINE		2 sec, 0 nanosec
LATENCY_BUDGET	default	0
OWNERSHIP		EXCLUSIVE
OWNERSHIP_STRENGTH	3	≥ 1, increase on each restart
LIVELINESS	default, 4	kind=AUTOMATIC
	default, 4	lease_duration=INFINITE_SEC, INFINITE_NSEC
TIME_BASED_FILTER	default, 5	0
PARTITION	default	zero-length sequence
RELIABILITY	6	kind=BEST_EFFORT
	7	max_blocking_time = application specific
TRANSPORT_PRIORITY	default	0
LIFESPAN	default	INFINITE_SEC, INFINITE_NSEC
DESTINATION_ORDER	default	BY_RECEPTION_TIMESTAMP
HISTORY	default	kind=KEEP_LAST
	default	depth=1
RESOURCE_LIMITS	default, 8	max_samples = UNLIMITED
	default, 8	max_instances = UNLIMITED
	default, 8	max_samples_per_instance = UNLIMITED
ENTITY_FACTORY	7	application specific
WRITER_DATA_LIFECYCLE	7	application specific, default TRUE
READER_DATA_LIFECYCLE	7	application specific

NOTES:

0. Default. This is the default value in the OMG DDS standard. Set it explicitly or verify that it is the default in your implementation.

1. DURABILITY. It is expected that the DataWriter lifespan will match the operating time of the device. DataWriters should not be created and destroyed for each sample update.

2. DURABILITY_SERVICE. This value is not used with the selected DURABILITY.

3. OWNERSHIP_STRENGTH. This value must increase every time the device starts, as specified in Section D.5.

4. LIVELINESS. The publisher may choose to use a MANUAL kind and a shorter lease duration without affecting interoperability.

5. TIME_BASED_FILTER. The subscriber may choose a longer minimum separation without affecting interoprability.

6. RELIABILITY kind. Publishers should not offer RELIABLE, as this may delay delivery of new sample updates.

7. RELIABILITY max blocking time, FACTORY, and LIFECYCLE. These values are application-specific and may be changed without affecting interoperability.

8. RESOURCE_LIMITS. These values may be decreased by the publisher or subscriber, subject to the requirements in Section A.10.

D.5.6 Command Profile

Table D.12: OMG DDS QoS Policies for the Command Profile

QoS Policy	Notes	Value
USER_DATA		copy of `tms::DeviceInfo.deviceId`
TOPIC_DATA	default	zero-length sequence
GROUP_DATA	default	zero-length sequence
DURABILITY	1	VOLATILE
DURABILITY_SERVICE	2	N/A
PRESENTATION	default	access_scope=INSTANCE
	default	coherent_access=false
	default	ordered_access=false
DEADLINE	default	INFINITE_SEC, INFINITE_NSEC
LATENCY_BUDGET	default	0
OWNERSHIP		EXCLUSIVE
OWNERSHIP_STRENGTH	3	≥ 1, increase on each restart
LIVELINESS	default, 4	kind=AUTOMATIC
	default, 4	lease_duration=INFINITE_SEC, INFINITE_NSEC
TIME_BASED_FILTER	default, 5	0
PARTITION	default	zero-length sequence
RELIABILITY	6	kind=RELIABLE
	7	max_blocking_time = application specific
TRANSPORT_PRIORITY	default	0
LIFESPAN	default	INFINITE_SEC, INFINITE_NSEC
DESTINATION_ORDER	default	BY_RECEPTION_TIMESTAMP
HISTORY	default	kind=KEEP_LAST
	default	depth=1
RESOURCE_LIMITS	default, 8	max_samples = UNLIMITED
	default, 8	max_instances = UNLIMITED
	default, 8	max_samples_per_instance = UNLIMITED
ENTITY_FACTORY	7	application specific
WRITER_DATA_LIFECYCLE	7	application specific, default TRUE
READER_DATA_LIFECYCLE	7	application specific

NOTES:

0. Default. This is the default value in the OMG DDS standard. Set it explicitly or verify that it is the default in your implementation.

1. DURABILITY. It is expected that the DataWriter lifespan will match the operating time of the device. DataWriters should not be created and destroyed for each sample update.

2. DURABILITY_SERVICE. This value is not used with the selected DURABILITY.

3. OWNERSHIP_STRENGTH. This value must increase every time the device starts, as specified in Section D.5.

4. LIVELINESS. The publisher may choose to use a MANUAL kind and a shorter lease duration without affecting interoperability.

5. TIME_BASED_FILTER. The subscriber may choose a longer minimum separation without affecting interoprability.

6. RELIABILITY kind. Subscribers may select BEST_EFFORT under some circumstances. For example, an MD may not require reliable delivery of power measurements.

7. RELIABILITY max blocking time, FACTORY, and LIFECYCLE. These values are application-specific and may be changed without affecting interoperability.

8. RESOURCE_LIMITS. These values may be decreased by the publisher or subscriber, subject to the requirements in Section A.10.

D.5.7 Response Profile

Table D.13: OMG DDS QoS Policies for the Response Profile

QoS Policy	Notes	Value
USER_DATA		copy of `tms::DeviceInfo.deviceId`
TOPIC_DATA	default	zero-length sequence
GROUP_DATA	default	zero-length sequence
DURABILITY	1	VOLATILE
DURABILITY_SERVICE	2	N/A
PRESENTATION	default	access_scope=INSTANCE
	default	coherent_access=false
	default	ordered_access=false
DEADLINE	default	INFINITE_SEC, INFINITE_NSEC
LATENCY_BUDGET	default	0
OWNERSHIP		EXCLUSIVE
OWNERSHIP_STRENGTH	3	≥ 1, increase on each restart
LIVELINESS	default, 4	kind=AUTOMATIC
	default, 4	lease_duration=INFINITE_SEC, INFINITE_NSEC
TIME_BASED_FILTER	default, 5	0
PARTITION	default	zero-length sequence
RELIABILITY	6	kind=RELIABLE
	7	max_blocking_time = application specific
TRANSPORT_PRIORITY	default	0
LIFESPAN	default	INFINITE_SEC, INFINITE_NSEC
DESTINATION_ORDER	default	BY_RECEPTION_TIMESTAMP
HISTORY	default	kind=KEEP_LAST
	default	depth=1
RESOURCE_LIMITS	default, 8	max_samples = UNLIMITED
	default, 8	max_instances = UNLIMITED
	default, 8	max_samples_per_instance = UNLIMITED
ENTITY_FACTORY	7	application specific
WRITER_DATA_LIFECYCLE	7	application specific, default TRUE
READER_DATA_LIFECYCLE	7	application specific

NOTES:

0. Default. This is the default value in the OMG DDS standard. Set it explicitly or verify that it is the default in your implementation.

1. DURABILITY. It is expected that the DataWriter lifespan will match the operating time of the device. DataWriters should not be created and destroyed for each sample update.

2. DURABILITY_SERVICE. This value is not used with the selected DURABILITY.

3. OWNERSHIP_STRENGTH. This value must increase every time the device starts, as specified in Section D.5.

4. LIVELINESS. The publisher may choose to use a MANUAL kind and a shorter lease duration without affecting interoperability.

5. TIME_BASED_FILTER. The subscriber may choose a longer minimum separation without affecting interoprability.

6. RELIABILITY kind. Subscribers may select BEST_EFFORT under some circumstances. For example, an MD may not require reliable delivery of power measurements.

7. RELIABILITY max blocking time, FACTORY, and LIFECYCLE. These values are application-specific and may be changed without affecting interoperability.

8. RESOURCE_LIMITS. These values may be decreased by the publisher or subscriber, subject to the requirements in Section A.10.

D.5.8 Reply Profile

Table D.14: OMG DDS QoS Policies for the Reply Profile

QoS Policy	Notes	Value
USER_DATA		copy of `tms::DeviceInfo.deviceId`
TOPIC_DATA	default	zero-length sequence
GROUP_DATA	default	zero-length sequence
DURABILITY	1	VOLATILE
DURABILITY_SERVICE	2	N/A
PRESENTATION	default	access_scope=INSTANCE
	default	coherent_access=false
	default	ordered_access=false
DEADLINE	default	INFINITE_SEC, INFINITE_NSEC
LATENCY_BUDGET	default	0
OWNERSHIP		EXCLUSIVE
OWNERSHIP_STRENGTH	3	≥ 1, increase on each restart
LIVELINESS	default, 4	kind=AUTOMATIC
	default, 4	lease_duration=INFINITE_SEC, INFINITE_NSEC
TIME_BASED_FILTER	default, 5	0
PARTITION	default	zero-length sequence
RELIABILITY	6	kind=RELIABLE
	7	max_blocking_time = application specific
TRANSPORT_PRIORITY	default	0
LIFESPAN	default	INFINITE_SEC, INFINITE_NSEC
DESTINATION_ORDER	default	BY_RECEPTION_TIMESTAMP
HISTORY	default	kind=KEEP_LAST
	8	depth=128
RESOURCE_LIMITS	default, 9	max_samples = UNLIMITED
	default, 9	max_instances = UNLIMITED
	default, 9	max_samples_per_instance = UNLIMITED
ENTITY_FACTORY	7	application specific
WRITER_DATA_LIFECYCLE	7	application specific, default TRUE
READER_DATA_LIFECYCLE	7	application specific

NOTES:

0. Default. This is the default value in the OMG DDS standard. Set it explicitly or verify that it is the default in your implementation.

1. DURABILITY. It is expected that the DataWriter lifespan will match the operating time of the device. DataWriters should not be created and destroyed for each sample update.

2. DURABILITY_SERVICE. This value is not used with the selected DURABILITY.

3. OWNERSHIP_STRENGTH. This value must increase every time the device starts, as specified in Section D.5.

4. LIVELINESS. The publisher may choose to use a MANUAL kind and a shorter lease duration without affecting interoperability.

5. TIME_BASED_FILTER. The subscriber may choose a longer minimum separation without affecting interoprability.

6. RELIABILITY kind. Subscribers may select BEST_EFFORT under some circumstances. For example, an MD may not require reliable delivery of power measurements.

7. RELIABILITY max blocking time, FACTORY, and LIFECYCLE. These values are application-specific and may be changed without affecting interoperability.

8. HISTORY depth. The MC may send a single device this many requests before waiting for a reply. If more requests are sent, then some replies may be lost.

9. RESOURCE_LIMITS. These values may be decreased by the publisher or subscriber, subject to the requirements in Section A.10.

D.6 DDS Security

This section is an optional extension to the TMS data model that specifies how the DDS Security architecture integrates into TMS. This allows the DDS Security builtin plugins to be enabled and interoperate, thus protecting the TMS communications against tampering. These protections do not prevent tampering with the devices themselves or attacks against the network itself.

At present, acquisitions programs may ask vendors to future-proof their product by demonstrating that the device controller is capable of running with this extension enabled. However, it is not recommended to require DDS Security for general deployment until the external interfaces (Section D.6.2) are fully defined and implemented.

DDS Security may be deployed in many different settings, and so many different actors may be involved and sequences of events may occur. For convenience, the following actor names and events are assumed in the following explanations. An *acquisitions officer* requires Section D.6 and provide additional requirements for Section D.6.2. A *manufacturer* implements these requirements in their TMS product. A *logistician* generates a public/private key pair for the device, signs it using an identity certificate authority (CA), and installs it on the device. The *Officer in Charge* (OIC) of the microgrid selects which policy should govern the microgrid, selects which devices are authorized to operate, signs these materials using a permissions CA, and installs them on each device in the microgrid.

D.6.1 Data Model Integration

The following data model features support the deep integration of DDS Security.

The *device roles* provide clear rules for role-based access control.

The *deviceId* provides a NAME to be signed in each device's identity certificate.

The *DeviceInfo* provides the device role and key performance parameters. These values may be signed in X.509 certificate extensions and checked to prevent a compromised certificate from being used to impersonate a different device type.

D.6.2 External Interfaces

In order to deploy a complete product, the following interfaces must be implemented. No further specifications are provided for them in this revision of the standard.

Key Management. The device must provide a solution for securely generating and storing a public/private key pair. This may include hardware security modules, trusted platform modules, entropy generators, or other technologies that protect against tampering.

Certificate Management. The device must provide a solution for generating a certificate signing request (CSR) and storing the resulting certificate.

Configuration Management. The device must provide a solution for loading security configuration policies that apply to the entire microgrid.

User Interface. The device must provide a means for the user to select the appropriate security policy and diagnose any operational issues. This interface may provide a means to "pair" the device with a ... This interface should include a "break glass" procedure that allows an operator with physical presence to assert positive control of the device, while leaving a forensic log that evidences what happened in case investigation is required.

D.6.3 DDS Security Configuration

The builtin DDS Security plugins are configured using the `PropertyQosPolicy` of the `DomainParticipantQos`. These properties are used in addition to those defined in Section D.5. These properties are device-specific, may be changed by the external interfaces above, and must be set programmatically before enabling the domain participant.

The **dds.sec.auth.identity_ca** shall be set to the certificate authority (CA) used to sign device identities.

The **dds.sec.auth.private_key** shall be set to the device's private key.

The **dds.sec.auth.password** may be set to provide a password for decrypting the private key.

The **dds.sec.auth.identity_certificate** shall be set to the device's X.509 PEM certificate as signed by the identity CA.

The **dds.sec.access.permissions_ca** shall be set to the CA used to sign microgrid configuration settings.

The **dds.sec.access.governance** shall be set to the microgrid governance file as specified in Section D.6.4 and signed by the permissions CA.

The **dds.sec.access.permissions** shall be set to the device permissions file as specified in Section D.6.5 and signed by the permissions CA.

D.6.4 Microgrid Governance

All devices that are trusted to operate on a given microgrid must have a single, shared *microgrid governance* file. The OIC may install this file at time of operation or select it from a list of pre-installed options. This file specifies what protections should be applied to various portions of the OMG DDS traffic. This file is signed by the permissions CA to prevent tampering.

Three configuration options are defined: Unsecured for disabling security, Secured for data integrity and confidentiality, and Zero Trust for data integrity even if one or more devices are compromised. These options allow the OIC to select protections that are appropriate for their operational risk posture.

Table D.15 defines the configuration settings for these three options. The following paragraphs explain these settings and describe the intended usage and tradeoffs of each configuration.

Table D.15: Microgrid Governance Settings

	Configuration		
XML Element	Unsecured	Secured	Zero Trust
<domain_rule>			
<allow_unauthentiated_participants>	true	false	false
<enable_join_access_control>	false	true	true
<discovery_protection_kind>	NONE	NONE	NONE
<liveliness_protection_kind>	NONE	NONE	NONE
<rtps_protection_kind>	NONE	ENCRYPT	SWOA†
<topic_rule>			
<topic_expression>	*	*	*
<enable_discovery_protection>	false	true	true
<enable_liveliness_protection>	false	true	true
<enable_read_access_control>	false	true	true
<enable_write_access_control>	false	true	true
<metadata_protection_kind>	NONE	NONE	NONE
<data_protection_kind>	NONE	NONE	NONE

† SWOA is short for SIGN_WITH_ORIGIN_AUTHENTICATION.

The <*_protection_kind> settings can be confusing, as most are set to NONE even in the Secured and Zero Trust configurations. Quoting the DDS Security section on the Domain Governance Document, applying protections to the RTPS protocol "is in addition to any protection that may occur for individual submessages and for submessage data payloads." So the RTPS protection actually includes all the other protection kinds, and enabling protection at another point would uselessly double-protect things.

Unsecured Configuration. This configuration disables all DDS Security protections. It is intended to enable interoperability with devices that do not support DDS Security, development activities before DDS Security is turned on, or operating unsecured because DDS Security is not configured properly or malfunctions.

This configuration may be implemented by disabling the DDS Security features entirely or by loading a governance file that has them all disabled.

This configuration will be tested to interoperate with other devices that are set to Unsecured, regardless of permissions CA, or do not support DDS Security.

Secured Configuration. This is the recommended baseline configuration and provides data integrity and data confidentiality with the least computational and network overhead. It does not prevent authorized devices from impersonating each other.

Inconsistent configuration between devices will prevent proper microgrid operation, as devices will be unable to establish shared session keys.

This configuration will be tested to only interoperate with other devices that are set to Secured and use the same governance file.

Zero Trust Configuration. This configuration provides data integrity, transparency for traffic inspection, and prevents authorized devices from impersonating each other. It requires more computational and network overhead than the Secured configuration. It does not provide data confidentiality.

Inconsistent configuration between devices will prevent proper microgrid operation, as devices will be unable to establish shared session keys. However, since the data is signed but not encrypted, it may be possible to operate in a degraded state similar to the Unsecured configuration.

This configuration will be tested to only interoperate with other devices that are set to Zero Trust and use the same governance file.

D.6.5 Device Permissions

Each device that is trusted to operate on a given microgrid must have its own, unique *device permissions* file. The OIC may install this file at time of operation or select it from a list of pre-installed options. This file specifies when the device is allowed to operate and what topics it is allowed to publish or subscribe. This file is signed by the permissions CA to prevent tampering.

Table D.16 defines the configuration settings for these files. The following paragraphs explain these settings and provide usage recommendations.

Table D.16: Device Permissions Settings

XML Element	Value
<grant>	Attribute name=deviceId
<subject_name>	X.509 distinguished name
<validity>	N/A
<not_before>	Authorized start time
<not_after>	Authorized stop time
<allow_rule>	N/A
<domains>	Authorized domain IDs (usually <id> 0)
<publish>	N/A
<topics>	N/a
<topic>	Name of each topic authorized for publication
<subscribe>	N/A
<topics>	N/A
<topic>	Name of each topic authorized for subscription
<default>	DENY

The <subject_name> must match the device's identity certificate.

Recommendations:

1. The deviceId should be a copy of the X.509 distingished name (DN) or else the deviceId be the common name (CN) within the DN.

2. Consider the default time that the device clock assumes after power loss when setting <not_before>.

3. Consider the impact of clock synchronization issues and long-lasting operations when setting <not_after>. An extremely long window may be preferable to denied operations. The CA may be rotated if a certificate is compromised.

4. Authorize all topics in the TopicInfo to avoid malfunction.

E. Machine Readable Files

E.1 IDL

The OMG DDS IDL file for TMS is attached to this PDF as mil-std-3071_data_model.idl. This file i s normative. The following i s a syntax-highlighted, verbatim copy of the file contents. Automatic l ine wraps are i ndicated by a ↩.

An alternative IDL file for OMG DDS tools that do not support `@try construct` is attached to this PDF as mil-std-3071_data_model_xtypes_warning.idl. See Section D.1 for details on its correct usage.

```
/**
 * OMG DDS Interface Definition Language for the Tactical Microgrid Standard (TMS) and distributed with MIL-STD-3071
 * DISTRIBUTION STATEMENT A. Approved for Public Release: Distribution Unlimited
 */

/**
 * This version of the IDL includes the OMG-DDS XTypes @try_construct annotations.
 * Use this version if your OMG-DDS IDL compiler supports the @try_construct annotations.
 * See the MIL-STD-3071 specification, APPENDIX D OMG DDS Requirements for additional details.
 */

/* Value slightly greater than pi, used as a limit for radian measurements. */
#define RADIAN_MAX 3.1416


module tms {

   const string TMS_VERSION = "2.0.3-prerelease2";

   /** TMS Topic Names */
   module topic {

      /**
       * Liveliness heartbeat for all devices.
       * Version: 1.0.0
       * Section: DISCOVERY
       * QoS Profile: Medium
       * Data Trigger: Periodic 1 s
       * Data Type: tms::Heartbeat
       */
      const string TOPIC_HEARTBEAT = "Heartbeat";

      /**
       * Product information for the device. Used to discover TMS compliant devices.
       * Version: 1.0.0
       * Section: DISCOVERY
       * QoS Profile: PublishLast
       * Data Trigger: At Start
       * Data Type: tms::DeviceInfo
       */
      const string TOPIC_DEVICE_INFO = "DeviceInfo";

      /**
       * Device image as an icon.
       * Version: 1.0.0
       * Section: DISCOVERY
       * QoS Profile: PublishLast
       * Data Trigger: At Start
       * Data Type: tms::DeviceIcon
       */
      const string TOPIC_DEVICE_ICON = "DeviceIcon";

      /**
       * Device and Platform Identity Nickname.
       * Version: 1.0.0
       * Section: DISCOVERY
       * QoS Profile: PublishLast
       * Data Trigger: At Start, On Change
```

```
 * Data Type: tms::IdentityNicknameState
 */
const string TOPIC_IDENTITY_NICKNAME_STATE = "IdentityNicknameState";

/**
 * Request to change a Device or Platform Identity Nickname.
 * Version: 1.0.0
 * Section: DISCOVERY
 * QoS Profile: Command
 * Data Trigger: As Needed
 * Data Type: tms::IdentityNicknameRequest
 * Reply Topic: Reply
 * Reply Topic: IdentityNicknameState
 */
const string TOPIC_IDENTITY_NICKNAME_REQUEST = "IdentityNicknameRequest";

/**
 * Power connections manually entered by a microgrid operator.
 * Version: 1.0.0
 * Section: TOPOLOGY
 * QoS Profile: PublishLast
 * Data Trigger: At Start, On Change
 * Data Type: tms::PowerConnectionState
 */
const string TOPIC_OPERATOR_POWER_CONNECTION_STATE = "OperatorPowerConnectionState";

/**
 * Power connections discovered by a power device, usually through a smart cable or operator entry.
 * Version: 1.0.0
 * Section: TOPOLOGY
 * QoS Profile: PublishLast
 * Data Trigger: At Start, On Change
 * Data Type: tms::PowerConnectionState
 */
const string TOPIC_DISCOVERED_POWER_CONNECTION_STATE = "DiscoveredPowerConnectionState";

/**
 * Power connections used by the microgrid controller.  May include operator connections, discovered connections, ↩
     and other connections inferred by the MC.
 * Version: 1.0.0
 * Section: TOPOLOGY
 * QoS Profile: PublishLast
 * Data Trigger: At Start, On Change
 * Data Type: tms::PowerConnectionState
 */
const string TOPIC_MICROGRID_POWER_CONNECTION_STATE = "MicrogridPowerConnectionState";

/**
 * Reports a device diagnostics state including information and alarm states.
 * Version: 1.0.0
 * Section: DIAGNOSTIC
 * QoS Profile: PublishLast
 * Data Trigger: At Start, On Change
 * Data Type: tms::ActiveDiagnosticState
 */
const string TOPIC_ACTIVE_DIAGNOSTIC_STATE = "ActiveDiagnosticState";

/**
 * Reports changes to a device clock(s). A device must support relative time and can optionally support absolute ↩
     time.
 * Version: 1.0.0
 * Section: STATUS
 * QoS Profile: PublishLast
 * Data Trigger: At Start, On Change
 * Data Type: tms::ClockState
 */
const string TOPIC_CLOCK_STATE = "ClockState";

/**
 * The selected master Microgrid Controller for a specific power device.
 * Version: 1.0.0
 * Section: STATUS
 * QoS Profile: PublishLast
 * Data Trigger: At Start, On Change
 * Data Type: tms::ActiveMicrogridControllerState
 */
const string TOPIC_ACTIVE_MICROGRID_CONTROLLER_STATE = "ActiveMicrogridControllerState";

/**
 * Reports power measurements for alternating current power device ports.
 * Version: 1.0.0
 * Section: POWER_DEVICE
 * QoS Profile: Medium
 * Data Trigger: Periodic 1 s
 * Data Type: tms::ac::MeasurementUpdate
 */
const string TOPIC_AC_MEASUREMENT_UPDATE = "AcMeasurementUpdate";

/**
 * Reports dynamic state changes for a power device ports.
 * Version: 1.0.0
 * Section: POWER_DEVICE
```

```
 * QoS Profile: PublishLast
 * Data Trigger: At Start, On Change
 * Data Type: tms::PowerPortState
 */
const string TOPIC_POWER_PORT_STATE = "PowerPortState";

/**
 * Request to change the circuit continuity of a single power port
 * Version: 1.0.0
 * Section: DIST_DEVICE
 * QoS Profile: Command
 * Data Trigger: As Needed
 * Data Type: tms::PowerSwitchRequest
 * Reply Topic: Reply
 * Reply Topic: PowerPortState
 */
const string TOPIC_POWER_SWITCH_REQUEST = "PowerSwitchRequest";

/**
 * Reports the operational state of a Source or Storage device
 * Version: 1.0.0
 * Section: SOURCE_DEVICE
 * QoS Profile: PublishLast
 * Data Trigger: At Start, On Change
 * Data Type: tms::EnergyStartStopState
 */
const string TOPIC_ENERGY_START_STOP_STATE = "EnergyStartStopState";

/**
 * Request to change the EnergyStartStopLevel of a Source or Storage device.
 * Version: 1.0.0
 * Section: SOURCE_DEVICE
 * QoS Profile: Command
 * Data Trigger: As Needed
 * Data Type: tms::EnergyStartStopRequest
 * Reply Topic: Reply
 * Reply Topic: EnergyStartStopState
 */
const string TOPIC_ENERGY_START_STOP_REQUEST = "EnergyStartStopRequest";

/**
 * Reports the active load sharing status of an alternating current Source / Storage device.
 * Version: 1.0.0
 * Section: SOURCE_DEVICE
 * QoS Profile: PublishLast
 * Data Trigger: At Start, On Change
 * Data Type: tms::ac::LoadSharingState
 */
const string TOPIC_AC_LOAD_SHARING_STATE = "AcLoadSharingState";

/**
 * Request to change the AC load sharing status of a Source / Storage device.
 * Version: 1.0.0
 * Section: SOURCE_DEVICE
 * QoS Profile: Command
 * Data Trigger: As Needed
 * Data Type: tms::ac::LoadSharingRequest
 * Reply Topic: Reply
 * Reply Topic: AcLoadSharingState
 */
const string TOPIC_AC_LOAD_SHARING_REQUEST = "AcLoadSharingRequest";

/**
 * Reports dynamic state changes of control device hardware.
 * Version: 1.0.0
 * Section: SOURCE_DEVICE
 * QoS Profile: Slow
 * Data Trigger: Periodic 10 s
 * Data Type: tms::ControlHardwareUpdate
 */
const string TOPIC_CONTROL_HARDWARE_UPDATE = "ControlHardwareUpdate";

/**
 * Reports dynamic state of power hardware components.
 * Version: 1.0.0
 * Section: SOURCE_DEVICE
 * QoS Profile: Slow
 * Data Trigger: Periodic 10 s
 * Data Type: tms::PowerHardwareUpdate
 */
const string TOPIC_POWER_HARDWARE_UPDATE = "PowerHardwareUpdate";

/**
 * The dynamic state of a Storage device.
 * Version: 1.0.0
 * Section: STORAGE_DEVICE
 * QoS Profile: Slow
 * Data Trigger: Periodic 10 s
 * Data Type: tms::StorageUpdate
 */
const string TOPIC_STORAGE_UPDATE = "StorageUpdate";
```

```
/**
 * The reply to a request.
 * Version: 1.0.0
 * Section: RESPONSE
 * QoS Profile: Reply
 * Data Trigger: Within 0.1 s
 * Data Type: tms::Reply
 */
const string TOPIC_REPLY = "Reply";

/**
 * The active power device control parameters for this power device.
 * Version: 1.0.0
 * Section: DEVICE_PARAMETERS
 * QoS Profile: PublishLast
 * Data Trigger: At Start, On Change
 * Data Type: tms::ControlParameterState
 */
const string TOPIC_CONTROL_PARAMETER_STATE = "ControlParameterState";

/**
 * Request to change the device control parameters as defined by the DeviceInfo.controlParameters.
 * Version: 1.0.0
 * Section: DEVICE_PARAMETERS
 * QoS Profile: Command
 * Data Trigger: As Needed
 * Data Type: tms::ControlParameterRequest
 * Reply Topic: Reply
 * Reply Topic: ControlParameterState
 */
const string TOPIC_CONTROL_PARAMETER_REQUEST = "ControlParameterRequest";

/**
 * The active metric parameter values for this power device.
 * Version: 1.0.0
 * Section: DEVICE_PARAMETERS
 * QoS Profile: PublishLast
 * Data Trigger: At Start, On Change
 * Data Type: tms::MetricParameterState
 */
const string TOPIC_METRIC_PARAMETER_STATE = "MetricParameterState";

/**
 * Provide the operator's response to an authorization request.
 * Version: 1.0.0
 * Section: BLACK_START
 * QoS Profile: Response
 * Data Trigger: As Needed
 * Data Type: tms::AuthorizationToEnergizeReply
 */
const string TOPIC_AUTHORIZATION_TO_ENERGIZE_REPLY = "AuthorizationToEnergizeReply";

/**
 * Request authorization to execute a request when a black start is detected.
 * Version: 1.0.0
 * Section: BLACK_START
 * QoS Profile: Command
 * Data Trigger: As Needed
 * Data Type: tms::AuthorizationToEnergizeRequest
 * Reply Topic: Reply
 * Reply Topic: AuthorizationToEnergizeReply
 */
const string TOPIC_AUTHORIZATION_TO_ENERGIZE_REQUEST = "AuthorizationToEnergizeRequest";

/**
 * Report the outcome of a request that required authorization.
 * Version: 1.0.0
 * Section: BLACK_START
 * QoS Profile: Response
 * Data Trigger: Within 0.1 s
 * Data Type: tms::AuthorizationToEnergizeResult
 */
const string TOPIC_AUTHORIZATION_TO_ENERGIZE_RESULT = "AuthorizationToEnergizeResult";

/**
 * The last published OperatorIntent received by a Microgrid Controller
 * Version: 1.0.0
 * Section: OPERATOR_INTENT
 * QoS Profile: PublishLast
 * Data Trigger: At Start, On Change
 * Data Type: tms::OperatorIntentState
 */
const string TOPIC_OPERATOR_INTENT_STATE = "OperatorIntentState";

/**
 * Operator defined microgrid control directives that influence how the microgrid behaves.
 * Version: 1.0.0
 * Section: OPERATOR_INTENT
 * QoS Profile: Command
 * Data Trigger: As Needed
 * Data Type: tms::OperatorIntentRequest
 * Reply Topic: OperatorIntentState
```

```
    */
   const string TOPIC_OPERATOR_INTENT_REQUEST = "OperatorIntentRequest";

   /**
    * Grounding Circuit states.
    * Version: 1.0.0
    * Section: GROUNDING_CIRCUIT
    * QoS Profile: PublishLast
    * Data Trigger: At Start, On Change
    * Data Type: tms::GroundingCircuitState
    */
   const string TOPIC_DEVICE_GROUNDING_STATE = "GroundingCircuitState";

   /**
    * Grounding Circuit request.
    * Version: 1.0.0
    * Section: GROUNDING_CIRCUIT
    * QoS Profile: Command
    * Data Trigger: As Needed
    * Data Type: tms::GroundingCircuitRequest
    * Reply Topic: Reply
    * Reply Topic: GroundingCircuitState
    */
   const string TOPIC_GROUNDING_CIRCUIT_REQUEST = "GroundingCircuitRequest";

   /**
    * Reports power measurements for direct current power device ports.
    * Version: 1.0.0
    * Section: DC_POWER_DEVICE
    * QoS Profile: Medium
    * Data Trigger: Periodic 1 s
    * Data Type: tms::dc::MeasurementUpdate
    */
   const string TOPIC_DC_MEASUREMENT_UPDATE = "DcMeasurementUpdate";

   /**
    * Request to change the DC load sharing status of a device.
    * Version: 1.0.0
    * Section: DC_POWER_DEVICE
    * QoS Profile: Command
    * Data Trigger: As Needed
    * Data Type: tms::dc::LoadSharingRequest
    * Reply Topic: Reply
    * Reply Topic: DcLoadSharingState
    */
   const string TOPIC_DC_LOAD_SHARING_REQUEST = "DcLoadSharingRequest";

   /**
    * Summarized power measurements for AC.  Fast updates useful for controlling power balance.
    * Version: 1.0.0
    * Section: SOURCE_DEVICE
    * QoS Profile: Continuous
    * Data Trigger: Periodic 0.1 s
    * Data Type: tms::ac::MeasurementUpdate
    */
   const string TOPIC_AC_SUMMARY_MEASUREMENT_UPDATE = "AcSummaryMeasurementUpdate";

   /**
    * Reports the active direct current load sharing status of a device.
    * Version: 1.0.0
    * Section: DC_POWER_DEVICE
    * QoS Profile: PublishLast
    * Data Trigger: At Start, On Change
    * Data Type: tms::dc::LoadSharingState
    */
   const string TOPIC_DC_LOAD_SHARING_STATE = "DcLoadSharingState";

   /**
    * Summarized power measurements for DC.  Fast updates useful for controlling power balance.
    * Version: 1.0.0
    * Section: SOURCE_DEVICE
    * QoS Profile: Continuous
    * Data Trigger: Periodic 0.1 s
    * Data Type: tms::dc::MeasurementUpdate
    */
   const string TOPIC_DC_SUMMARY_MEASUREMENT_UPDATE = "DcSummaryMeasurementUpdate";
};  //end module topic


const uint32 TopicName_MINLEN = 1;
const uint32 TopicName_MAXLEN = 64;
/**
 * Specifies a communications data topic.
 * Minimum length: 1
 * Maximum length: 64
 */
typedef string <TopicName_MAXLEN> TopicName;


const uint32 TopicList_MINLEN = 0;
const uint32 TopicList_MAXLEN = 64;
```

```
/**
 * Specifies a list of communications data topics.
 * Minimum length: 0
 * Maximum length: 64
 */
typedef sequence<TopicName, TopicList_MAXLEN> TopicList;


/**
 * Indicate whether a cable has been sensed in a power port.
 */
@extensibility(APPENDABLE)
enum CableSenseStatus {
    CS_UNKNOWN, // Cable presence is neither confirmed nor denied through active detection. Cable sense is not ↩
        available, either due to device design or failure.
    CS_DISCONNECTED, // Sensing indicates that a cable is not connected. CST_MEASUREMENT devices must use CS_UNKNOWN ↩
        instead of this value.
    CS_CONNECTED // Sensing indicates that a cable is connected.
}; // enum CableSenseStatus


/**
 * Indicate whether electricity can flow through a connection.
 */
@extensibility(APPENDABLE)
enum CircuitContinuity {
    CC_UNKNOWN, // No information is available, either due to design or malfunction.
    CC_OPEN, // Electrical or mechanical sensors indicate that continuity has been broken.
    CC_CLOSED, // Electrical or mechanical sensors indicate that continuity has been established.
    CC_INCONSISTENT // Electrical or mechanical sensors indicate that continuity has been established on some wires ↩
        and broken on other wires.
}; // enum CircuitContinuity


/**
 * Indicate how the monotonic clock is initialized.
 */
@extensibility(APPENDABLE)
enum ClockMonotonicPersistenceType {
    CMPT_UNKNOWN, // Fault detection indicates that initialization may be inconsistent.
    CMPT_RESET, // Clock initializes to 0.
    CMPT_ACCUMULATE, // Clock initializes to a value stored shortly before shutdown. This is effectively an aggregate↩
         runtime meter.
    CMPT_OTHER // Another clock initialization scheme is used.
}; // enum ClockMonotonicPersistenceType


/**
 * Indicate the desired circuit continuity of a switch.
 */
@extensibility(APPENDABLE)
enum DesiredCircuitContinuity {
    DCC_UNKNOWN, // Unknown value. This is a generic placeholder representing the case in which an appropriate value ↩
        could not be determined.
    DCC_OPEN, // Transition to CC_OPEN.
    DCC_CLOSED, // Transition to CC_CLOSED. Predicated on AuthorizationToEnergizeRequest.
    DCC_SYNC_CLOSED, // Transition to CC_CLOSED when synchronization is achieved. May close to a dead circuit, ↩
        predicated on AuthorizationToEnergizeRequest. May act as DCC_CLOSED_INTERNAL when the internal circuit is ↩
        dead.
    DCC_CLOSED_INTERNAL, // Transition to CC_CLOSED. One side of the switch must be a circuit enclosed inside a ↩
        device with protective guards.
    DCC_NO_CHANGE // No transition. May be used instead of DCC_CLOSED in non-active settings groups.
}; // enum DesiredCircuitContinuity


/**
 * Identify the role filled by a device.
 */
@extensibility(APPENDABLE)
enum DeviceRole {
    ROLE_UNKNOWN, // Unknown value. This is a generic placeholder representing the case in which an appropriate value↩
         could not be determined.  Never transmitted in this version of the standard.
    ROLE_MICROGRID_CONTROLLER, // Microgrid controller, a control service.
    ROLE_SOURCE, // Source power device.
    ROLE_LOAD, // Load power device.
    ROLE_STORAGE, // Storage power device.
    ROLE_DISTRIBUTION, // Distribution power device.
    ROLE_MICROGRID_DASHBOARD, // Microgrid dashboard, a control service.
    ROLE_CONVERSION, // Conversion power device.
    ROLE_MONITOR // Monitor microgrid, read-only access.
}; // enum DeviceRole


/**
 * Describe how a DTC affects the availability of the device
 */
@extensibility(APPENDABLE)
enum DtcSeverity {
    SEV_UNKNOWN, // Unknown value.  Never transmitted in this version of the standard.
    SEV_1_CLEAR, // DTC has cleared. Never transmitted in this version of the standard.
    SEV_2_INFORMATIVE, // Informational only. No performance impact is expected.
    SEV_3_PREVENTATIVE, // Preventative maintenance has been scheduled.
```

```
    SEV_4_DEGRADED, // Poor configuration or operation resulting in degraded performance, including reduced ↩
        efficiency and increased emissions.
    SEV_5_WARNING, // Increased wear and premature failure likely.
    SEV_6_MINOR, // The device may be failing to meet operating performance criteria. The DTC may represent cause or ↩
        effect.
    SEV_7_MAJOR, // This condition may cause protection logic to enter a controlled shutdown sequence. A corrective ↩
        action may be required before the device can be restarted.
    SEV_8_CRITICAL, // A loss of essential functionality has been detected. Routine maintenance or repair may be ↩
        required.
    SEV_9_FATAL, // A loss of essential functionality has been detected. Permanent damage may have occurred.
    SEV_10_HUMAN_SAFETY // Potential user safety risk.
}; // enum DtcSeverity


/**
 * Start/stop operating level of a SRC or STOR device.
 */
@extensibility(APPENDABLE)
enum EnergyStartStopLevel {
    ESSL_UNKNOWN, // Unknown value. This is a generic placeholder representing the case in which an appropriate value↩
         could not be determined.
    ESSL_ANY, // Wildcard value.  Allows the EnergyStartStopRequest.fromLevel to match any EnergyStartStopState.↩
        presentLevel.  Invalid for other uses.
    ESSL_OFF, // Device controller is operational.  Energy and power component temperatures are ambient.  Energy ↩
        component is disabled.  Power component is disabled.  Synchronization is not maintained.  Power port switch ↩
        is open.
    ESSL_WARM, // Device controller is operational.  Energy and power component temperatures are regulated.  Energy ↩
        component is disabled.  Power component is disabled.  Synchronization is not maintained.  Power port switch ↩
        is open.
    ESSL_IDLE, // Device controller is operational.  Energy and power component temperatures are regulated.  Energy ↩
        component is enabled but not ready to operate at rated capacity.  Power component is disabled.  ↩
        Synchronization is not maintained.  Power port switch is open.
    ESSL_READY, // Device controller is operational.  Energy and power component temperatures are regulated.  Energy ↩
        component is enabled and ready to operate at rated capcity.  Power component is enabled.  Synchronization is↩
         not maintained.  Power port switch is open.
    ESSL_READY_SYNCED, // Device controller is operational.  Energy and power component temperatures are regulated.  ↩
        Energy component is enabled and ready to operate at rated capcity.  Power component is enabled.  ↩
        Synchronization is maintained.  Power port switch is open.
    ESSL_OPERATIONAL // Device controller is operational.  Energy and power component temperatures are regulated.  ↩
        Energy component is enabled and ready to operate at rated capcity.  Power component is enabled.  ↩
        Synchronization is maintained.  Power port switch is closed.
}; // enum EnergyStartStopLevel


const uint32 EnergyStartStopLevelSequence_MINLEN = 0;
const uint32 EnergyStartStopLevelSequence_MAXLEN = 8;
/**
 * A sequence of EnergyStartStopLevel, allowing each operating level to be listed once.
 * Minimum length: 0
 * Maximum length: 8
 */
typedef sequence<@try_construct(USE_DEFAULT) EnergyStartStopLevel, EnergyStartStopLevelSequence_MAXLEN> ↩
     EnergyStartStopLevelSequence;


/**
 * Ground fault detection.
 */
@extensibility(APPENDABLE)
enum GroundFaultDetection {
    GFD_UNKNOWN, // Unknown value. This is a generic placeholder representing the case in which an appropriate value ↩
        could not be determined.
    GFD_NOT_APPLICABLE, // No ground fault detector is present on the device.
    GFD_NONE, // No ground fault detected.
    GFD_ANY, // Ground fault detected. Multiple lines detected or specific line unknown.
    GFD_A, // Ground fault detected on phase A.
    GFD_B, // Ground fault detected on phase B.
    GFD_C, // Ground fault detected on phase C.
    GFD_DCPOS, // Ground fault detected on DC+.
    GFD_DCNEG // Ground fault detected on DC-.
}; // enum GroundFaultDetection


/**
 * Describe the design type of a grounding circuit.
 */
@extensibility(APPENDABLE)
enum GroundingDesignType {
    GROUNDING_UNKNOWN, // Unknown value. This is a generic placeholder representing the case in which an appropriate ↩
        value could not be determined.
    GROUNDING_UNGROUNDED, // No grounding.
    GROUNDING_SOLID, // Solidly grounded.
    GROUNDING_HIGH_RESISTANCE, // High-resistance grounded system. Generally designed for 1-10 A ground fault current↩
        .
    GROUNDING_LOW_RESISTANCE, // Low-resistance grounded system. Generally designed to limit equipment damage.
    GROUNDING_REACTANCE // Reactance grounded system. Generally limits ground fault current to rated line current.
}; // enum GroundingDesignType


/**
 * Indicate the microgrid level operating mode
 */
```

```
@extensibility(APPENDABLE)
enum OperatingMode {
    OPM_UNKNOWN, // Unknown value. This is a generic placeholder representing the case in which an appropriate value ↩
         could not be determined.
    OPM_NORMAL, // Normal operating mode.
    OPM_EMERGENCY, // Emergency operating mode.
    OPM_SILENT_WATCH // Silent watch mode.
}; // enum OperatingMode


/**
 * Specify the type of Operator priority.
 */
@extensibility(APPENDABLE)
enum OperatorPriorityType {
    OPT_UNKNOWN, // Unknown value. This is a generic placeholder representing the case in which an appropriate value ↩
         could not be determined.
    OPT_NEVER_OPERATE, // Transition the device to the lowest supported EnergyStartStopLevel or open the power port.
    OPT_ALWAYS_OPERATE, // Transition the device to EnergyStartStopLevel ESSL_OPERATIONAL or close the power port.
    OPT_NUMERIC_RANK // Priority based on OperatorPriority.numericRank.
}; // enum OperatorPriorityType


/**
 * Specify the type of Operator Intent.
 */
@extensibility(APPENDABLE)
enum OperatorIntentType {
    OIT_UNKNOWN, // Unknown value. This is a generic placeholder representing the case in which an appropriate value ↩
         could not be determined.
    OIT_DEFAULT_INTENT, // Microgrid Controller defined default operator intent.
    OIT_OPERATOR_DEFINED // Microgrid Dashboard activated operator intent.
}; // enum OperatorIntentType


/**
 * Indicate how a power connection was detected.
 */
@extensibility(APPENDABLE)
enum PowerConnectionDetectionType {
    PCDT_UNKNOWN, // Unknown value. This is a generic placeholder representing the case in which an appropriate value↩
         could not be determined.
    PCDT_OPERATOR, // Operator entry. PowerConnectionId portA and portB are present.
    PCDT_CABLE_ID, // Identification embedded in a smart cable.  PowerConnectionId portA is present.  ↩
         PowerConnectionId portB is absent for power device connections.  PortB is present for MC power connections. ↩
         PowerPortStatus.cableId must be present
    PCDT_PROBE, // Active probe signal sent down the line.   PowerConnectionId portA is present, portB is absent.
    PCDT_CORRELATION, // Correlation of power measurements was used.  PowerConnectionId portA and portB are present.
    PCDT_COMMUNICATIONS, // Power line carrier or similar network communications was used.  PowerConnectionId portA ↩
         and portB are present.
    PCDT_HISTORY // Device reset or other event has lost continuous monitoring of the connection. This connection was↩
          detected before the event and has neither been confirmed nor denied since.
}; // enum PowerConnectionDetectionType


/**
 * Indicator of success or failure. Intended to support automatic handling.
 */
@extensibility(APPENDABLE)
enum ReplyCode {
    REPLY_UNKNOWN, // Unknown value. This is a generic placeholder representing the case in which an appropriate ↩
         value could not be determined.
    REPLY_OK, // The request has succeeded.
    REPLY_BAD_REQUEST, // Some value in the request was invalid.
    REPLY_METHOD_NOT_ALLOWED, // A request method is not supported for the requested resource.
    REPLY_CONFLICT, // Indicates that the request could not be processed because of conflict in the current state of ↩
         the resource.
    REPLY_GONE, // Indicates that the resource requested is no longer available and will not be available again. This↩
          should be used when a resource has been intentionally removed and the resource should be purged.
    REPLY_PRECONDITION_FAILED, // The request does not meet one of the preconditions.
    REPLY_INTERNAL_SERVER_ERROR, // A generic error message, given when an unexpected condition was encountered and ↩
         no more specific message is suitable.
    REPLY_NOT_IMPLEMENTED, // The request either does not recognize the request method, or it lacks the ability to ↩
         fulfil the request.
    REPLY_SERVICE_UNAVAILABLE, // The device cannot handle the request (because it is overloaded or down for ↩
         maintenance). Generally, this is a temporary state
    REPLY_PENDING_AUTHORIZATION, // Request is valid, and authorization is required before processing.
    REPLY_NOT_MASTER, // The requestingDeviceId is not the selected master MC.
    /**
     * Change History:
     *  1) Version: 2.0.1, Change: add, Interoperability: No-xtypes
     *     Allow device to provide more specified reason a request
     *     is denied.
     */
    REPLY_PERMISSION_DENIED // Remote control is disabled for this request.
}; // enum ReplyCode


/**
 * Indicate the wiring scheme for an AC or DC circuit.
 */
@extensibility(APPENDABLE)
```

```
enum CircuitWiring {
    WIRING_UNKNOWN, // Unknown value. This is a generic placeholder representing the case in which an appropriate ↩
        value could not be determined.
    WIRING_AC_SINGLE, // Single-phase AC circuit.  Hot wire.
    WIRING_AC_SPLIT, // Split-phase AC circuit.  Wires A, B, and N.
    WIRING_AC_3WYE, // 3-phase AC circuit, wye configuration.  Wires A, B, C, and N.
    WIRING_AC_3DELTA, // 3-phase AC circuit, delta configuration.  Wires A, B, and C.
    WIRING_DC, // 2-wire DC circuit.  Positive and negative.
    WIRING_DC_3WIRE // 3-wire DC circuit.  Positive, negative, and neutral.
}; // enum CircuitWiring


/**
 * Indicate the polarity of a power connector.
 */
@extensibility(APPENDABLE)
enum PowerConnectorPolarity {
    POLARITY_UNKNOWN, // Unknown value. This is a generic placeholder representing the case in which an appropriate ↩
        value could not be determined.
    POLARITY_PIN, // This connector contains pins to insert in sockets.
    POLARITY_SOCKET, // This connector contains sockets to receive pins. Includes terminal blocks.
    POLARITY_UNIVERSAL // This connector is symmetric.
}; // enum PowerConnectorPolarity


/**
 * Indicate the standard defining the physical shape and form of a power connector.
 */
@extensibility(APPENDABLE)
enum PowerConnectorType {
    CONNECTOR_UNKNOWN, // Unknown value. This is a generic placeholder representing the case in which an appropriate ↩
        value could not be determined.
    CONNECTOR_TERMINAL_BLOCK, // Terminal block.
    CONNECTOR_MILSTD, // Generic MIL spec connector. MIL-STD-1651, MIL-DTL-22992, and MIL-DTL-53126, as used by PDISE↩
        .
    CONNECTOR_NEMA5, // For convenience power ports, etc.
    CONNECTOR_CAMLOCK, // As used by MEPDIS-R.
    CONNECTOR_POWERLOCK, // ITT VEAM PowerLock
    CONNECTOR_IEC60309, // As used by MEPDIS-R.
    CONNECTOR_J1772, // Electric vehicle charging cable.
    CONNECTOR_POWERLOK, // Amphenol PowerLok.
    CONNECTOR_MILSTD1651, // MIL-STD-1651.
    CONNECTOR_MILDTL22992, // MIL-DTL-22992.
    CONNECTOR_MILDTL53126, // MIL-DTL-53126.
    CONNECTOR_OTHER, // Other connector type, not listed elsewhere.
    CONNECTOR_METER, // Used to represent a clamp on power meter.
    CONNECTOR_BUS, // Not a physical connector.  Used to represent the device internal bus.   Typically used by a ↩
        Distribution or Conversion device but not limited to these device roles.
    /**
     * Change History:
     *  1) Version: 2.0.3, Change: add, Interoperability: Yes
     *     New connector requested by user.
     */
    CONNECTOR_VOLTARIUS // Amphenol Voltarius high-voltage interconnect.
}; // enum PowerConnectorType


/**
 * Anticipated direction of power flow.
 */
@extensibility(APPENDABLE)
enum PowerPortDirectionality {
    PPD_UNKNOWN, // not initialized.
    PPD_NONE, // directionality not applicable (e.g., for a logical port, expect zero power).
    PPD_IN, // Port intended to receive power.
    PPD_OUT, // Port intended to send power .
    PPD_IN_OUT // Port intended to send and receive power.
}; // enum PowerPortDirectionality


/**
 * Indicate how a cable can be detected.
 */
@extensibility(APPENDABLE)
enum PowerConnectorFeature {
    PCF_UNKNOWN, // Unknown value. This is a generic placeholder representing the case in which an appropriate value ↩
        could not be determined.
    PCF_CABLE_SENSE, // A continuity circuit or similar feature detects when a cable is connected.
    PCF_CABLE_ID_READER, // A cable ID reader is present.
    PCF_CABLE_MEASUREMENT, // Voltage and current monitoring can establish the presence of a cable.
    PCF_COMMUNICATION, // Contains network interface.
    PCF_TOPOLOGY_DISCOVERY // Ability to discover connectivity to remote device power ports.
}; // enum PowerConnectorFeature


/**
 * Describes a feature on a switch.  The MANUAL, AUTO, and REQUEST features cover what type of actors may change the ↩
     switch state.
 */
@extensibility(APPENDABLE)
enum PowerSwitchFeature {
    PSF_UNKNOWN, // Unknown value. This is a generic placeholder representing the case in which an appropriate value ↩
```

```
        could not be determined.
    PSF_MANUAL_OPEN, // The operator may open the switch using an on-device interface.
    PSF_MANUAL_CLOSE, // The operator may close the switch using an on-device interface.
    PSF_AUTO_OPEN, // The power device may automatically open the switch.  For example, load shed logic.
    PSF_AUTO_CLOSE, // The power device may automatically close the switch.  For example, automatic backup power logic.
    PSF_REQUEST_OPEN, // A request such as PowerSwitchRequest or EnergyStartStopRequest may open the switch.  For example, the switch may be a contactor, relay, motorized switch, or a circuit breaker with a shunt trip.
    PSF_REQUEST_CLOSE, // A request such as PowerSwitchRequest or EnergyStartStopRequest may close the switch.  For example, the switch may be a contactor, relay, or motorized switch.
    PSF_LOCK, // The operator may lock the switch to prevent AUTO or REQUEST operation.  This lock may be per-device or per-port.  For example, IEEE C37.2 device function 1, 5, 69, or 86.
    PSF_RECLOSER, // Automatic recloser.  For example, IEEE C37.2 device function 79.  Depends on PSF_AUTO_CLOSE.
    PSF_BREAKER, // Automatic over current / thermal protection.  For example, IEEE C37.2 device function 50 or 51.  Depends on PSF_AUTO_OPEN.
    PSF_GFI, // Automatic ground fault protection.  Depends on PSF_AUTO_OPEN.
    PSF_ARC_FLASH, // Automatic arc flash protection.  For example, IEEE C37.2 device function AFD.  Depends on PSF_AUTO_OPEN.
    PSF_SYNCHRONIZER, // Synchronize before close.  For example, IEEE C37.2 device function 25.  Depends on PSF_AUTO_CLOSE or PSF_REQUEST_CLOSE.
    PSF_SURGE, // Automatic over voltage protection.  Depends on PSF_AUTO_OPEN.
    PSF_SWITCH_CONDITIONS // Supports PowerSwitchRequest.switchConditions.  Depends on PSF_REQUEST_CLOSE and PowerPortInfo.hasExternalMeter.
}; // enum PowerSwitchFeature


/**
 * Indicate why a power switch entered a continuity state.
 */
@extensibility(APPENDABLE)
enum PowerSwitchReason {
    PSR_UNKNOWN, // No information is available, either due to design or malfunction.
    PSR_STARTUP, // Initial state when the device started.
    PSR_MANUAL, // Changed by a local operator, either mechanical or digital.  PSR_OPERATOR_LOGIN if the operator is authenticated and has an identity.
    PSR_AUTO, // Automatic operation initiated by the device and not covered by a more specific AUTO reason.
    PSR_REQUEST, // Responded to a MC request.
    PSR_OPERATOR_LOGIN, // Changed by a local operator who is logged in to an internal display and has an identity. PSR_MANUAL when not logged in or no operator identity.
    PSR_AUTO_PROTECTION, // Activated by an internal protection function.  Should report SPN_TMS_TRIP_* for additional details.
    PSR_AUTO_LOAD_SHED, // Activated by internal load shed logic.
    PSR_AUTO_VOLTAGE_LOSS, // Responded to an interruption in line voltage.
    PSR_AUTO_COMMS_LOSS, // Responded to an interruption in microgrid communications.
    PSR_MALFUNCTION // Apparent internal malfunction.
}; // enum PowerSwitchReason


/**
 * Describes a feature on a specific energy source device.
 */
@extensibility(APPENDABLE)
enum SourceFeature {
    SRCF_UNKNOWN, // Unknown value. This is a generic placeholder representing the case in which an appropriate value could not be determined.
    SRCF_GENSET, // Genset source.
    SRCF_FUEL_CELL, // Fuel cell source.
    SRCF_SOLAR, // Solar source.
    SRCF_WIND, // Wind source.
    SRCF_VEHICLE // Source shared with a vehicle power train.
}; // enum SourceFeature


/**
 * Describes a feature on a specific energy load.
 */
@extensibility(APPENDABLE)
enum LoadFeature {
    LOADF_UNKNOWN, // Unknown value. This is a generic placeholder representing the case in which an appropriate value could not be determined.
    LOADF_DEMAND_RESPONSE, // Device can shed load to provide grid support.
    LOADF_CHANGE_NOTIFICATION, // Device can notify the microgrid before causing a large load transient.
    LOADF_SOFT_START // Device can smoothly ramp up and down to minimize load transients.
}; // enum LoadFeature


/**
 * Describes a feature on a specific energy distribution device.
 */
@extensibility(APPENDABLE)
enum DistributionFeature {
    DISTF_UNKNOWN, // Unknown value. This is a generic placeholder representing the case in which an appropriate value could not be determined.
    DISTF_CLAMP_METER, // clamp-on power meter, two logical power ports.
    DISTF_TAP_METER, // pass-through power meter, two physical power ports.
    DISTF_PCC, // has additional capabilities for PCC interconnect with another grid.
    DISTF_FEEDER, // intended for supplying power to other DIST devices.
    DISTF_DISTRIBUTION // intended for supplying power to LOAD devices.
}; // enum DistributionFeature


/**
```

```
 * Describes a feature on a specific energy converter.
 */
@extensibility(APPENDABLE)
enum ConversionFeature {
    CONVF_UNKNOWN, // Unknown value. This is a generic placeholder representing the case in which an appropriate ↩
        value could not be determined.
    CONVF_ACTIVE, // Uses power electronics or other active components (programmable).
    CONVF_PASSIVE // Uses transformers or other passive components (fixed function).
}; // enum ConversionFeature


/**
 * Describes a feature on a specific microgrid dashboard.
 */
@extensibility(APPENDABLE)
enum MicrogridDashboardFeature {
    MDF_UNKNOWN, // Unknown value. This is a generic placeholder representing the case in which an appropriate value ↩
        could not be determined.
    MDF_DISPLAY, // Can show microgrid status.
    MDF_CONTROL // Can control microgrid status.
}; // enum MicrogridDashboardFeature


/**
 * Describes a feature on a specific microgrid controller.
 */
@extensibility(APPENDABLE)
enum MicrogridControllerFeature {
    MCF_UNKNOWN, // Unknown value. This is a generic placeholder representing the case in which an appropriate value ↩
        could not be determined.
    MCF_FIXED, // special-purpose microgrid controller designed for a single platform or limited microgrid ↩
        configuration.
    MCF_GENERAL // general-purpose microgrid controller.
}; // enum MicrogridControllerFeature


/**
 * Describes a feature of a specific energy storage device.
 */
@extensibility(APPENDABLE)
enum StorageFeature {
    STORF_UNKNOWN, // Unknown value. This is a generic placeholder representing the case in which an appropriate ↩
        value could not be determined.
    STORF_GRID, // Grid attached storage, does not require pass-through from source to load.
    STORF_SUBCYCLE_UPS, // Can provide fast voltage support to recover from a grid fault condition.
    STORF_CHARGING, // Integrated into a charging station, as for electric vehicles.
    STORF_VEHICLE // Integrated into an electric vehicle power train.
}; // enum StorageFeature


/**
 * Identifies the time when configuration settings are to be used.
 */
@extensibility(APPENDABLE)
enum ConfigId {
    CONFIG_UNKNOWN, // Unknown value. This is a generic placeholder representing the case in which an appropriate ↩
        value could not be determined.
    CONFIG_DEFAULTS, // Read-only manufacturer defaults.
    CONFIG_ACTIVE, // Settings for use at the present time.
    CONFIG_ON_REBOOT, // Settings for use after a reboot / restart.
    CONFIG_ON_COMMS_LOSS // Settings for use after a communications loss timeout.
}; // enum ConfigId


/**
 * Describes the type of fault. See SAE J1939.
 */
@extensibility(APPENDABLE)
enum FailureModeIndicator {
    FMI_UNKNOWN, // No information is available, either due to design or malfunction.
    FMI_HIGH_MOST_SEVERE_HIGH, // 0, Data valid but above normal operational range, most severe level.
    FMI_HIGH_MOST_SEVERE_LOW, // 1, Data valid but below normal operational range, most severe level.
    FMI_DATA_ERRATIC, // 2, Data erratic, intermittent or incorrect (rationality).
    FMI_VOLTAGE_HIGH, // 3, Voltage above normal or shorted to high source.
    FMI_VOLTAGE_LOW, // 4, Voltage below normal or shorted to low source.
    FMI_CURRENT_LOW, // 5, Current below normal or open circuit.
    FMI_CURRENT_HIGH, // 6, Current above normal or ground circuit.
    FMI_NOT_RESPONDING, // 7, Mechanical system not responding or out of adjustment.
    FMI_ABNORMAL_FREQUENCY, // 8, Abnormal frequency or pulse width or period.
    FMI_ABNORMAL_UPDATE_RATE, // 9, Abnormal update rate.
    FMI_ABNORMAL_RATE_OF_CHARGE, // 10, Abnormal rate of change.
    FMI_FAILURE_CODE_NOT_IDENTIFIABLE, // 11, Failure code not identifiable.
    FMI_BAD_INTELLIGENT_DEVICE, // 12, Bad intelligent device or component.
    FMI_OUT_OF_CALIBRATION, // 13, Out of calibration.
    FMI_SPECIAL_INSTRUCTIONS, // 14, Special instructions.
    FMI_HIGH_LEAST_SEVERE, // 15, Data valid but above normal operational range, least severe level.
    FMI_HIGH_MODERATELY_SEVERE, // 16, Data valid but above normal operational range, moderately severe level.
    FMI_LOW_LEAST_SEVERE, // 17, Data valid but below normal operational range, least severe level.
    FMI_LOW_MODERATELY_SEVERE, // 18, Data valid but below normal operational range, moderately severe level.
    FMI_RECEIVED_DATA_ERROR, // 19, Received network data in error (multiplexed data).
    FMI_DATA_DRIFTED_HIGH, // 20, Data drifted high (rationality high).
    FMI_DATA_DRIFTED_LOW, // 21, Data drifted low (rationality low).
```

```
    FMI_CONDITION_EXISTS // 31, Condition exists.
}; // enum FailureModeIndicator


/**
 * Identifies a port within a device. Port numbers below 0 are invalid.
 */
typedef int32 PowerPortNumber;

/** Maximum number of power ports that a device may contain. */
const uint32 MAX_PORTS = 64;

/** A constant value for an unspecified port number (-1). */
const int32 INVALID_PORT_NUMBER = -1;


const uint32 Identity_MINLEN = 1;
const uint32 Identity_MAXLEN = 32;
/**
 * Universally unique identifier for a device, person, platform, or other entity.
 * Minimum length: 1
 * Maximum length: 32
 */
typedef string <Identity_MAXLEN> Identity;


const uint32 IdentitySequence_MINLEN = 0;
const uint32 IdentitySequence_MAXLEN = MAX_PORTS;
/**
 * A sequence of Identity.
 * Minimum length: 0
 * Maximum length: MAX_PORTS
 */
typedef sequence<Identity, IdentitySequence_MAXLEN> IdentitySequence;


const uint32 GlobalTradeItemNumber_LEN = 14;
/**
 * Encodes a Global Trade Item Number (GTIN).
 * Fixed length: 14
 */
typedef char GlobalTradeItemNumber[GlobalTradeItemNumber_LEN];


/**
 * Identifies a grounding circuit within a device.
 */
typedef int32 GroundingCircuitNumber;

/** Maximum number of grounding circuits that a device may contain. */
const int32 MAX_GROUNDING_CIRCUITS = 8;

/** A constant value for an invalid grounding circuit (-1). */
const int32 INVALID_GROUNDING_CIRCUIT_NUMBER = -1;


const uint32 NatoStockNumber_LEN = 13;
/**
 * Encodes a NATO Stock Number (NSN), aka National Stock Number.
 * Fixed length: 13
 */
typedef char NatoStockNumber[NatoStockNumber_LEN];


const uint32 String16_MINLEN = 0;
const uint32 String16_MAXLEN = 16;
/**
 * A variable length string.
 * Minimum length: 0
 * Maximum length: 16
 */
typedef string <String16_MAXLEN> String16;


const uint32 String32_MINLEN = 0;
const uint32 String32_MAXLEN = 32;
/**
 * A variable length string.
 * Minimum length: 0
 * Maximum length: 32
 */
typedef string <String32_MAXLEN> String32;


const uint32 String1_32_MINLEN = 1;
const uint32 String1_32_MAXLEN = 32;
/**
 * A variable length string with a fixed minimum.
 * Minimum length: 1
 * Maximum length: 32
 */
typedef string <String1_32_MAXLEN> String1_32;
```

```
  /**
   * Identifies the sensor or subsystem at issue or a particular event or condition.
   */
  typedef uint32 SuspectParameterNumber;

  /** First SPN in the range reserved for SAE J1939 standard parameters    * Change History:
   *  1) Version: 2.0.3, Change: add, Interoperability: Yes
   *     Added range start and SPN_J1939_* constants
*/
  const uint32 SPN_RANGE_J1939 = 0;

  /** Fuel Level 1 (tms::FuelState.fuelLevel) */
  const uint32 SPN_J1939_FUEL_LEVEL_1 = 96;

  /** Engine Oil Pressure (tms::EngineState.oilPressure) */
  const uint32 SPN_J1939_ENGINE_OIL_PRESSURE_1 = 100;

  /** Engine Coolant Temperature (tms::EngineState.coolantTemperature) */
  const uint32 SPN_J1939_ENGINE_COOLANT_TEMPERATURE = 110;

  /** Key Switch Battery Potential (TMS: internal starter battery for control and power hardware, not the main STOR ↩
       pack) */
  const uint32 SPN_J1939_KEY_SWITCH_BATTERY_POTENTIAL = 158;

  /** Engine Shutdown Override Switch (TMS: Battle Short enabled) */
  const uint32 SPN_J1939_ENGINE_SHUTDOWN_OVERRIDE_SWITCH = 1237;

  /** Emergency Shutdown Switch (TMS: Emergency Stop enabled) */
  const uint32 SPN_J1939_EMERGENCY_SHUTDOWN_SWITCH = 3585;

  /** First SPN in the range reserved for vendor-specific parameters    * Change History:
   *  1) Version: 2.0.3, Change: add, Interoperability: Yes
   *     Added range start
*/
  const uint32 SPN_RANGE_PROPRIETARY = 516096;

  /** First SPN in the range reserved for TMS standard parameters    * Change History:
   *  1) Version: 2.0.3, Change: add, Interoperability: Yes
   *     Added range start and SPN_TMS_* constants
*/
  const uint32 SPN_RANGE_TMS = 524288;

  /** Local automatic control has been disabled, as for maintenance or troubleshooting activities (FMI_CONDITION_EXISTS↩
       ) */
  const uint32 SPN_TMS_CONTROL_LOCKOUT_LOCAL = 524300;

  /** Remote automatic control has been disabled, including all TMS requests (FMI_CONDITION_EXISTS) */
  const uint32 SPN_TMS_CONTROL_LOCKOUT_REMOTE = 524301;

  /** Indicates a protections trip (PSR_AUTO_PROTECTION) due to over-voltage (FMI_HIGH), under-voltage (FMI_LOW), or ↩
       rate of change of voltage (FMI_ABNORMAL_ROC).  Corresponds to ANSI C37.2 numbers 59* and 27*. */
  const uint32 SPN_TMS_TRIP_VOLTAGE = 524400;

  /** Indicates a protections trip (PSR_AUTO_PROTECTION) due to over-frequency (FMI_HIGH), under-frequency (FMI_LOW), ↩
       or rate of change of frequency (FMI_ABNORMAL_ROC).  Corresponds to ANSI C37.2 numbers 81O, 81U, and 81R. */
  const uint32 SPN_TMS_TRIP_FREQUENCY = 524401;

  /** Indicates a protections trip (PSR_AUTO_PROTECTION) due to over-amperage (FMI_HIGH), under-amperage (FMI_LOW), or ↩
       rate of change of amperage (FMI_ABNORMAL_ROC).  Corresponds to ANSI C37.2 numbers 51* and 37. */
  const uint32 SPN_TMS_TRIP_AMPERAGE = 524402;

  /** Indicates a protections trip (PSR_AUTO_PROTECTION) due to over-power (FMI_HIGH), under-power / reverse power (↩
       FMI_LOW), or rate of change of power (FMI_ABNORMAL_ROC).  Corresponds to ANSI C37.2 numbers 32* and 37P. */
  const uint32 SPN_TMS_TRIP_POWER = 524403;

  /** Indicates a protections trip (PSR_AUTO_PROTECTION) due to over-energy (FMI_HIGH), under-energy (FMI_LOW), or rate↩
        of change of energy (FMI_ABNORMAL_ROC). */
  const uint32 SPN_TMS_TRIP_ENERGY = 524404;

  /** Indicates a protections trip (PSR_AUTO_PROTECTION) due to a grounding fault.  Corresponds to ANSI C37.2 numbers ↩
       64* */
  const uint32 SPN_TMS_TRIP_GROUND = 524405;

  /** Indicates a protections trip (PSR_AUTO_PROTECTION) due to over-temperature (FMI_HIGH) or under-temperature (↩
       FMI_LOW).  Does not apply to thermal-based current or power protections. */
  const uint32 SPN_TMS_TRIP_TEMPERATURE = 524406;


  /**
   * Tap changer index in a passive conversion device.  Defaults to 0 when no changer is present.
   */
  typedef int32 TapNumber;

  /** Maximum number of positions in a tap changer.  Many tap changers have 33 positions. */
  const int32 MAX_TAPS = 64;


  const uint32 PowerPortNumberSequence_MINLEN = 0;
  const uint32 PowerPortNumberSequence_MAXLEN = MAX_PORTS;
  /**
```

```
 * A sequence of PowerPortNumber.
 * Minimum length: 0
 * Maximum length: MAX_PORTS
 */
typedef sequence<PowerPortNumber, PowerPortNumberSequence_MAXLEN> PowerPortNumberSequence;


const uint32 ConversionFeatureSequence_MINLEN = 0;
const uint32 ConversionFeatureSequence_MAXLEN = 3;
/**
 * A sequence of ConversionFeature.
 * Minimum length: 0
 * Maximum length: 3
 */
typedef sequence<@try_construct(USE_DEFAULT) ConversionFeature, ConversionFeatureSequence_MAXLEN> ↩
    ConversionFeatureSequence;


const uint32 OctetSequence_MINLEN = 1;
const uint32 OctetSequence_MAXLEN = 32768;
/**
 * A sequence of Octet (byte).
 * Minimum length: 1
 * Maximum length: 32768
 */
typedef sequence<octet, OctetSequence_MAXLEN> OctetSequence;


const uint32 DistributionFeatureSequence_MINLEN = 0;
const uint32 DistributionFeatureSequence_MAXLEN = 6;
/**
 * A sequence of DistributionFeature.
 * Minimum length: 0
 * Maximum length: 6
 */
typedef sequence<@try_construct(USE_DEFAULT) DistributionFeature, DistributionFeatureSequence_MAXLEN> ↩
    DistributionFeatureSequence;


const uint32 PowerConnectorFeatureSequence_MINLEN = 0;
const uint32 PowerConnectorFeatureSequence_MAXLEN = 6;
/**
 * A sequence of PowerConnectorFeature.
 * Minimum length: 0
 * Maximum length: 6
 */
typedef sequence<@try_construct(USE_DEFAULT) PowerConnectorFeature, PowerConnectorFeatureSequence_MAXLEN> ↩
    PowerConnectorFeatureSequence;


const uint32 PowerSwitchFeatureSequence_MINLEN = 0;
const uint32 PowerSwitchFeatureSequence_MAXLEN = 15;
/**
 * A sequence of PowerSwitchFeature.
 * Minimum length: 0
 * Maximum length: 15
 */
typedef sequence<@try_construct(USE_DEFAULT) PowerSwitchFeature, PowerSwitchFeatureSequence_MAXLEN> ↩
    PowerSwitchFeatureSequence;


const uint32 LoadFeatureSequence_MINLEN = 0;
const uint32 LoadFeatureSequence_MAXLEN = 4;
/**
 * A sequence of LoadFeature.
 * Minimum length: 0
 * Maximum length: 4
 */
typedef sequence<@try_construct(USE_DEFAULT) LoadFeature, LoadFeatureSequence_MAXLEN> LoadFeatureSequence;


const uint32 MicrogridControllerFeatureSequence_MINLEN = 1;
const uint32 MicrogridControllerFeatureSequence_MAXLEN = 3;
/**
 * A sequence of MicrogridControllerFeature.
 * Minimum length: 1
 * Maximum length: 3
 */
typedef sequence<@try_construct(USE_DEFAULT) MicrogridControllerFeature, MicrogridControllerFeatureSequence_MAXLEN> ↩
    MicrogridControllerFeatureSequence;


const uint32 MicrogridDashboardFeatureSequence_MINLEN = 1;
const uint32 MicrogridDashboardFeatureSequence_MAXLEN = 3;
/**
 * A sequence of MicrogridDashboardFeature.
 * Minimum length: 1
 * Maximum length: 3
 */
typedef sequence<@try_construct(USE_DEFAULT) MicrogridDashboardFeature, MicrogridDashboardFeatureSequence_MAXLEN> ↩
    MicrogridDashboardFeatureSequence;
```

```
const uint32 EnumLabelSequence_MINLEN = 0;
const uint32 EnumLabelSequence_MAXLEN = 128;
/**
 * A sequence of String1_32 values that contain unique labels for an enumerated parameter.
 * Minimum length: 0
 * Maximum length: 128
 */
typedef sequence<String1_32, EnumLabelSequence_MAXLEN> EnumLabelSequence;


const uint32 SourceFeatureSequence_MINLEN = 0;
const uint32 SourceFeatureSequence_MAXLEN = 6;
/**
 * A sequence of SourceFeature.
 * Minimum length: 0
 * Maximum length: 6
 */
typedef sequence<@try_construct(USE_DEFAULT) SourceFeature, SourceFeatureSequence_MAXLEN> SourceFeatureSequence;


const uint32 StorageFeatureSequence_MINLEN = 0;
const uint32 StorageFeatureSequence_MAXLEN = 5;
/**
 * A sequence of StorageFeature.
 * Minimum length: 0
 * Maximum length: 5
 */
typedef sequence<@try_construct(USE_DEFAULT) StorageFeature, StorageFeatureSequence_MAXLEN> StorageFeatureSequence;


const uint32 ThermalLoadSequence_MINLEN = 0;
const uint32 ThermalLoadSequence_MAXLEN = 5;
/**
 * A sequence of float32. Estimate of temperature control load as a fraction of capacity.  0=no load, 1=full rated ↩
      load (hot or cold).  Values above 1 indicate overload conditions.
 * Minimum length: 0
 * Maximum length: 5
 * Minimum Value: 0
 */
typedef sequence<float, ThermalLoadSequence_MAXLEN> ThermalLoadSequence;


const uint32 ThermalZoneSequence_MINLEN = 0;
const uint32 ThermalZoneSequence_MAXLEN = 5;
/**
 * A sequence of String1_32.
 * Minimum length: 0
 * Maximum length: 5
 */
typedef sequence<String1_32, ThermalZoneSequence_MAXLEN> ThermalZoneSequence;



/**
 * Represent a point in a two-dimensional space.
 */
@nested
@extensibility(APPENDABLE)
struct Point2D {

    /**
     * Horizontal coordinate.
     */
    float x;

    /**
     * Vertical coordinate.
     */
    float y;
}; // end struct Point2D


const uint32 Point2DSequence_MINLEN = 0;
const uint32 Point2DSequence_MAXLEN = 21;
/**
 * A sequence of Point2D.
 * Minimum length: 0
 * Maximum length: 21
 */
typedef sequence<Point2D, Point2DSequence_MAXLEN> Point2DSequence;



/**
 * Represent a curve in a two-dimensional space.
 */
@nested
@extensibility(APPENDABLE)
struct Curve2D {

    /**
```

```
     * Vertices on the curve.
     */
    @try_construct(TRIM)
    Point2DSequence points;
}; // end struct Curve2D


/**
 * Specify output variable y as a function of input variable x.
 */
@nested
@extensibility(APPENDABLE)
struct ControlCurve {

    /**
     * Minimum acceptable value of y. Devices should operate above this curve.
     */
    Curve2D minimum;

    /**
     * Nominal value of y. Used as the set-point for PID controls.
     */
    Curve2D nominal;

    /**
     * Maximum acceptable value of y. Devices should operate below this curve.
     */
    Curve2D maximum;
}; // end struct ControlCurve


/**
 * Key value for a request that targets all devices in the grid.
 */
@nested
@extensibility(APPENDABLE)
struct GridRequest {

    /**
     * Identity of the device that sent this request.
     */
    Identity requestingDeviceId;
}; // end struct GridRequest


/**
 * Local oscillator timestamp, not affected by clock jumps.
 */
@nested
@extensibility(APPENDABLE)
struct ClockMonotonic {

    /**
     * integer portion of the timestamp.
     */
    uint32 seconds;

    /**
     * fractional portion of the timestamp.
     */
    uint32 nanoseconds;
}; // end struct ClockMonotonic


/**
 * Design information for thermal management.
 */
@nested
@extensibility(APPENDABLE)
struct ThermalInfo {

    /**
     * Name of each thermal zone.
     */
    @try_construct(TRIM)
    ThermalZoneSequence thermalZone;
}; // end struct ThermalInfo


/**
 * Provides operating information of the thermal management in this device.
 */
@nested
@extensibility(APPENDABLE)
struct ThermalState {

    /**
```

```
     * Load for each thermalZone, in same order published in tms.ThermalInfo.
     */
    @try_construct(TRIM)
    ThermalLoadSequence thermalLoad;
}; // end struct ThermalState


/**
 * Model a synchronous generator, motor, or similar device for simulation and control.  Direct-axis and quadrature-axis (DQ) synchronous machine model based on IEEE 115.
 */
@nested
@extensibility(APPENDABLE)
struct SynchronousMachineCoefficients {

    /**
     * Ra, stator resistance per phase
     * units: ohm (ohm)
     */
    @unit("ohm")
    float statorResistancePerPhase;

    /**
     * Xl, stator leakage reactance
     * units: ohm (ohm)
     */
    @unit("ohm")
    float statorLeakageReactance;

    /**
     * X0, zero-sequence reactance
     * units: ohm (ohm)
     */
    @unit("ohm")
    float zeroSequenceReactance;

    /**
     * X2, negative-sequence reactance
     * units: ohm (ohm)
     */
    @unit("ohm")
    float negativeSequenceReactance;

    /**
     * R0, zero-sequence resistance
     * units: ohm (ohm)
     */
    @unit("ohm")
    float zeroSequenceResistance;

    /**
     * R2 negative-sequence resistance
     * units: ohm (ohm)
     */
    @unit("ohm")
    float negativeSequenceResistance;

    /**
     * Xd, direct-axis synchronous reactance
     * units: ohm (ohm)
     */
    @unit("ohm")
    float directAxisSynchronousReactance;

    /**
     * Xd', direct-axis transient reactance
     * units: ohm (ohm)
     */
    @unit("ohm")
    float directAxisTransientReactance;

    /**
     * Xd'', direct-axis subtransient reactance
     * units: ohm (ohm)
     */
    @unit("ohm")
    float directAxisSubtransientReactance;

    /**
     * Xq, quadrature-axis synchronous reactance
     * units: ohm (ohm)
     */
    @unit("ohm")
    float quadratureAxisSynchronousReactance;

    /**
     * Xq', quadrature-axis transient reactance
     * units: ohm (ohm)
     */
    @unit("ohm")
    float quadratureAxisTransientReactance;
```

```
    /**
     * Xq'', quadrature-axis subtransient reactance
     * units: ohm (ohm)
     */
    @unit("ohm")
    float quadratureAxisSubtransientReactance;

    /**
     * Td', direct-axis transient short-circuit time constant
     * units: second (s)
     */
    @unit("s")
    float directAxisTransientShortCircuitTimeConstant;

    /**
     * Td'', direct-axis subtransient short-circuit time constant
     * units: second (s)
     */
    @unit("s")
    float directAxisSubtransientShortCircuitTimeConstant;

    /**
     * Td0', direct-axis transient open-circuit time constant
     * units: second (s)
     */
    @unit("s")
    float directAxisTransientOpenCircuitTimeConstant;

    /**
     * Td0'', direct-axis subtransient open-circuit time constant
     * units: second (s)
     */
    @unit("s")
    float directAxisSubtransientOpenCircuitTimeConstant;

    /**
     * Tq', quadrature-axis transient short-circuit time constant
     * units: second (s)
     */
    @unit("s")
    float quadratureAxisTransientShortCircuitTimeConstant;

    /**
     * Tq'', quadrature-axis subtransient short-circuit time constant
     * units: second (s)
     */
    @unit("s")
    float quadratureAxisSubtransientShortCircuitTimeConstant;

    /**
     * Tq0', quadrature-axis transient open-circuit time constant
     * units: second (s)
     */
    @unit("s")
    float quadratureAxisTransientOpenCircuitTimeConstant;

    /**
     * Tq0'', quadrature-axis subtransient open-circuit time constant
     * units: second (s)
     */
    @unit("s")
    float quadratureAxisSubtransientOpenCircuitTimeConstant;
}; // end struct SynchronousMachineCoefficients


/**
 * Design information for power electronics.
 */
@nested
@extensibility(APPENDABLE)
struct PowerElectronicsInfo {

    /**
     * Minimum rated operating temperature.
     * units: degree Celsius (degC)
     * Change History:
     *  1) Version: 2.0.1, Change: modify, Interoperability: No-xtypes
     *     Removed minimum and maximum range specification.
     */
    @unit("degC")
    float minTemperature;

    /**
     * Maximum rated operating temperature.
     * units: degree Celsius (degC)
     * Change History:
     *  1) Version: 2.0.1, Change: modify, Interoperability: No-xtypes
     *     Removed minimum and maximum range specification.
     */
    @unit("degC")
```

```
    float maxTemperature;
}; // end struct PowerElectronicsInfo


/**
 * Operating state of the power electronics hardware.
 */
@nested
@extensibility(APPENDABLE)
struct PowerElectronicsState {

    /**
     * Present temperature.
     * units: degree Celsius (degC)
     * Change History:
     *  1) Version: 2.0.1, Change: modify, Interoperability: No-xtypes
     *     Removed minimum and maximum range specification.
     */
    @unit("degC")
    float temperature;
}; // end struct PowerElectronicsState


/**
 * Identifies an energy storage cell within a device.
 */
typedef uint16 StorageCellNumber;

/** Number to use for a single-cell device. */
const StorageCellNumber ONLY_CELL = 0;

/** Maximum number of cells that a device may report. */
const StorageCellNumber MAXLEN_storageCells = 50;



/**
 * Power measurement for direct current devices.
 */
@nested
@extensibility(APPENDABLE)
struct StorageCellState {

    /**
     * Identifies the energy storage cell being reported on within a device.
     */
    StorageCellNumber cellNumber;

    /**
     * Cumulative estimate of the energy storage wear, measured as the number of operating charge/discharge cycles.  ↩
          Set to NaN if not available.
     */
    float cycleCounter;

    /**
     * Estimate of present charge as a fraction of rated capacity.  0=empty and 1=full.
     * units: per unit (p.u.)
     * Change History:
     *  1) Version: 2.0.1, Change: modify, Interoperability: No-xtypes
     *     Made the field optional.
     */
    @range(min=0, max=1)
    @unit("p.u.")
    @optional
    float stateOfCharge;

    /**
     * Temperature measurement of the individual storage cell.  Set to NaN if not available.
     * units: degree Celsius (degC)
     * Change History:
     *  1) Version: 2.0.1, Change: modify, Interoperability: No-xtypes
     *     Removed minimum and maximum range specification.
     */
    @unit("degC")
    float temperature;

    /**
     * DC bus voltage across the energy storage cell terminals.  Set to NaN if not available.
     * units: volt (V)
     */
    @unit("V")
    float voltage;
}; // end struct StorageCellState


const uint32 StorageCellStateSequence_MINLEN = 0;
const uint32 StorageCellStateSequence_MAXLEN = MAXLEN_storageCells;
/**
 * A sequence of StorageCellState.
 * Minimum length: 0
 * Maximum length: MAXLEN_storageCells
```

```
 */
typedef sequence<StorageCellState, StorageCellStateSequence_MAXLEN> StorageCellStateSequence;


/**
 * Operating state of the energy storage device or unit.
 */
@nested
@extensibility(APPENDABLE)
struct EnergyStorageState {

    /**
     * Present temperature of the energy storage components, minimum value measured.
     * units: degree Celsius (degC)
     * Change History:
     *  1) Version: 2.0.1, Change: modify, Interoperability: No-xtypes
     *     Removed minimum and maximum range specification.
     */
    @unit("degC")
    float minTemperature;

    /**
     * Present temperature of the energy storage components, average of values measured.
     * units: degree Celsius (degC)
     * Change History:
     *  1) Version: 2.0.1, Change: modify, Interoperability: No-xtypes
     *     Removed minimum and maximum range specification.
     */
    @unit("degC")
    float averageTemperature;

    /**
     * Present temperature of the energy storage components, maximum value measured.
     * units: degree Celsius (degC)
     * Change History:
     *  1) Version: 2.0.1, Change: modify, Interoperability: No-xtypes
     *     Removed minimum and maximum range specification.
     */
    @unit("degC")
    float maxTemperature;

    /**
     * Cumulative time that the energy storage has been actively charging or discharging.  Usually displayed as an ↩
         hour meter.
     * units: second (s)
     */
    @unit("s")
    float runTime;

    /**
     * Information on each storage cell.
     */
    @try_construct(TRIM)
    StorageCellStateSequence storageCells;
}; // end struct EnergyStorageState


/**
 * Design information for an energy storage device or unit.
 */
@nested
@extensibility(APPENDABLE)
struct EnergyStorageInfo {

    /**
     * The high state of charge during continuous operations, only to be exceeded for limited-duration operation. ↩
         Increased wear when charging above this point
     * units: per unit (p.u.)
     * Change History:
     *  1) Version: 2.0.1, Change: modify, Interoperability: No-xtypes
     *     Made the field optional.
     */
    @range(min=0, max=1)
    @unit("p.u.")
    @optional
    float highStateOfCharge;

    /**
     * The low state of charge during continuous operations, only to go below for limited-duration operation. ↩
         Increased wear when discharging below this point.
     * units: per unit (p.u.)
     * Change History:
     *  1) Version: 2.0.1, Change: modify, Interoperability: No-xtypes
     *     Made the field optional.
     */
    @range(min=0, max=1)
    @unit("p.u.")
    @optional
    float lowStateOfCharge;
```

```
    /**
     * Minimum rated operating temperature.
     * units: degree Celsius (degC)
     * Change History:
     *  1) Version: 2.0.1, Change: modify, Interoperability: No-xtypes
     *     Removed minimum and maximum range specification.
     */
    @unit("degC")
    float minTemperature;

    /**
     * Most efficient operating temperature.
     * units: degree Celsius (degC)
     * Change History:
     *  1) Version: 2.0.1, Change: modify, Interoperability: No-xtypes
     *     Removed minimum and maximum range specification.
     */
    @unit("degC")
    float nomTemperature;

    /**
     * Maximum rated operating temperature.
     * units: degree Celsius (degC)
     * Change History:
     *  1) Version: 2.0.1, Change: modify, Interoperability: No-xtypes
     *     Removed minimum and maximum range specification.
     */
    @unit("degC")
    float maxTemperature;
}; // end struct EnergyStorageInfo


/**
 * Design information for a power generator.
 */
@nested
@extensibility(APPENDABLE)
struct GeneratorInfo {

    /**
     * Minimum field current value for normal operation.
     * units: ampere (A)
     */
    @unit("A")
    float minFieldCurrent;

    /**
     * Maximum field current value for normal operation.
     * units: ampere (A)
     */
    @unit("A")
    float maxFieldCurrent;

    /**
     * Maximum stator temperature. Indicates overheating.
     * units: degree Celsius (degC)
     * Change History:
     *  1) Version: 2.0.1, Change: modify, Interoperability: No-xtypes
     *     Removed minimum and maximum range specification.
     */
    @unit("degC")
    float maxStatorTemperature;
}; // end struct GeneratorInfo


/**
 * Report information on AVR and generator condition.
 */
@nested
@extensibility(APPENDABLE)
struct GeneratorState {

    /**
     * field current, as controlled by AVR.
     * units: ampere (A)
     */
    @unit("A")
    float fieldCurrent;

    /**
     * stator temperature.
     * units: degree Celsius (degC)
     */
    @unit("degC")
    float statorTemperature;
}; // end struct GeneratorState


/**
```

```
 * Design information for an engine, such as on a generator set (genset).
 */
@nested
@extensibility(APPENDABLE)
struct EngineInfo {

    /**
     * Minimum oil pressure for the operation of the engine that prevents engine damage.
     * units: pascal (Pa)
     * Change History:
     *  1) Version: 2.0.1, Change: modify, Interoperability: No-xtypes
     *     Removed minimum range specification.
     */
    @unit("Pa")
    float minOilPressure;

    /**
     * Maximum oil pressure for the operation of the engine that prevents engine damage.
     * units: pascal (Pa)
     * Change History:
     *  1) Version: 2.0.1, Change: modify, Interoperability: No-xtypes
     *     Removed minimum range specification.
     */
    @unit("Pa")
    float maxOilPressure;

    /**
     * Minimum coolant temperature for the operation of the engine that prevents engine damage.
     * units: degree Celsius (degC)
     */
    @unit("degC")
    float minCoolantTemperature;

    /**
     * Maximum coolant temperature for the operation of the engine that prevents engine damage.
     * units: degree Celsius (degC)
     */
    @unit("degC")
    float maxCoolantTemperature;

    /**
     * Stall warning speed of the engine. Speeds below this threshold may cause a malfunction or protective shutdown.
     * units: radian per second (rad/s)
     * Change History:
     *  1) Version: 2.0.1, Change: modify, Interoperability: No-xtypes
     *     Removed minimum range specification.
     */
    @unit("rad/s")
    float minEngineSpeed;

    /**
     * Maximum engine speed. I.e. the red line
     * units: radian per second (rad/s)
     * Change History:
     *  1) Version: 2.0.1, Change: modify, Interoperability: No-xtypes
     *     Removed minimum range specification.
     */
    @unit("rad/s")
    float maxEngineSpeed;

    /**
     * Minimum load required to prevent wet stack build up.
     * units: watt (W)
     * Change History:
     *  1) Version: 2.0.1, Change: modify, Interoperability: No-xtypes
     *     Removed minimum range specification.
     */
    @unit("W")
    float minWetStackPreventionLoad;

    /**
     * Minimum load required to clear wet stack build up.
     * units: watt (W)
     * Change History:
     *  1) Version: 2.0.1, Change: modify, Interoperability: No-xtypes
     *     Removed minimum range specification.
     */
    @unit("W")
    float minWetStackMitigationLoad;
}; // end struct EngineInfo


/**
 * Indicate the operating state of an engine.  Engines are not required to support all operating states.
 */
@extensibility(APPENDABLE)
enum EngineOperatingState {
    EOS_UNKNOWN, // Unknown value. This is a generic placeholder representing the case in which an appropriate value ↩
         could not be determined.
    EOS_STOPPED, // Engine is stopped.
    EOS_HEATING, // Engine is stopped and being heated for cold weather start.
    EOS_STARTING, // Engine is starting.
```

```
    EOS_IDLING, // Engine is started and running at a low, non-operational speed.
    EOS_WARMUP, // Engine is started and warming up to operational speed.
    EOS_RUNNING, // Engine is started and fully operational.
    EOS_COOLDOWN, // Engine is started and cooling down in preparation to stop.
    EOS_STOPPING, // Engine is stopping.
    EOS_DISABLED // Engine is disabled.  It may be broken or undergoing maintenance.
}; // enum EngineOperatingState


/**
 * Report key parameters about the state of the engine.
 */
@nested
@extensibility(APPENDABLE)
struct EngineState {

    /**
     * Present operating state.  Should be consistent with the EnergyStartStopState.
     */
    @try_construct(USE_DEFAULT)
    EngineOperatingState operatingState;

    /**
     * Oil pressure.
     * units: pascal (Pa)
     */
    @unit("Pa")
    float oilPressure;

    /**
     * Coolant temperature.
     * units: degree Celsius (degC)
     * Change History:
     *  1) Version: 2.0.1, Change: modify, Interoperability: No-xtypes
     *     Removed minimum and maximum range specification.
     */
    @unit("degC")
    float coolantTemperature;

    /**
     * Speed.
     * units: radian per second (rad/s)
     * Change History:
     *  1) Version: 2.0.1, Change: modify, Interoperability: No-xtypes
     *     Removed minimum range specification.
     */
    @unit("rad/s")
    float speed;

    /**
     * wet stack per unit, mitigation required or indicated at 1.
     * units: per unit (p.u.)
     */
    @range(min=0, max=1)
    @unit("p.u.")
    @optional
    float wetStack;

    /**
     * Engine runtime.
     * units: second (s)
     * Change History:
     *  1) Version: 2.0.1, Change: modify, Interoperability: No-xtypes
     *     Removed minimum range specification.
     *  2) Version: 2.0.1, Change: modify, Interoperability: No-xtypes
     *     Name changed from engineRuntime and units to second.
     */
    @unit("s")
    float runTime;

    /**
     * Fuel consumption rate at the present power level.
     * units: liters per second (L/s)
     */
    @min(0)
    @unit("L/s")
    @optional
    float fuelConsumptionRate;
}; // end struct EngineState


/**
 * Design information for the fuel associated with a power device.
 */
@nested
@extensibility(APPENDABLE)
struct FuelInfo {

    /**
     * The rated fuel capacity of the device.
```

```
     * units: liter (L)
     * Change History:
     *  1) Version: 2.0.1, Change: modify, Interoperability: No-xtypes
     *     Removed minimum range specification.
     */
    @unit("L")
    float maxFuelLevel;

    /**
     * Level where engine automatically shuts down.
     * units: per unit (p.u.)
     */
    @range(min=0, max=1)
    @unit("p.u.")
    float lowFuelLevelCutoff;
}; // end struct FuelInfo


/**
 * Provides dynamic state information regarding fuel levels and pumps.
 */
@nested
@extensibility(APPENDABLE)
struct FuelState {

    /**
     * The current fuel level of the device.
     * units: liter (L)
     */
    @unit("L")
    float fuelLevel;

    /**
     * True indicates that fuel is being pulled from external tank (fuel pump is on). False otherwise or if the ↩
          device does not have a fuel pump.
     */
    boolean fuelPumpRunning;
}; // end struct FuelState


/**
 * Design information for a device capable of load sharing.  Expected step response when directly connected to a load↩
       .  Both acceptance and rejection responses start from steady state and stop when the response settles.  The ↩
       load step occurs at t=0, and the step size should be at least 50% of the rated power capacity of the load ↩
       sharing device.
 */
@nested
@extensibility(APPENDABLE)
struct LoadStepResponse {

    /**
     * Initial condition for the load acceptance step.  Final condition for the load rejection step.
     * units: watt (W)
     */
    @unit("W")
    float loadStepLowRealPower;

    /**
     * Initial condition for the load rejection step.  Final condition for the load acceptance step.
     * units: watt (W)
     */
    @unit("W")
    float loadStepHighRealPower;

    /**
     * Initial condition for the load acceptance step.  Final condition for the load rejection step.
     * units: volt ampere reactive (var)
     */
    @unit("var")
    float loadStepLowReactivePower;

    /**
     * Initial condition for the load rejection step.  Final condition for the load acceptance step.
     * units: volt ampere reactive (var)
     */
    @unit("var")
    float loadStepHighReactivePower;

    /**
     * Plot of expected frequency response to the load acceptance step.
     * units: hertz, second (Hz, s)
     */
    @unit("Hz,␣s")
    Curve2D loadAcceptanceFrequency;

    /**
     * Plot of expected voltage response to the load acceptance step.
     * units: volt, second (V, s)
     */
    @unit("V,␣s")
```

```
    Curve2D loadAcceptanceVoltage;

    /**
     * Plot of expected real power response to the load acceptance step.
     * units: watt, second (W, s)
     */
    @unit("W,␣s")
    Curve2D loadAcceptanceRealPower;

    /**
     * Plot of expected reactive power response to the load acceptance step.
     * units: volt ampere reactive, second (var, s)
     */
    @unit("var,␣s")
    Curve2D loadAcceptanceReactivePower;

    /**
     * Plot of expected frequency response to the load rejection step.
     * units: hertz, second (Hz, s)
     */
    @unit("Hz,␣s")
    Curve2D loadRejectionFrequency;

    /**
     * Plot of expected voltage response to the load rejection step.
     * units: volt, second (V, s)
     */
    @unit("V,␣s")
    Curve2D loadRejectionVoltage;

    /**
     * Plot of expected real power response to the load rejection step.
     * units: watt, second (W, s)
     */
    @unit("W,␣s")
    Curve2D loadRejectionRealPower;

    /**
     * Plot of expected reactive power response to the load rejection step.
     * units: volt ampere reactive, second (var, s)
     */
    @unit("var,␣s")
    Curve2D loadRejectionReactivePower;
}; // end struct LoadStepResponse


const uint32 LoadStepResponseSequence_MINLEN = 0;
const uint32 LoadStepResponseSequence_MAXLEN = 16;
/**
 * A sequence of LoadStepResponse.
 * Minimum length: 0
 * Maximum length: 16
 */
typedef sequence<LoadStepResponse, LoadStepResponseSequence_MAXLEN> LoadStepResponseSequence;



/**
 * Design information for a device capable of load sharing.
 */
@nested
@extensibility(APPENDABLE)
struct LoadSharingInfo {

    /**
     * The power port number.
     */
    PowerPortNumber portNumber;

    /**
     * The device can allow the frequency (AC) and voltage (AC and DC) to sag as load increases.
     */
    boolean supportsDroop;

    /**
     * The droop curves can have multiple piecewise linear segments.
     */
    boolean supportsMultiSegmentDroop;

    /**
     * The device can regulate output power instead of frequency and voltage.
     */
    boolean supportsConstantPower;

    /**
     * Minimum rated real power.
     * units: watt (W)
     */
    @unit("W")
    float minRealPower;

    /**
```

```
     * Maximum rated real power.
     * units: watt (W)
     */
    @unit("W")
    float maxRealPower;

    /**
     * Rated overload output real power as a function of overload duration
     * units: second, watt (s, W)
     */
    @unit("s, W")
    @optional
    Curve2D maxOverloadRealPower;

    /**
     * Minimum rated reactive power.
     * units: volt ampere reactive (var)
     */
    @unit("var")
    float minReactivePower;

    /**
     * Maximum rated reactive power.
     * units: volt ampere reactive (var)
     */
    @unit("var")
    float maxReactivePower;

    /**
     * Maximum rated apparent power (absolute value).
     * units: volt ampere (VA)
     */
    @unit("VA")
    float maxApparentPower;

    /**
     * Rated power factor, usually 0.8.
     */
    float powerFactor;

    /**
     * Expected response to various load steps.
     */
    @optional
    @try_construct(TRIM)
    LoadStepResponseSequence loadResponse;
}; // end struct LoadSharingInfo


const uint32 LoadSharingInfoSequence_MINLEN = 1;
const uint32 LoadSharingInfoSequence_MAXLEN = MAX_PORTS;
/**
 * A sequence of LoadSharingInfo.
 * Minimum length: 1
 * Maximum length: MAX_PORTS
 */
typedef sequence<LoadSharingInfo, LoadSharingInfoSequence_MAXLEN> LoadSharingInfoSequence;



/**
 * Design information for power storage devices.
 */
@nested
@extensibility(APPENDABLE)
struct StorageInfo {

    /**
     * A set of features this device supports.
     */
    @try_construct(TRIM)
    StorageFeatureSequence features;

    /**
     * Rated energy at full state of charge.
     * units: joule (J)
     */
    @unit("J")
    float maxChargeEnergy;

    /**
     * Load sharing capabilities of this device.
     */
    LoadSharingInfo loadSharing;

    /**
     * List of the EnergyStartStopLevels supported by this device for use in EnergyStartStopState or ↩
          EnergyStartStopRequest.
     */
    @try_construct(TRIM)
    EnergyStartStopLevelSequence supportedEnergyStartStopLevels;
}; // end struct StorageInfo
```

```
/**
 * Design information for control hardware.
 */
@nested
@extensibility(APPENDABLE)
struct ControlHardwareInfo {

    /**
     * True indicates that the device has a real time clock. False otherwise.
     */
    boolean hasRealtimeClock;

    /**
     * Minimum rated operating temperature.
     * units: degree Celsius (degC)
     * Change History:
     *  1) Version: 2.0.1, Change: modify, Interoperability: No-xtypes
     *     Removed minimum and maximum range specification.
     */
    @unit("degC")
    float minTemperature;

    /**
     * Maximum rated operating temperature.
     * units: degree Celsius (degC)
     * Change History:
     *  1) Version: 2.0.1, Change: modify, Interoperability: No-xtypes
     *     Removed minimum and maximum range specification.
     */
    @unit("degC")
    float maxTemperature;

    /**
     * Number of network port connectors.
     */
    @min(0)
    uint16 numNetworkPorts;
}; // end struct ControlHardwareInfo


/**
 * Operating state for the control hardware in this device.
 * Topic Usage:
 * - TOPIC_CONTROL_HARDWARE_UPDATE
 */
@topic
@extensibility(APPENDABLE)
struct ControlHardwareUpdate {

    /**
     * The device described by this structure.
     */
    @key
    Identity deviceId;

    /**
     * Time of these measurements.
     */
    ClockMonotonic timestamp;

    /**
     * Central Processing Unit (CPU) usage, as a per unit measure of the whole capacity.
     * units: per unit (p.u.)
     * Change History:
     *  1) Version: 2.0.1, Change: modify, Interoperability: No-xtypes
     *     Made the field optional.
     */
    @range(min=0, max=1)
    @unit("p.u.")
    @optional
    float processorUsage;

    /**
     * Memory utilization of the hardware, as a per unit measure of the whole capacity.
     * units: per unit (p.u.)
     * Change History:
     *  1) Version: 2.0.1, Change: modify, Interoperability: No-xtypes
     *     Made the field optional.
     */
    @range(min=0, max=1)
    @unit("p.u.")
    @optional
    float memoryUsage;

    /**
     * Network utilization of the hardware, as a per unit measure of the whole capacity. If the device has multiple ↩
          network interfaces, this field will describe the network interface utilized by TMS communication protocols.
     * units: per unit (p.u.)
```

```
     * Change History:
     *  1) Version: 2.0.1, Change: modify, Interoperability: No-xtypes
     *     Made the field optional.
     */
    @range(min=0, max=1)
    @unit("p.u.")
    @optional
    float networkUsage;

    /**
     * Disk storage utilization of the hardware, as a per unit measure of the whole capacity.
     * units: per unit (p.u.)
     * Change History:
     *  1) Version: 2.0.1, Change: modify, Interoperability: No-xtypes
     *     Made the field optional.
     */
    @range(min=0, max=1)
    @unit("p.u.")
    @optional
    float storageUsage;

    /**
     * Temperature measurement of the control hardware.
     * units: degree Celsius (degC)
     * Change History:
     *  1) Version: 2.0.1, Change: modify, Interoperability: No-xtypes
     *     Removed minimum and maximum range specification.
     */
    @unit("degC")
    float temperature;
}; // end struct ControlHardwareUpdate


/**
 * Design information for power hardware. Power hardware refers to accessories or supporting components of a power ↩
      device, such as an engine, control boards, etc.
 */
@nested
@extensibility(APPENDABLE)
struct PowerHardwareInfo {

    /**
     * Engine information.
     */
    @optional
    EngineInfo engine;

    /**
     * Fuel information.
     */
    @optional
    FuelInfo fuel;

    /**
     * Generator information.
     */
    @optional
    GeneratorInfo generator;

    /**
     * Energy storage information.
     */
    @optional
    EnergyStorageInfo energyStorage;

    /**
     * Power electronics information.
     */
    @optional
    PowerElectronicsInfo powerElectronics;

    /**
     * Thermal zone information.
     */
    @optional
    ThermalInfo thermal;

    /**
     * Dynamic stability information for synchronous machines.
     */
    @optional
    SynchronousMachineCoefficients synchronousMachineModel;
}; // end struct PowerHardwareInfo


/**
 * Operating state of the power hardware.
 * Topic Usage:
 * - TOPIC_POWER_HARDWARE_UPDATE
 */
```

```
@topic
@extensibility(APPENDABLE)
struct PowerHardwareUpdate {

    /**
     * The device described by this structure.
     */
    @key
    Identity deviceId;

    /**
     * Time of these measurements.
     */
    ClockMonotonic timestamp;

    /**
     * Operating state of the engine if present.
     */
    @optional
    EngineState engine;

    /**
     * Fuel state, if present.
     */
    @optional
    FuelState fuel;

    /**
     * Operating state of a generator, if present.
     */
    @optional
    GeneratorState generator;

    /**
     * Operating state of energy storage if present.
     */
    @optional
    EnergyStorageState energyStorage;

    /**
     * Operating state of active components including inverters and rectifiers, if present.
     */
    @optional
    PowerElectronicsState powerElectronics;

    /**
     * Thermal operating state including heating and cooling components, if present.
     */
    @optional
    ThermalState thermal;
}; // end struct PowerHardwareUpdate


/**
 * Represent an individual alarm or fault condition originating from the indicated sensor or subsystem, as identified↩
       by the affectedIdentityId, portNumber, and suspectParameter.
 */
@nested
@extensibility(APPENDABLE)
struct DiagnosticTroubleCode {

    /**
     * Number shown on the device exterior to represent this port.  Present when this condition pertains to a ↩
          specific power port and absent when it pertains to the entire device.
     */
    @optional
    PowerPortNumber portNumber;

    /**
     * Parameter representing this DTC.
     */
    SuspectParameterNumber suspectParameter;

    /**
     * Type of fault for this parameter.
     */
    @try_construct(USE_DEFAULT)
    FailureModeIndicator failureMode;

    /**
     * Number of times this DTC was raised. This field increments on a state change from inactive to active. In order↩
           to increment, the cleared state must be sensed. The count must not increment simply due to restarts. The ↩
          count should be stored in non-volatile memory in order to preserve it over power cycles. The count ↩
          saturates at 4294967294 $(2^{32}-2)$. The value 4294967295 $(2^{32}-1)$ indicates that the count is not ↩
          available.
     */
    uint32 occurrenceCount;

    /**
     * Timestamp when this DTC was raised or activated.
     */
```

```
    ClockMonotonic timeRaised;

    /**
     * Expected impact of the DTC on grid operation. Intended to support automatic handling of parameters that are ↩
         unknown to the receiving system.
     */
    @try_construct(USE_DEFAULT)
    DtcSeverity severity;

    /**
     * Estimate of when the DTC severity will change, assuming continued operation under present conditions.
     */
    @optional
    ClockMonotonic estimateTime;

    /**
     * Expected DTC severity at the estimateTime.
     */
    @optional
    @try_construct(USE_DEFAULT)
    DtcSeverity estimateSeverity;

    /**
     * Short, human-readable text description of the DTC. This should summarize to the operator what the issue is (e.↩
         g. ``oil pressure'' or ``low oil pressure''). Intended to support human operators for parameters that are ↩
         unknown to the receiving system.
     */
    String32 hint;

    /**
     * The device described by this structure. Present when the device or platform experiencing the alarm or fault ↩
         condition is other than the device publishing the tms::ActiveDiagnosticState.
     * Change History:
     *  1) Version: 2.0.3, Change: add, Interoperability: Yes
     *     Allow a device to report DiagnosticTroubleCodes on
     *     behalf of another device.
     */
    @optional
    Identity affectedIdentityId;
}; // end struct DiagnosticTroubleCode


const uint32 DiagnosticTroubleCodeSequence_MINLEN = 0;
const uint32 DiagnosticTroubleCodeSequence_MAXLEN = 64;
/**
 * A sequence of DiagnosticTroubleCode.
 * Minimum length: 0
 * Maximum length: 64
 */
typedef sequence<DiagnosticTroubleCode, DiagnosticTroubleCodeSequence_MAXLEN> DiagnosticTroubleCodeSequence;



/**
 * Clock synchronized to a global time base.
 */
@nested
@extensibility(APPENDABLE)
struct ClockRealtime {

    /**
     * Epoch.
     */
    uint16 epoch;

    /**
     * integer portion of the timestamp.
     */
    uint32 seconds;

    /**
     * fractional portion of the timestamp.
     */
    uint32 nanoseconds;
}; // end struct ClockRealtime


/**
 * Source of the realtime clock, as enumerated for timeSource in IEEE-1588-2008 Table 7.
 */
@extensibility(APPENDABLE)
enum ClockSource {
    CLOCK_UNKNOWN, // No information is available, either due to design or malfunction.
    CLOCK_ATOMIC, // Direct connection to a calibrated atomic clock.
    CLOCK_GPS, // Satellite-based system.
    CLOCK_RADIO, // Terrestrial radio system.
    CLOCK_PTP, // PTP, IEEE-1588.
    CLOCK_NTP, // NTP, RFC 5905 or SNTP, RFC 4330.
    CLOCK_HAND, // User input.
    CLOCK_OTHER, // Other source.
    CLOCK_INTERNAL // Arbitrary or unknown epoch
}; // enum ClockSource
```

```
/**
 * Simultaneous measurement of both system clocks, and synchronization status for .
 * Topic Usage:
 * - TOPIC_CLOCK_STATE
 */
@topic
@extensibility(APPENDABLE)
struct ClockState {

    /**
     * The device described by this structure.
     */
    @key
    Identity deviceId;

    /**
     * Measured monotonic time, sampled simultaneously with realtime.
     */
    ClockMonotonic monotonic;

    /**
     * Indicate whether the monotonic clock initializes to 0 or accumulates operating time across restarts.
     */
    @try_construct(USE_DEFAULT)
    ClockMonotonicPersistenceType monotonicPersistence;

    /**
     * Measured real time, sampled simultaneously with monotonic.
     */
    ClockRealtime realtime;

    /**
     * Source of the realtime clock, as enumerated for timeSource in IEEE-1588-2008 Table 7.
     */
    @try_construct(USE_DEFAULT)
    ClockSource realtimeSource;

    /**
     * Last time the realtime clock was updated by the source.
     */
    ClockRealtime realtimeSetTime;
}; // end struct ClockState


/**
 * Allows Request / Reply topics to be paired together. Value is requester assigned sequence number sent in a request↩
      and returned in a reply.
 */
typedef uint64 RequestSequence;


/**
 * Key value for a request that targets one power device configuration.
 */
@nested
@extensibility(APPENDABLE)
struct DeviceConfigRequest {

    /**
     * Identity of the device that sent this request.
     */
    Identity requestingDeviceId;

    /**
     * Identity of the device that should receive this request.
     */
    Identity targetDeviceId;

    /**
     * Power device configuration that this request belongs to.
     */
    @try_construct(USE_DEFAULT)
    ConfigId config;
}; // end struct DeviceConfigRequest


/**
 * Key value for a request that targets one device.
 */
@nested
@extensibility(APPENDABLE)
struct DeviceRequest {

    /**
     * Identity of the device that sent this request.
     */
    Identity requestingDeviceId;
```

```
    /**
     * Identity of the device that should receive this request.
     */
    Identity targetDeviceId;
}; // end struct DeviceRequest


/**
 * Identify a power port (often remote)
 */
@nested
@extensibility(APPENDABLE)
struct PowerPortId {

    /**
     * Device containing the port.
     */
    Identity deviceId;

    /**
     * Number shown on the device exterior to represent this port.
     */
    PowerPortNumber portNumber;
}; // end struct PowerPortId


/**
 * Design information for power connectors.
 */
@nested
@extensibility(APPENDABLE)
struct PowerConnectorInfo {

    /**
     * Zero length indicates no features.
     */
    @try_construct(TRIM)
    PowerConnectorFeatureSequence features;

    /**
     * Physical type of this connector.
     */
    @try_construct(USE_DEFAULT)
    PowerConnectorType physicalType;

    /**
     * Polarity of this connector.
     */
    @try_construct(USE_DEFAULT)
    PowerConnectorPolarity polarity;
}; // end struct PowerConnectorInfo


/**
 * Design information for power switches.
 */
@nested
@extensibility(APPENDABLE)
struct PowerSwitchInfo {

    /**
     * Zero length indicates no features.
     */
    @try_construct(TRIM)
    PowerSwitchFeatureSequence features;

    /**
     * Maximum (in/out) interrupt current.
     * units: ampere (A)
     */
    @unit("A")
    float interruptAmperage;
}; // end struct PowerSwitchInfo


/**
 * Design information for the connector, rated circuit, and switching capabilities of a power port.
 */
@nested
@extensibility(APPENDABLE)
struct PowerPortInfo {

    /**
     * Number shown on the device exterior to represent this port.
     */
    PowerPortNumber portNumber;
```

```
/**
 * Type of wiring provided by this port.
 */
@try_construct(USE_DEFAULT)
CircuitWiring wiring;

/**
 *  Anticipated direction of power flow.
 */
@try_construct(USE_DEFAULT)
PowerPortDirectionality directionality;

/**
 * Indicate whether this power port has a switch.
 */
boolean hasSwitch;

/**
 * Indicate whether this power port has an external meter.
 */
boolean hasExternalMeter;

/**
 * Indicate whether this power port has an internal meter.
 */
boolean hasInternalMeter;

/**
 * Indicate whether this port supports the AC or DC SummaryMeasurementUpdate.  Depends on corresponding topic.
 */
boolean hasSummaryMeasurementUpdate;

/**
 * minimum (least output) rated operating amperage.
 * units: ampere (A)
 */
@unit("A")
float minAmperage;

/**
 * maximum (greatest output) rated operating amperage.
 * units: ampere (A)
 */
@unit("A")
float maxAmperage;

/**
 * Maximum rated fault amperage that will not damage the power port.  Must be greater than maxAmperage.  Defined ↩
      in the NEC as the short-circuit current rating (SCCR).
 * units: ampere (A)
 */
@unit("A")
float shortCircuitAmperage;

/**
 * minimum rated operating voltage.
 * units: volt (V)
 */
@unit("V")
float minVoltage;

/**
 * maximum rated operating voltage.
 * units: volt (V)
 */
@unit("V")
float maxVoltage;

/**
 * minimum rated operating frequency, not present for DC, required for AC
 * units: hertz (Hz)
 * Change History:
 *  1) Version: 2.0.2, Change: modify, Interoperability: Yes
 *     Corrected the units from volt to hertz.
 */
@unit("Hz")
@optional
float minFrequency;

/**
 * maximum rated operating frequency, not present for DC, required for AC.
 * units: hertz (Hz)
 * Change History:
 *  1) Version: 2.0.2, Change: modify, Interoperability: Yes
 *     Corrected the units from volt to hertz.
 */
@unit("Hz")
@optional
float maxFrequency;

/**
 * Information about the connector.  Every power port must have a connector.
```

```
     */
    PowerConnectorInfo connectorInfo;

    /**
     * Information about the switch.  If hasSwitch is true, then required, otherwise not present.
     */
    @optional
    PowerSwitchInfo switchInfo;
}; // end struct PowerPortInfo


const uint32 PowerPortInfoSequence_MINLEN = 0;
const uint32 PowerPortInfoSequence_MAXLEN = MAX_PORTS;
/**
 * A sequence of PowerPortInfo.
 * Minimum length: 0
 * Maximum length: MAX_PORTS
 */
typedef sequence<PowerPortInfo, PowerPortInfoSequence_MAXLEN> PowerPortInfoSequence;


/**
 * Indicate the status of a pending power switch transition.  Power switches are not required to support all ↩
      transition codes.
 */
@extensibility(APPENDABLE)
enum PowerSwitchTransitionCode {
    PSTC_UNKNOWN, // Unknown value. This is a generic placeholder representing the case in which an appropriate value↩
         could not be determined.
    PSTC_NONE, // No transition was initiated.
    PSTC_IN_PROGRESS, // Transition is in progress and needs time to complete.  No pending conditions.  For example, ↩
         IEEE C37.2 device function 62.
    PSTC_PENDING_SYNC, // Waiting for the synchronizer to close.  For example, IEEE C37.2 device function 25.
    PSTC_PENDING_SWITCH_CONDITIONS, // Waiting for the PowerSwitchCloseConditions to be met.
    PSTC_PENDING_INTERLOCK, // Waiting for a sequencing condition to be met.  For example, IEEE C37.2 device function↩
         3.
    PSTC_COMPLETE, // The transition completed successfully.
    PSTC_MANUAL_OVERRIDE, // The last transition request was overridden (followed and replaced) by MANUAL action.
    PSTC_AUTO_OVERRIDE, // The last transition request was overridden (followed and replaced) by AUTO logic.
    PSTC_AUTO_LOCK, // The transition was attempted one or more times.  Automatic logic locked out further attempts. ↩
         For example, IEEE C37.2 device function 66.  MC may re-try the request.
    PSTC_MALFUNCTION // The transition failed for an unknown reason.
}; // enum PowerSwitchTransitionCode


/**
 * Describe the present state and most recent transition of a power switch
 */
@nested
@extensibility(APPENDABLE)
struct PowerSwitchStatus {

    /**
     * Present circuit continuity through the switch. May be measured from the switch position or from electrical ↩
          continuity. Continuity is unknown for manual switches with no monitoring capability. Continuity may be ↩
          inconsistent for multi-phase switches.
     */
    @try_construct(USE_DEFAULT)
    CircuitContinuity presentContinuity;

    /**
     * Continuity that the switch is transitioning to, or presentContinuity if no transition.  May be initiated by a ↩
          manual, automatic, or requested action.
     */
    @try_construct(USE_DEFAULT)
    CircuitContinuity futureContinuity;

    /**
     * This power switch is known to be locked and requests will get REPLY_PRECONDITION_FAILED.  A device-level lock ↩
          must be reported on all affected ports.  Unknown locks may REPLY_OK and then PSTC_MALFUNCTION.
     */
    boolean requestLock;

    /**
     * Type of action that caused the switch to be in its presentContinuity.
     */
    @try_construct(USE_DEFAULT)
    PowerSwitchReason presentContinuityReason;

    /**
     * Device or operator that caused the switch to be in its presentContinuity.  Present with an MC identity for ↩
          REQUEST, operator identity for OPERATOR_LOGIN, and this deviceId for AUTO, AUTO_PROTECTION, AUTO_LOAD_SHED,↩
           AUTO_VOLTAGE_LOSS, and AUTO_COMMS_LOSS.  Absent for all other PowerSwitchReason values.
     */
    @optional
    Identity presentContinuityActor;

    /**
     * Progress towards completing a transition, or final result of a requested transition.
     */
    @try_construct(USE_DEFAULT)
```

```
    PowerSwitchTransitionCode futureContinuityReason;
}; // end struct PowerSwitchStatus


/**
 * Describe the state of a grounding circuit.
 */
@nested
@extensibility(APPENDABLE)
struct GroundingCircuitStatus {

    /**
     * Number used to represent this circuit.
     */
    GroundingCircuitNumber groundNumber;

    /**
     * Status of the grounding control switch.
     */
    @optional
    PowerSwitchStatus control;

    /**
     * Status of the grounding pulse switch.
     */
    @optional
    PowerSwitchStatus pulse;

    /**
     * Status of the ground fault detector. GFD_NOT_APPLICABLE if no detector.
     */
    @try_construct(USE_DEFAULT)
    GroundFaultDetection faultDetection;
}; // end struct GroundingCircuitStatus


const uint32 GroundingCircuitStateSequence_MINLEN = 0;
const uint32 GroundingCircuitStateSequence_MAXLEN = MAX_GROUNDING_CIRCUITS;
/**
 * A sequence of Grounding Circuit Status.
 * Minimum length: 0
 * Maximum length: MAX_GROUNDING_CIRCUITS
 */
typedef sequence<GroundingCircuitStatus, GroundingCircuitStateSequence_MAXLEN> GroundingCircuitStateSequence;



/**
 * Report the present state of all the ground circuits in a device.
 * Topic Usage:
 * - TOPIC_DEVICE_GROUNDING_STATE
 */
@topic
@extensibility(APPENDABLE)
struct GroundingCircuitState {

    /**
     * The device described by this structure.
     */
    @key
    Identity deviceId;

    /**
     * Information on each ground circuit.
     */
    @try_construct(TRIM)
    GroundingCircuitStateSequence grounds;
}; // end struct GroundingCircuitState



/**
 * Design information describing a grounding circuit in a device.
 */
@nested
@extensibility(APPENDABLE)
struct GroundingInfo {

    /**
     * Number used to represent this circuit.
     */
    GroundingCircuitNumber groundNumber;

    /**
     * Type of grounding provided by this circuit.
     */
    @try_construct(USE_DEFAULT)
    GroundingDesignType groundType;

    /**
     * List of ports protected by this circuit.
```

```
     */
    @try_construct(TRIM)
    PowerPortNumberSequence protectedPorts;

    /**
     * Not present if no control switch.  Required if control switch exists.  Length 0 indicates no features.
     */
    @try_construct(TRIM)
    PowerSwitchFeatureSequence controlSwitchFeatures;

    /**
     * Not present if no pulse switch.  Required if pulse switch exists.  Length 0 indicates no features.
     */
    @try_construct(TRIM)
    PowerSwitchFeatureSequence pulseSwitchFeatures;
}; // end struct GroundingInfo


const uint32 GroundingInfoSequence_MINLEN = 0;
const uint32 GroundingInfoSequence_MAXLEN = MAX_GROUNDING_CIRCUITS;
/**
 * A sequence of GroundingInfo.
 * Minimum length: 0
 * Maximum length: MAX_GROUNDING_CIRCUITS
 */
typedef sequence<GroundingInfo, GroundingInfoSequence_MAXLEN> GroundingInfoSequence;



/** Maximum number of parameters that a device may contain. */
const uint32 MAX_PARAMETERS = 128;

/**
 * Describe the allowed values for a parameter.
 */
@nested
@extensibility(APPENDABLE)
struct ParameterMetadata {

    /**
     * Unique name of this parameter.
     */
    String1_32 name;

    /**
     * Units of measure such as V, A, Hz, or V/Hz.
     */
    String16 units;

    /**
     * Recommended lower bound. Set to 0 if defining an enumeration, else set to -infinity if no limit, otherwise ↩
          hardMinValue <= nominalMinValue <= nominalMaxValue <= hardMaxValue.
     */
    float nominalMinValue;

    /**
     * Recommended upper bound. Set to number of enumerations - 1 if defining an enumeration, else set to +infinity ↩
          if no limit, otherwise hardMinValue <= nominalMinValue <= nominalMaxValue <= hardMaxValue.
     */
    float nominalMaxValue;

    /**
     * Absolute lower bound. Lower values will be rejected. Set to -infinity if no limit, otherwise hardMinValue <= ↩
          nominalMinValue <= nominalMaxValue <= hardMaxValue.
     */
    float hardMinValue;

    /**
     * Absolute upper bound. Higher values will be rejected. Set to +infinity if no limit, otherwise hardMinValue <= ↩
          nominalMinValue <= nominalMaxValue <= hardMaxValue.
     */
    float hardMaxValue;

    /**
     * Smallest change >= 0 that may have an effect. Set to 1 if defining an enumeration, otherwise set to 0 if no ↩
          limit.
     */
    @min(0)
    float resolution;

    /**
     * Enumeration labels. The sequence length must equal the number of enumerated values, otherwise 0 if not ↩
          defining an enumeration. Each value must be at least 1 character in length. No duplicates are allowed per ↩
          ParameterMetadata.
     */
    @try_construct(TRIM)
    EnumLabelSequence enumLabels;
}; // end struct ParameterMetadata


/**
```

```
 * Provide the value for a parameter.
 */
@nested
@extensibility(APPENDABLE)
struct ParameterValue {

    /**
     * Name of this parameter.
     */
    String1_32 name;

    /**
     * Value of this parameter, with units according to ParameterMetadata.
     */
    float value;
}; // end struct ParameterValue


const uint32 ParameterValueSequence_MINLEN = 0;
const uint32 ParameterValueSequence_MAXLEN = 128;
/**
 * A sequence of ParameterValue.
 * Minimum length: 0
 * Maximum length: 128
 */
typedef sequence<ParameterValue, ParameterValueSequence_MAXLEN> ParameterValueSequence;


const uint32 ParameterMetadataSequence_MINLEN = 0;
const uint32 ParameterMetadataSequence_MAXLEN = 128;
/**
 * A sequence of ParameterMetadata.
 * Minimum length: 0
 * Maximum length: 128
 */
typedef sequence<ParameterMetadata, ParameterMetadataSequence_MAXLEN> ParameterMetadataSequence;



/**
 * Describe the state of a power port.
 */
@nested
@extensibility(APPENDABLE)
struct PowerPortStatus {

    /**
     * Number shown on the device exterior to represent this port.
     */
    PowerPortNumber portNumber;

    /**
     * Indicates whether a power cable is plugged in.
     */
    @try_construct(USE_DEFAULT)
    CableSenseStatus cableStatus;

    /**
     * Indicates that a smart cable is attached. This value is present when a smart cable is connected and absent ↩
          when it is disconnected.
     */
    @optional
    Identity cableId;

    /**
     * Indicates that this port is connected to a remote port.
     */
    @optional
    PowerPortId connection;

    /**
     * Indicates the state of a switch associated with this port.  Must be present when PowerPortInfo.hasSwitch is ↩
          true and absent otherwise.
     */
    @optional
    PowerSwitchStatus switchStatus;
}; // end struct PowerPortStatus


const uint32 PowerPortStatusSequence_MINLEN = 0;
const uint32 PowerPortStatusSequence_MAXLEN = MAX_PORTS;
/**
 * A sequence of PowerPortStatus.
 * Minimum length: 0
 * Maximum length: MAX_PORTS
 */
typedef sequence<PowerPortStatus, PowerPortStatusSequence_MAXLEN> PowerPortStatusSequence;



/**
 * Specifies the operator priority for a single device or power port.
```

```
 */
@nested
@extensibility(APPENDABLE)
struct OperatorPriority {

    /**
     * The operator priority type.
     */
    @try_construct(USE_DEFAULT)
    OperatorPriorityType priorityType;

    /**
     * Rank values indicate priority order, where 0 is the highest priority and increasing values are lower priority.↩
          The numericRank is used when priorityType is OPT_NUMERIC_RANK.
     */
    int16 numericRank;
}; // end struct OperatorPriority



/**
 * Specify the device power port level operator intent.
 */
@nested
@extensibility(APPENDABLE)
struct PowerPortIntent {

    /**
     * The device described by this structure.
     */
    Identity deviceId;

    /**
     * Number shown on the device exterior to represent this port.
     */
    PowerPortNumber portNumber;

    /**
     * The operator assigned priority of this power port.
     */
    OperatorPriority priority;
}; // end struct PowerPortIntent



/**
 * Identify a connection between power ports
 */
@nested
@extensibility(APPENDABLE)
struct PowerConnectionId {

    /**
     * One end of the connection.
     */
    PowerPortId portA;

    /**
     * The other end of the connection.  Absent if not available.  See PowerConnectionDetectionType.
     */
    @optional
    PowerPortId portB;
}; // end struct PowerConnectionId



/**
 * Design information for power sources.
 */
@nested
@extensibility(APPENDABLE)
struct SourceInfo {

    /**
     * A set of features this device supports.
     */
    @try_construct(TRIM)
    SourceFeatureSequence features;

    /**
     * Load sharing capabilities of this device.
     */
    LoadSharingInfo loadSharing;

    /**
     * List of the EnergyStartStopLevels supported by this device for use in EnergyStartStopState or ↩
         EnergyStartStopRequest.
     */
    @try_construct(TRIM)
    EnergyStartStopLevelSequence supportedEnergyStartStopLevels;
}; // end struct SourceInfo
```

```
/**
 * Active diagnostic messages.
 * Topic Usage:
 * - TOPIC_ACTIVE_DIAGNOSTIC_STATE
 */
@topic
@extensibility(APPENDABLE)
struct ActiveDiagnosticState {

    /**
     * The device described by this structure.
     */
    @key
    Identity deviceId;

    /**
     * Active trouble codes. A given suspect parameter can only appear once in this sequence.
     */
    @try_construct(TRIM)
    DiagnosticTroubleCodeSequence codes;

    /**
     * Count of trouble codes that were truncated from the codes because the maximum sequence length was exceeded. A ↩
          count of 0 indicates that all active trouble codes are listed.
     */
    uint32 overflow;
}; // end struct ActiveDiagnosticState


/**
 * Report the result of a request that required authorization.
 * Topic Usage:
 * - TOPIC_AUTHORIZATION_TO_ENERGIZE_RESULT
 */
@topic
@extensibility(APPENDABLE)
struct AuthorizationToEnergizeResult {

    /**
     * Copy of the corresponding AuthorizationToEnergizeRequest.requestId.
     */
    @key
    GridRequest relatedRequestId;

    /**
     * Copy of the corresponding AuthorizationToEnergizeRequest.sequenceId.
     */
    RequestSequence relatedSequenceId;

    /**
     * Copy of AuthorizationToEnergizeReply.deviceId if a response was received, else absent.
     */
    @optional
    Identity authorizationDeviceId;

    /**
     * Copy of the corresponding AuthorizationToEnergizeRequest.energizeRequestingDeviceId.
     */
    Identity energizeRequestingDeviceId;

    /**
     * Copy of the corresponding AuthorizationToEnergizeRequest.energizeSequenceId.
     */
    tms::RequestSequence energizeSequenceId;

    /**
     * Copy of the AuthorizationToEnergizeReply.userId if a response was received and the field was present, else ↩
          absent.
     */
    @optional
    Identity userId;

    /**
     * True if the authorization was accepted, the request will be executed, and both relatedRequestId and ↩
          relatedSequenceId are valid.
     */
    boolean accepted;

    /**
     * True confirms that an AuthorizationToEnergizeReply was received. False indicates a response due to ↩
          authorization timeout, and therefore authorizationDeviceId is absent.
     */
    boolean responseReceived;

    /**
     * True confirms that energizeRequestingDeviceId, energizeSequenceId, and powerPorts in the ↩
          AuthorizationToEnergizeReply matched a pending energize request. False otherwise.
     */
    boolean energizeRequestValid;
```

```
    /**
     * True confirms that the AuthorizationToEnergizeReply.userId and AuthorizationToEnergizeReply.timeOfReview are ↩
          valid. False otherwise.
     */
    boolean authorizationReviewValid;

    /**
     * Local monotonic clock when the outcome decision was made.
     */
    ClockMonotonic outcomeMonotonic;

    /**
     * Local realtime clock when the outcome decision was made.
     */
    ClockRealtime outcomeRealtime;
}; // end struct AuthorizationToEnergizeResult


/**
 * Request authorization for a manual black start command.
 * Topic Usage:
 * - TOPIC_AUTHORIZATION_TO_ENERGIZE_REQUEST
 */
@topic
@extensibility(APPENDABLE)
struct AuthorizationToEnergizeRequest {

    /**
     * Identity of this request.
     */
    @key
    GridRequest requestId;

    /**
     * GridRequest sequence data used to associate a request and returning reply.
     */
    tms::RequestSequence sequenceId;

    /**
     * Copy of the requestingDeviceId from the request that requires authorization.
     */
    Identity energizeRequestingDeviceId;

    /**
     * Copy of the sequenceId from the request that requires authorization.
     */
    tms::RequestSequence energizeSequenceId;

    /**
     * All power ports that would be energized by executing this request.
     */
    @try_construct(TRIM)
    PowerPortNumberSequence powerPorts;
}; // end struct AuthorizationToEnergizeRequest


/**
 * Provide the operator's response to an authorization request.
 * Topic Usage:
 * - TOPIC_AUTHORIZATION_TO_ENERGIZE_REPLY
 */
@topic
@extensibility(APPENDABLE)
struct AuthorizationToEnergizeReply {

    /**
     * Identity of the device that processed the corresponding AuthorizationToEnergizeRequest and sent this response.
     */
    Identity deviceId;

    /**
     * Copy of the corresponding AuthorizationToEnergizeRequest.requestId.
     */
    @key
    GridRequest relatedRequestId;

    /**
     * Copy of the corresponding AuthorizationToEnergizeRequest.sequenceId.
     */
    RequestSequence relatedSequenceId;

    /**
     * Copy of the corresponding AuthorizationToEnergizeRequest.energizeRequestingDeviceId.
     */
    Identity energizeRequestingDeviceId;

    /**
     * Copy of the corresponding AuthorizationToEnergizeRequest.energizeSequenceId.
     */
```

```
    tms::RequestSequence energizeSequenceId;

    /**
     * Copy of the AuthorizationToEnergizeRequest.powerPorts.
     */
    @try_construct(TRIM)
    PowerPortNumberSequence powerPorts;

    /**
     * Identity of the user that provided this response.  Absent if not available.
     */
    @optional
    Identity userId;

    /**
     * True if the request is authorized to proceed.
     */
    boolean accept;

    /**
     * True if the request must not be executed.
     */
    boolean deny;

    /**
     * Time the user provided this response.
     */
    ClockRealtime timeOfReview;
}; // end struct AuthorizationToEnergizeReply


/**
 * Change the nickname associated with a Identity.
 * Topic Usage:
 * - TOPIC_IDENTITY_NICKNAME_REQUEST
 */
@topic
@extensibility(APPENDABLE)
struct IdentityNicknameRequest {

    /**
     * Identity of this request.
     */
    @key
    DeviceRequest requestId;

    /**
     * Request sequence data used to associate a request and returning reply.
     */
    tms::RequestSequence sequenceId;

    /**
     * New nickname to be used for this Identity.
     */
    String32 nickname;
}; // end struct IdentityNicknameRequest


/**
 * Small image representing a device or platform.
 * Topic Usage:
 * - TOPIC_DEVICE_ICON
 */
@topic
@extensibility(APPENDABLE)
struct DeviceIcon {

    /**
     * The device or platform illustrated by this image. Only the platform representative may publish a platform icon↩
         .
     * Change History:
     *  1) Version: 2.0.3, Change: modify, Interoperability: Yes
     *     Update the deviceId description to allow the platform
     *     representative to publish a platform icon.
     */
    @key
    Identity deviceId;

    /**
     * Type of the image format. Currently, the only supported type is 'image/png'.  See the IANA Media Type list - ↩
          http://www.iana.org/assignments/media-types/media-types.xhtml.
     */
    String32 mimeType;

    /**
     * Contents of the image with a resolution of 64x64 and transparent background.  The image should be clearly ↩
          visible on a white or black background.
     */
    @try_construct(TRIM)
    OctetSequence data;
```

```
}; // end struct DeviceIcon


/**
 * Specify the device level operator intent.
 */
@nested
@extensibility(APPENDABLE)
struct DeviceIntent {

    /**
     * The device described by this structure.
     */
    Identity deviceId;

    /**
     * The device Battle Short state.
     */
    boolean battleShort;

    /**
     * The operator assigned priority of the device.
     */
    OperatorPriority priority;
}; // end struct DeviceIntent


const uint32 DeviceIntentSequence_MINLEN = 0;
const uint32 DeviceIntentSequence_MAXLEN = 200;
/**
 * A sequence of DeviceIntent.
 * Minimum length: 0
 * Maximum length: 200
 */
typedef sequence<DeviceIntent, DeviceIntentSequence_MAXLEN> DeviceIntentSequence;



/**
 * Request to change the device control parameters as defined by the DeviceInfo.controlParameters.
 * Topic Usage:
 * - TOPIC_CONTROL_PARAMETER_REQUEST
 */
@topic
@extensibility(APPENDABLE)
struct ControlParameterRequest {

    /**
     * Identity of this request.
     */
    @key
    DeviceConfigRequest requestId;

    /**
     * Request sequence data used to associate a request and returning reply.
     */
    tms::RequestSequence sequenceId;

    /**
     * Selected parameter values to update (may be partial or full list).
     */
    @try_construct(TRIM)
    ParameterValueSequence parameters;
}; // end struct ControlParameterRequest



/**
 * State of control parameters.
 * Topic Usage:
 * - TOPIC_CONTROL_PARAMETER_STATE
 */
@topic
@extensibility(APPENDABLE)
struct ControlParameterState {

    /**
     * The device described by this structure.
     */
    @key
    Identity deviceId;

    /**
     * Configuration that these values belong to.
     */
    @try_construct(USE_DEFAULT)
    ConfigId config;

    /**
     * A full list of the parameters available on this device.
     */
```

```
    @try_construct(TRIM)
    ParameterValueSequence parameters;
}; // end struct ControlParameterState


/**
 * Report the present state of all the power ports in a device.
 * Topic Usage:
 * - TOPIC_POWER_PORT_STATE
 */
@topic
@extensibility(APPENDABLE)
struct PowerPortState {

    /**
     * The device described by this structure.
     */
    @key
    Identity deviceId;

    /**
     * Information on each power port.
     */
    @try_construct(TRIM)
    PowerPortStatusSequence powerPorts;
}; // end struct PowerPortState


/**
 * Design information for the distribution device role.
 */
@nested
@extensibility(APPENDABLE)
struct DistributionInfo {

    /**
     * A set of features this device supports.
     */
    @try_construct(TRIM)
    DistributionFeatureSequence features;
}; // end struct DistributionInfo


/**
 * Provide a short, human-readable name for an Identity.
 * Topic Usage:
 * - TOPIC_IDENTITY_NICKNAME_STATE
 */
@topic
@extensibility(APPENDABLE)
struct IdentityNicknameState {

    /**
     * Identity of the device that sent this structure.
     */
    @key
    Identity deviceId;

    /**
     * Identity using the nickname. It must match a DeviceInfo.deviceId or DeviceInfo.product.platformId.
     */
    @key
    Identity identityId;

    /**
     * Nickname given to this identityId.
     */
    String32 nickname;
}; // end struct IdentityNicknameState


/**
 * Remotely transition a grounding circuit switch.
 * Topic Usage:
 * - TOPIC_GROUNDING_CIRCUIT_REQUEST
 */
@topic
@extensibility(APPENDABLE)
struct GroundingCircuitRequest {

    /**
     * Identity of this request.
     */
    @key
    DeviceConfigRequest requestId;

    /**
     * Request sequence data used to associate a request and returning reply.
```

```
     */
    tms::RequestSequence sequenceId;

    /**
     * Grounding circuit containing the switch that should change state.
     */
    GroundingCircuitNumber groundNumber;

    /**
     * Desired continuity through the grounding control switch.
     */
    @try_construct(USE_DEFAULT)
    DesiredCircuitContinuity control;

    /**
     * Desired continuity through the grounding pulse switch.
     */
    @try_construct(USE_DEFAULT)
    DesiredCircuitContinuity pulse;
}; // end struct GroundingCircuitRequest


/**
 * Periodic indication of device availability.
 * Topic Usage:
 * - TOPIC_HEARTBEAT
 */
@topic
@extensibility(APPENDABLE)
struct Heartbeat {

    /**
     * The device described by this structure.
     */
    @key
    Identity deviceId;

    /**
     * A counter that starts at 0 and increments by 1 for each new heartbeat.
     */
    uint32 sequenceNumber;
}; // end struct Heartbeat


/**
 * Design information for load devices.
 */
@nested
@extensibility(APPENDABLE)
struct LoadInfo {

    /**
     * A set of features this device supports.
     */
    @try_construct(TRIM)
    LoadFeatureSequence features;

    /**
     * Maximum real power utilization from this Load.
     * units: watt (W)
     */
    @unit("W")
    float maxRealPower;

    /**
     * Maximum reactive power utilization from this Load.
     * units: volt ampere reactive (var)
     */
    @unit("var")
    float maxReactivePower;
}; // end struct LoadInfo


/**
 * Latest values for device metric parameters.
 * Topic Usage:
 * - TOPIC_METRIC_PARAMETER_STATE
 */
@topic
@extensibility(APPENDABLE)
struct MetricParameterState {

    /**
     * The device described by this structure.
     */
    @key
    Identity deviceId;

    /**
```

```
     * A full list of read-only parameter values on this device.
     */
    @try_construct(TRIM)
    ParameterValueSequence metricParameters;
}; // end struct MetricParameterState


/**
 * Specify the microgrid level load sharing intent.
 */
@nested
@extensibility(APPENDABLE)
struct LoadSharingIntent {

    /**
     * Per unit of used capacity that triggers the MC to add additional power source to the microgrid. Per unit is in the range 0 - 1.
     * units: per unit (p.u.)
     */
    @range(min=0, max=1)
    @unit("p.u.")
    float increaseCapacityThreshold;

    /**
     * Per unit of used capacity that triggers the MC to remove a power source from the microgrid. Per unit is in the range 0 - 1 and must be less than increaseCapacityThreshold.
     * units: per unit (p.u.)
     */
    @range(min=0, max=1)
    @unit("p.u.")
    float decreaseCapacityThreshold;

    /**
     * The time duration the decreaseCapacityThreshold must be met before the Microgrid Controller removes a power source from the microgrid. This duration prevents power sources from being added and removed in rapid succession.
     * units: second (s)
     */
    @unit("s")
    float decreaseCapacityDuration;

    /**
     * The priority above which the Microgrid Controller is authorized to trigger load curtailment. Load curtailment is accomplished by stopped a load device. A value of OPT_ALWAYS_OPERATE disables load curtailment.
     */
    OperatorPriority priorityLoadCurtailmentAllowed;

    /**
     * The priority above which the Microgrid Controller is authorized to trigger port power load shedding. Port power load shedding is accomplished by opening a power port on a distribution device. A value of OPT_ALWAYS_OPERATE disables circuit load shedding.
     */
    OperatorPriority priorityPowerPortSheddingAllowed;
}; // end struct LoadSharingIntent


/**
 * Specify the storage device intent.
 */
@nested
@extensibility(APPENDABLE)
struct StorageIntent {

    /**
     * The desired total state of charge to hold in reserve.
     * units: per unit (p.u.)
     */
    @range(min=0, max=1)
    @unit("p.u.")
    float reservedStateOfCharge;

    /**
     * Time till the reservedStateOfCharge is reached and maintained.
     */
    ClockMonotonic timeTillReservedSoc;

    /**
     * The priority above which the MC is authorized to trigger load shedding. Load shedding is accomplished by stopped a smart load device. A value of OPT_ALWAYS_OPERATE disables load curtailment.
     */
    OperatorPriority priorityLoadSheddingAllowed;

    /**
     * The priority above which the MC is authorized to trigger power port load curtailment. Power port shedding is accomplished by opening a power port on a distribution device. A value of OPT_ALWAYS_OPERATE disables circuit load shedding.
     */
    OperatorPriority priorityPowerPortSheddingAllowed;

    /**
```

```
     * The list of storage devices that should reach the reservedStateOfCharge.  At empty list indicates all ↩
         available storage devices.
     */
    @try_construct(TRIM)
    IdentitySequence reservedDeviceIds;
}; // end struct StorageIntent


/**
 * Specify the microgrid level operator intent.
 */
@nested
@extensibility(APPENDABLE)
struct MicrogridIntent {

    /**
     * Overall operating mode for the microgrid.
     */
    @try_construct(USE_DEFAULT)
    OperatingMode operatingModeValue;

    /**
     * Load sharing Microgrid Controller directives.
     */
    LoadSharingIntent loadSharingIntentValue;

    /**
     * Present if the operator provides Storage device Microgrid Controller directives.
     */
    @optional
    StorageIntent storageIntentValue;
}; // end struct MicrogridIntent


const uint32 PowerPortIntentSequence_MINLEN = 0;
const uint32 PowerPortIntentSequence_MAXLEN = 4000;
/**
 * A sequence of PowerPortIntent.
 * Minimum length: 0
 * Maximum length: 4000
 */
typedef sequence<PowerPortIntent, PowerPortIntentSequence_MAXLEN> PowerPortIntentSequence;


/**
 * Specifies a complete set of operator directives used by the MC to control the microgrid.
 */
@nested
@extensibility(APPENDABLE)
struct OperatorIntent {

    /**
     * The device that defined this structure.
     */
    GridRequest requestId;

    /**
     * The operator intent type.
     */
    @try_construct(USE_DEFAULT)
    OperatorIntentType intentType;

    /**
     * The microgrid level directives.
     */
    MicrogridIntent microgrid;

    /**
     * The device level directives.
     */
    @try_construct(TRIM)
    DeviceIntentSequence devices;

    /**
     * The power port level directives.
     */
    @try_construct(TRIM)
    PowerPortIntentSequence powerPorts;
}; // end struct OperatorIntent


/**
 * Used to request activation of an OperatorIntent.
 * Topic Usage:
 * - TOPIC_OPERATOR_INTENT_REQUEST
 */
@topic
@extensibility(APPENDABLE)
struct OperatorIntentRequest {
```

```
    /**
     * The device sending this request.
     */
    @key
    GridRequest requestId;

    /**
     * Request sequence data used to associate a request and returning reply.
     */
    tms::RequestSequence sequenceId;

    /**
     * The OperatorIntent to activate.
     */
    OperatorIntent desiredOperatorIntent;
}; // end struct OperatorIntentRequest


/**
 * The active OperatorIntent.
 * Topic Usage:
 * - TOPIC_OPERATOR_INTENT_STATE
 */
@topic
@extensibility(APPENDABLE)
struct OperatorIntentState {

    /**
     * The device described by this structure.
     */
    @key
    Identity deviceId;

    /**
     * The complete active OperatorIntent.
     */
    OperatorIntent activeOperatorIntent;
}; // end struct OperatorIntentState


/**
 * Key value for a request that targets one power port of a device.
 */
@nested
@extensibility(APPENDABLE)
struct PowerPortConfigRequest {

    /**
     * Identity of the device that sent this request.
     */
    Identity requestingDeviceId;

    /**
     * Identity of the device that should received this request.
     */
    Identity targetDeviceId;

    /**
     * Power device configuration that this request belongs to.
     */
    @try_construct(USE_DEFAULT)
    ConfigId config;

    /**
     * Power port number that should change state.
     */
    PowerPortNumber portNumber;
}; // end struct PowerPortConfigRequest


/**
 * Identify links between devices
 */
@nested
@extensibility(APPENDABLE)
struct PowerConnection {

    /**
     * Connection described by this object.
     */
    PowerConnectionId connectionId;

    /**
     * Technique used to detect this connection.
     */
    @try_construct(USE_DEFAULT)
    PowerConnectionDetectionType detectionType;
```

```
    /**
     * Device or user that detected this connection.
     */
    Identity detectionSource;

    /**
     * Confidence in the link detection. -1 = confident the link does not exist, 0 = no opinion, 1 = confident the ↩
         link exists.
     */
    @range(min=-1, max=1)
    float detectionConfidence;
}; // end struct PowerConnection


const uint32 PowerConnectionSequence_MINLEN = 0;
const uint32 PowerConnectionSequence_MAXLEN = 200;
/**
 * A sequence of PowerConnection.
 * Minimum length: 0
 * Maximum length: 200
 */
typedef sequence<PowerConnection, PowerConnectionSequence_MAXLEN> PowerConnectionSequence;



/**
 * List all power connections known to a device or user
 * Topic Usage:
 * - TOPIC_OPERATOR_POWER_CONNECTION_STATE
 * - TOPIC_DISCOVERED_POWER_CONNECTION_STATE
 * - TOPIC_MICROGRID_POWER_CONNECTION_STATE
 */
@topic
@extensibility(APPENDABLE)
struct PowerConnectionState {

    /**
     * The reporting device or user.
     */
    @key
    Identity deviceId;

    /**
     * List of power connections being reported by this device or user.
     */
    @try_construct(TRIM)
    PowerConnectionSequence connections;
}; // end struct PowerConnectionState



/**
 * The associated request to close a switch should wait until all of these conditions are met.  The MC may use these ↩
      to override any built-in device defaults, providing looser bounds to recover from abnormal conditions or ↩
      tighter bounds to prevent a large transient.  These conditions do not modify any built-in protections such as a↩
       synchronization check.  Requires PSF_SWITCH_CONDITIONS in PowerSwitchInfo.features
 */
@nested
@extensibility(APPENDABLE)
struct PowerSwitchCloseConditions {

    /**
     * Minimum external RMS AC or DC voltage.
     * units: volt (V)
     */
    @unit("V")
    float minVoltage;

    /**
     * Maximum external RMS AC or DC voltage.
     * units: volt (V)
     */
    @unit("V")
    float maxVoltage;

    /**
     * Minimum external AC frequency.  NaN for DC.
     * units: hertz (Hz)
     */
    @unit("Hz")
    float minFrequency;

    /**
     * Maximum external AC frequency.  NaN for DC.
     * units: hertz (Hz)
     */
    @unit("Hz")
    float maxFrequency;
}; // end struct PowerSwitchCloseConditions
```

```
/**
 * Remotely transition a power port switch.
 * Topic Usage:
 * - TOPIC_POWER_SWITCH_REQUEST
 */
@topic
@extensibility(APPENDABLE)
struct PowerSwitchRequest {

    /**
     * Identity of this request.
     */
    @key
    PowerPortConfigRequest requestId;

    /**
     * Request sequence data used to associate a request and returning reply.
     */
    tms::RequestSequence sequenceId;

    /**
     * Desired continuity through the switch.
     */
    @try_construct(USE_DEFAULT)
    DesiredCircuitContinuity continuity;

    /**
     * Conditions that may delay switch closure.  Can only be present for power ports that support ↩
          PSF_SWITCH_CONDITIONS.  Must be absent for DCC_OPEN or DCC_NO_CHANGE requests.
     */
    @optional
    PowerSwitchCloseConditions switchConditions;
}; // end struct PowerSwitchRequest



/**
 * Topic information. Defines which conditional and optional topics are supported by this device.
 */
@nested
@extensibility(APPENDABLE)
struct TopicInfo {

    /**
     * Release version of data topics and types.  See tms.TMS_VERSION.
     */
    String32 dataModelVersion;

    /**
     * List of conditional topic names that are published by this device.
     */
    @try_construct(TRIM)
    TopicList publishedConditionalTopics;

    /**
     * List of optional topic names that are published by this device.
     */
    @try_construct(TRIM)
    TopicList publishedOptionalTopics;

    /**
     * List of request topic names that are subscribed to by this device.
     */
    @try_construct(TRIM)
    TopicList supportedRequestTopics;

    /**
     * List of extension topic names that are published by this device.  Extension topics are not specified in the ↩
          standard.
     */
    @optional
    @try_construct(TRIM)
    TopicList extensionTopics;
}; // end struct TopicInfo



/**
 * Design information for active conversion load sharing devices.
 */
@nested
@extensibility(APPENDABLE)
struct ActiveConversionInfo {

    /**
     * Load sharing capabilities of each power port.
     */
    @try_construct(TRIM)
    LoadSharingInfoSequence loadSharing;
}; // end struct ActiveConversionInfo
```

```
/**
 * Design information for conversion taps.
 */
@nested
@extensibility(APPENDABLE)
struct ConversionTapInfo {

    /**
     * Tap changer position.
     */
    TapNumber tapId;

    /**
     * Phase shift between this port and the internal bus.
     */
    float phaseShift;

    /**
     * Voltage ratio between this port and the internal bus.
     */
    float voltageRatio;
}; // end struct ConversionTapInfo


const uint32 ConversionTapInfoSequence_MINLEN = 1;
const uint32 ConversionTapInfoSequence_MAXLEN = MAX_TAPS;
/**
 * A sequence of ConversionTapInfo.
 * Minimum length: 1
 * Maximum length: MAX_TAPS
 */
typedef sequence<ConversionTapInfo, ConversionTapInfoSequence_MAXLEN> ConversionTapInfoSequence;



/**
 * Design information for power port conversation devices.
 */
@nested
@extensibility(APPENDABLE)
struct PowerPortConversionInfo {

    /**
     * The power port number this structure represents.
     */
    PowerPortNumber portNumber;

    /**
     * Sequence to support tap changers
     */
    @try_construct(TRIM)
    ConversionTapInfoSequence conversionTaps;

    /**
     * False requires (or causes) circuit interrupt to change, true can change under rated load.
     */
    boolean changeUnderLoad;
}; // end struct PowerPortConversionInfo


const uint32 PowerPortConversionInfoSequence_MINLEN = 1;
const uint32 PowerPortConversionInfoSequence_MAXLEN = MAX_PORTS;
/**
 * A sequence of PowerPortConversionInfo.
 * Minimum length: 1
 * Maximum length: MAX_PORTS
 */
typedef sequence<PowerPortConversionInfo, PowerPortConversionInfoSequence_MAXLEN> PowerPortConversionInfoSequence;



/**
 * Design information for a passive energy converter.
 */
@nested
@extensibility(APPENDABLE)
struct PassiveConversionInfo {

    /**
     * Information for each port on the device.
     */
    @try_construct(TRIM)
    PowerPortConversionInfoSequence portConversion;
}; // end struct PassiveConversionInfo



/**
 * Design information for power conversion devices. Each power port must be listed once in either activeConversion or↩
       passiveConversion but not both.
 */
```

```
@nested
@extensibility(APPENDABLE)
struct ConversionInfo {

    /**
     * A set of features this device supports.
     */
    @try_construct(TRIM)
    ConversionFeatureSequence features;

    /**
     * Power regulation for active devices.
     */
    @optional
    ActiveConversionInfo activeConversion;

    /**
     * Power regulation for passive devices.
     */
    @optional
    PassiveConversionInfo passiveConversion;
}; // end struct ConversionInfo



/**
 * Information on how a product is marketed.
 */
@nested
@extensibility(APPENDABLE)
struct ProductInfo {

    /**
     * Supply type of the product.
     */
    NatoStockNumber nsn;

    /**
     * Commercial type of the product.
     */
    GlobalTradeItemNumber gtin;

    /**
     * Name of the product manufacturer.
     */
    String32 manufacturerName;

    /**
     * Name of the product model.
     */
    String32 modelName;

    /**
     * Manufacturer number for the product model.  May include non-numeric characters.
     */
    String32 modelNumber;

    /**
     * Unique production number for a specific product. Large enough to contain a variety of formats, including the ↩
          GS1 serial number (AI 21).
     */
    String32 serialNumber;

    /**
     * Software Version.
     */
    String32 softwareVersion;

    /**
     * A platform groups TMS products into a larger physical or logical package.  This grouping may be shown on the ↩
          microgrid dashboard, and it may supplement topology information in the microgrid controller.  If a product ↩
          is re-assigned to a different platform, it must resend the DeviceInfo as if it restarted.  Present when ↩
          such a grouping is known to the product and absent otherwise.
     */
    @optional
    Identity platformId;

    /**
     * When true, indicates this device acts as the platform representative responsible for publishing platform-↩
          specific information such as nickname or device icon. Only one device within a platform may set this to ↩
          true.
     * Change History:
     *  1) Version: 2.0.3, Change: add, Interoperability: Yes
     *     Added boolean to indicate the platform representative.
     */
    @optional
    boolean platformRepresentative;
}; // end struct ProductInfo



/**
```

```
 * Design information for a microgrid dashboard.
 */
@nested
@extensibility(APPENDABLE)
struct MicrogridDashboardInfo {

    /**
     * A set of features this device supports.
     */
    @try_construct(TRIM)
    MicrogridDashboardFeatureSequence features;
}; // end struct MicrogridDashboardInfo



/**
 * Design information for a microgrid controller.
 */
@nested
@extensibility(APPENDABLE)
struct MicrogridControllerInfo {

    /**
     * A set of features this device supports.
     */
    @try_construct(TRIM)
    MicrogridControllerFeatureSequence features;

    /**
     * The numeric priority ranking of this MC where 0 is the highest priority and increasing values are lower priority.
     * Change History:
     *  1) Version: 2.0.1, Change: add, Interoperability: No-Business Logic
     *     The active Microgrid Controller selection has changed to
     *     consider this field.  As a result, backwards
     *     compatibility is not guaranteed as older power devices
     *     will not implement the updated logic.
     */
    uint16 priorityRanking;
}; // end struct MicrogridControllerInfo



/**
 * Indicate the type of success or failure of a request.
 */
@nested
@extensibility(APPENDABLE)
struct ReplyStatus {

    /**
     * Indicate whether the request was accepted or rejected.
     */
    @try_construct(USE_DEFAULT)
    ReplyCode code;

    /**
     * Short textual description of the reply code intended for human operators.
     */
    String32 reason;
}; // end struct ReplyStatus



/**
 * Respond whether a request was accepted for execution or rejected.  Supported requestId types include GridRequest, DeviceRequest, DeviceConfigRequest, and PowerPortConfigRequest.
 * Topic Usage:
 * - TOPIC_REPLY
 */
@topic
@extensibility(APPENDABLE)
struct Reply {

    /**
     * Copy of requestingDeviceId from the requestId.
     */
    @key
    Identity requestingDeviceId;

    /**
     * Identity of the device that sent this Reply.
     */
    @key
    Identity targetDeviceId;

    /**
     * Copy of configId from a requestId of type DeviceConfigRequest or PowerPortConfigRequest, otherwise ConfigId.CONFIG_UNKNOWN.
     */
    @try_construct(USE_DEFAULT)
    ConfigId config;
```

```
    /**
     * Copy of portNumber from a requestId of type PowerPortConfigRequest, otherwise PowerPortNumber.↩
          INVALID_PORT_NUMBER.
     */
    PowerPortNumber portNumber;

    /**
     * Copy of sequenceId from the requestId.  The requestingDeviceId and requestSequenceId uniquely match the reply ↩
          code to the original request.
     */
    RequestSequence requestSequenceId;

    /**
     * Decision to accept or reason to reject the request.
     */
    ReplyStatus status;
}; // end struct Reply


/**
 * Change the level of a SRC or STOR device.
 * Topic Usage:
 * - TOPIC_ENERGY_START_STOP_REQUEST
 */
@topic
@extensibility(APPENDABLE)
struct EnergyStartStopRequest {

    /**
     * Identity of this request.
     */
    @key
    DeviceConfigRequest requestId;

    /**
     * Request sequence data used to associate a request and returning reply.
     */
    RequestSequence sequenceId;

    /**
     * Reject this request if fromLevel does not match presentLevel.  ESSL_ANY matches all levels.  ESSL_UNKNOWN is ↩
          invalid.
     */
    @try_construct(USE_DEFAULT)
    EnergyStartStopLevel fromLevel;

    /**
     * Desired operating level.  ESSL_ANY and ESSL_UNKNOWN are invalid.
     */
    @try_construct(USE_DEFAULT)
    EnergyStartStopLevel toLevel;

    /**
     * Conditions that may delay switch closure.  Only applies to ESSL_OPERATIONAL requests and power ports that ↩
          support PSF_SWITCH_CONDITIONS.
     */
    @optional
    PowerSwitchCloseConditions switchConditions;
}; // end struct EnergyStartStopRequest


/**
 * Report the operating level of a SRC and STOR device, including the progress of any transitions.
 * Topic Usage:
 * - TOPIC_ENERGY_START_STOP_STATE
 */
@topic
@extensibility(APPENDABLE)
struct EnergyStartStopState {

    /**
     * The device described by this structure.
     */
    @key
    Identity deviceId;

    /**
     * Present operating level.
     */
    @try_construct(USE_DEFAULT)
    EnergyStartStopLevel presentLevel;

    /**
     * Level that the device is transitioning to, or presentLevel if no transition.
     */
    @try_construct(USE_DEFAULT)
    EnergyStartStopLevel futureLevel;

    /**
```

```
     * This device or power switch is known to be locked and requests will get REPLY_PRECONDITION_FAILED.  A device-level lock must be reported on all affected ports.  Unknown locks may REPLY_OK and then PSTC_MALFUNCTION.
     */
    boolean requestLock;

    /**
     * Elapsed time the device has been in the present transition.  Remains at the total elapsed time when the transition is complete.
     * units: second (s)
     */
    @unit("s")
    float elapsedTime;

    /**
     * Estimates when the present transition will complete. It is zero (0.0) when the device is not in transition.
     * units: second (s)
     */
    @unit("s")
    float remainingTime;

    /**
     * Type of action that caused the device to be in its presentLevel.
     */
    @try_construct(USE_DEFAULT)
    PowerSwitchReason presentLevelReason;

    /**
     * Device or user that caused the device to be in its presentLevel.  Value may be a MC (REQUEST), this device (AUTO), or operator (MANUAL).  Present when the identity is known, otherwise absent.
     */
    @optional
    Identity presentLevelActor;

    /**
     * Progress towards completing a transition, or final result of a requested transition.
     */
    @try_construct(USE_DEFAULT)
    PowerSwitchTransitionCode futureLevelReason;
}; // end struct EnergyStartStopState



/**
 * Identify which :MC currently has write access to the active configuration.
 * Topic Usage:
 * - TOPIC_ACTIVE_MICROGRID_CONTROLLER_STATE
 */
@topic
@extensibility(APPENDABLE)
struct ActiveMicrogridControllerState {

    /**
     * The device described by this structure.
     */
    @key
    Identity deviceId;

    /**
     * The MC that currently controls this device. Present if a MC is selected, absent otherwise.
     */
    @optional
    Identity masterId;
}; // end struct ActiveMicrogridControllerState

module ac {

   const uint32 PhaseAngleSequence_MINLEN = 0;
   const uint32 PhaseAngleSequence_MAXLEN = 3;
   /**
    * A sequence of floats representing phase angles.
    * Minimum length: 0
    * Maximum length: 3
    */
   typedef sequence<float, PhaseAngleSequence_MAXLEN> PhaseAngleSequence;



   /**
    * Measurements for a single AC power line.
    */
   @nested
   @extensibility(APPENDABLE)
   struct PowerLineMeasurement {

       /**
        * RMS voltage.
        * units: volt (V)
        */
       @unit("V")
       float voltage;

       /**
```

```
     * Dominant line frequency.
     * units: hertz (Hz)
     * Change History:
     *  1) Version: 2.0.1, Change: modify, Interoperability: No-xtypes
     *     Removed minimum range specification.
     */
    @unit("Hz")
    float frequency;

    /**
     * RMS amperage.
     * units: ampere (A)
     */
    @unit("A")
    float amperage;

    /**
     * The angle (phase offset) by which the current lags the voltage.
     * units: radian (rad)
     * Change History:
     *  1) Version: 2.0.1, Change: modify, Interoperability: No-xtypes
     *     Made the field optional.
     */
    @range(min=-RADIAN_MAX, max=RADIAN_MAX)
    @unit("rad")
    @optional
    float phaseOffset;

    /**
     * Active RMS power.
     * units: watt (W)
     */
    @unit("W")
    float realPower;

    /**
     * Reactive RMS power.
     * units: volt ampere reactive (var)
     */
    @unit("var")
    float reactivePower;
}; // end struct PowerLineMeasurement


const uint32 PowerLineMeasurementSequence_MINLEN = 1;
const uint32 PowerLineMeasurementSequence_MAXLEN = 4;
/**
 * A sequence of PowerLineMeasurement.  Supports the AC circuits defined in PowerConnectorPhases.
 * Minimum length: 1
 * Maximum length: 4
 */
typedef sequence<PowerLineMeasurement, PowerLineMeasurementSequence_MAXLEN> PowerLineMeasurementSequence;



/**
 * Power measurement for an alternating current port.
 */
@nested
@extensibility(APPENDABLE)
struct PowerPortMeasurement {

    /**
     * Port being measured.
     */
    tms::PowerPortNumber portNumber;

    /**
     * Measured values for each AC power line.  Length and order of values are based on PowerPortInfo.wiring. ↩
          WIRING_SINGLE=1, WIRING_SPLIT=2, WIRING_3WYE=4, and WIRING_3DELTA=3.
     */
    @try_construct(TRIM)
    PowerLineMeasurementSequence line;

    /**
     * Measured phase difference between the voltages for all applicable wiring pairs.  Length and order of values↩
           are based on PowerPortInfo.wiring.
     * units: radian (rad)
     */
    @unit("rad")
    @try_construct(TRIM)
    PhaseAngleSequence phaseAngle;

    /**
     * Time derivative of realPower, summed across all wires excluding neutral.
     * units: watt per second (W/s)
     */
    @unit("W/s")
    float realPowerRateOfChange;

    /**
     * Time derivative of reactivePower, summed across all wires excluding neutral.
```

```
     * units: volt ampere reactive per second (var/s)
     */
    @unit("var/s")
    float reactivePowerRateOfChange;
}; // end struct PowerPortMeasurement


const uint32 PowerPortMeasurementSequence_MINLEN = 0;
const uint32 PowerPortMeasurementSequence_MAXLEN = tms::MAX_PORTS;
/**
 * A sequence of ac::PowerPortMeasurement.
 * Minimum length: 0
 * Maximum length: tms::MAX_PORTS
 */
typedef sequence<PowerPortMeasurement, PowerPortMeasurementSequence_MAXLEN> PowerPortMeasurementSequence;



/**
 * Publish measurements for all alternating current (AC) power ports on a device.
 * Topic Usage:
 * - TOPIC_AC_MEASUREMENT_UPDATE
 * - TOPIC_AC_SUMMARY_MEASUREMENT_UPDATE
 */
@topic
@extensibility(APPENDABLE)
struct MeasurementUpdate {

    /**
     * The device described by this structure.
     */
    @key
    tms::Identity deviceId;

    /**
     * Time of these measurements.
     */
    tms::ClockMonotonic timeMeasured;

    /**
     * Measurements for all power ports where PowerPortInfo.hasExternalMeter is true.
     */
    @try_construct(TRIM)
    PowerPortMeasurementSequence externalMeasurement;

    /**
     * Measurements for all power ports where PowerPortInfo.hasInternalMeter is true.
     */
    @try_construct(TRIM)
    PowerPortMeasurementSequence internalMeasurement;
}; // end struct MeasurementUpdate



/**
 * Load sharing parameters for AC power ports.
 */
@nested
@extensibility(APPENDABLE)
struct LoadSharingParameters {

    /**
     * The power port number.
     */
    tms::PowerPortNumber portNumber;

    /**
     * Desired frequency as a function of output real power.  Zero points disables frequency regulation and
          enables bounded or constant real power regulation.  One point is invalid.  Two points with the same
          frequency enables constant frequency (isochronous) regulation.  Two or more points with different
          frequencies enables droop frequency regulation.
     * units: watt, hertz (W, Hz)
     */
    @unit("W, Hz")
    tms::ControlCurve referenceFrequency;

    /**
     * Desired voltage as a function of output reactive power.  Zero points disables voltage regulation and
          enables bounded or constant reactive power regulation.  One point is invalid.  Two points with the same
          voltage enables constant voltage regulation.  Two or more points with different voltages enables droop
          voltage regulation.
     * units: volt ampere reactive, volt (var, V)
     */
    @unit("var, V")
    tms::ControlCurve referenceVoltage;

    /**
     * Minimum desired real power output.  May limit the device's ability to regulate frequency.
     * units: watt (W)
     */
    @unit("W")
    float minRealPower;
```

```
    /**
     * Maximum desired real power output.  May limit the device's ability to regulate frequency.
     * units: watt (W)
     */
    @unit("W")
    float maxRealPower;

    /**
     * Minimum desired voltage output.  May limit the device's ability to regulate real power.
     * units: hertz (Hz)
     */
    @unit("Hz")
    float minFrequency;

    /**
     * Maximum desired voltage output.  May limit the device's ability to regulate real power.
     * units: hertz (Hz)
     */
    @unit("Hz")
    float maxFrequency;

    /**
     * Minimum desired reactive power output.   May limit the device's ability to regulate voltage.
     * units: volt ampere reactive (var)
     */
    @unit("var")
    float minReactivePower;

    /**
     * Maximum desired reactive power output.  May limit the device's ability to regulate voltage.
     * units: volt ampere reactive (var)
     */
    @unit("var")
    float maxReactivePower;

    /**
     * Minimum desired voltage output.  May limit the device's ability to regulate reactive power.
     * units: volt (V)
     */
    @unit("V")
    float minVoltage;

    /**
     * Maximum desired voltage output.  May limit the device's ability to regulate reactive power.
     * units: volt (V)
     */
    @unit("V")
    float maxVoltage;
}; // end struct LoadSharingParameters


const uint32 LoadSharingParameterSequence_MINLEN = 1;
const uint32 LoadSharingParameterSequence_MAXLEN = tms::MAX_PORTS;
/**
 * A sequence of AC LoadSharingParameters.
 * Minimum length: 1
 * Maximum length: tms::MAX_PORTS
 */
typedef sequence<LoadSharingParameters, LoadSharingParameterSequence_MAXLEN> LoadSharingParameterSequence;



/**
 * Sets load sharing parameters for AC power ports.
 * Topic Usage:
 * - TOPIC_AC_LOAD_SHARING_REQUEST
 */
@topic
@extensibility(APPENDABLE)
struct LoadSharingRequest {

    /**
     * Identity of this request.
     */
    @key
    tms::PowerPortConfigRequest requestId;

    /**
     * Request sequence data used to associate a request and returning reply.
     */
    tms::RequestSequence sequenceId;

    /**
     * Desired parameters for controlling frequency, voltage, and power.
     */
    LoadSharingParameters parameters;
}; // end struct LoadSharingRequest



/**
```

```
    * Report the present value of the load sharing parameters for alternating current devices.
    * Topic Usage:
    * - TOPIC_AC_LOAD_SHARING_STATE
    */
   @topic
   @extensibility(APPENDABLE)
   struct LoadSharingState {

       /**
        * The device described by this structure.
        */
       @key
       tms::Identity deviceId;

       /**
        * Configuration that these values belong to.
        */
       tms::ConfigId config;

       /**
        * LoadSharingParameters for each power port that supports AC load sharing.
        */
       @try_construct(TRIM)
       LoadSharingParameterSequence parameters;
   }; // end struct LoadSharingState

}; //end module ac

module dc {


   /**
    * Measurements for a single DC power line.
    */
   @nested
   @extensibility(APPENDABLE)
   struct PowerLineMeasurement {

       /**
        * Averaged DC voltage.
        * units: volt (V)
        */
       @unit("V")
       float voltage;

       /**
        * Averaged DC amperage.
        * units: ampere (A)
        */
       @unit("A")
       float amperage;

       /**
        * Averaged DC power.
        * units: watt (W)
        */
       @unit("W")
       float power;
   }; // end struct PowerLineMeasurement


   const uint32 PowerLineMeasurementSequence_MINLEN = 1;
   const uint32 PowerLineMeasurementSequence_MAXLEN = 3;
   /**
    * A sequence of PowerLineMeasurement.  Supports the DC circuits defined in PowerConnectorPhases.
    * Minimum length: 1
    * Maximum length: 3
    */
   typedef sequence<PowerLineMeasurement, PowerLineMeasurementSequence_MAXLEN> PowerLineMeasurementSequence;



   /**
    * Power measurement for a direct current power port.
    */
   @nested
   @extensibility(APPENDABLE)
   struct PowerPortMeasurement {

       /**
        * Port being measured.
        */
       tms::PowerPortNumber portNumber;

       /**
        * Measured values for each DC line.  Length and order of values are based on PowerPortInfo.wiring.  PHASE_DC↩
            =2, PHASE_DC_3WIRE=3
        */
       @try_construct(TRIM)
       PowerLineMeasurementSequence line;

       /**
```

```
     * Time derivative of power, summed across all wires excluding neutral.
     * units: watt per second (W/s)
     */
    @unit("W/s")
    float powerRateOfChange;
}; // end struct PowerPortMeasurement


const uint32 PowerPortMeasurementSequence_MINLEN = 0;
const uint32 PowerPortMeasurementSequence_MAXLEN = tms::MAX_PORTS;
/**
 * A sequence of dc:PowerPortMeasurement.
 * Minimum length: 0
 * Maximum length: tms::MAX_PORTS
 */
typedef sequence<PowerPortMeasurement, PowerPortMeasurementSequence_MAXLEN> PowerPortMeasurementSequence;



/**
 * Publish measurements for all direct current (DC) power ports on a device.
 * Topic Usage:
 * - TOPIC_DC_MEASUREMENT_UPDATE
 * - TOPIC_DC_SUMMARY_MEASUREMENT_UPDATE
 */
@topic
@extensibility(APPENDABLE)
struct MeasurementUpdate {

    /**
     * The device described by this structure.
     */
    @key
    tms::Identity deviceId;

    /**
     * Time of these measurements.
     */
    tms::ClockMonotonic timeMeasured;

    /**
     * Measurements for all power ports where PowerPortInfo.hasExternalMeter is true.
     */
    @try_construct(TRIM)
    PowerPortMeasurementSequence externalMeasurement;

    /**
     * Measurements for all power ports where PowerPortInfo.hasInternalMeter is true.
     */
    @try_construct(TRIM)
    PowerPortMeasurementSequence internalMeasurement;
}; // end struct MeasurementUpdate



/**
 * Load sharing parameters for DC power ports.
 */
@nested
@extensibility(APPENDABLE)
struct LoadSharingParameters {

    /**
     * The power port number.
     */
    tms::PowerPortNumber portNumber;

    /**
     * Desired voltage as a function of output amperage.  Zero points disables voltage regulation and enables
          bounded or constant amperage regulation.  One point is invalid.  Two points with the same voltage
          enables constant voltage regulation.  Two or more points with different voltages enables droop voltage
          regulation.
     * units: ampere, volt (A, V)
     */
    @unit("A,␣V")
    tms::ControlCurve referenceVoltage;

    /**
     * Minimum desired amperage output.  May limit the device's ability to regulate voltage.
     * units: ampere (A)
     */
    @unit("A")
    float minAmperage;

    /**
     * Maximum desired amperage output.  May limit the device's ability to regulate voltage.
     * units: ampere (A)
     */
    @unit("A")
    float maxAmperage;

    /**
```

```
         * Minimum desired voltage output.  May limit the device's ability to regulate amperage.
         * units: volt (V)
         */
        @unit("V")
        float minVoltage;

        /**
         * Maximum desired voltage output.  May limit the device's ability to regulate amperage.
         * units: volt (V)
         */
        @unit("V")
        float maxVoltage;
    }; // end struct LoadSharingParameters


    const uint32 LoadSharingParameterSequence_MINLEN = 1;
    const uint32 LoadSharingParameterSequence_MAXLEN = tms::MAX_PORTS;
    /**
     * A sequence of DC LoadSharingParameters.
     * Minimum length: 1
     * Maximum length: tms::MAX_PORTS
     */
    typedef sequence<LoadSharingParameters, LoadSharingParameterSequence_MAXLEN> LoadSharingParameterSequence;



    /**
     * Sets load sharing parameters for DC power ports.
     * Topic Usage:
     * - TOPIC_DC_LOAD_SHARING_REQUEST
     */
    @topic
    @extensibility(APPENDABLE)
    struct LoadSharingRequest {

        /**
         * Identity of this request.
         */
        @key
        tms::PowerPortConfigRequest requestId;

        /**
         * Request sequence data used to associate a request and returning reply.
         */
        tms::RequestSequence sequenceId;

        /**
         * Desired parameters for controlling voltage and power.
         */
        LoadSharingParameters parameters;
    }; // end struct LoadSharingRequest



    /**
     * Report the present value of the load sharing parameters for direct current devices.
     * Topic Usage:
     * - TOPIC_DC_LOAD_SHARING_STATE
     */
    @topic
    @extensibility(APPENDABLE)
    struct LoadSharingState {

        /**
         * The device described by this structure.
         */
        @key
        tms::Identity deviceId;

        /**
         * Configuration that these values belong to.
         */
        tms::ConfigId config;

        /**
         * LoadSharingParameters for each power port that support DC load sharing.
         */
        @try_construct(TRIM)
        LoadSharingParameterSequence parameters;
    }; // end struct LoadSharingState

}; //end module dc


/**
 * Report the state of a storage device.
 * Topic Usage:
 * - TOPIC_STORAGE_UPDATE
 */
@topic
@extensibility(APPENDABLE)
struct StorageUpdate {
```

```
/**
 * The device described by this structure.
 */
@key
Identity deviceId;

/**
 * Measurement indicating the DC bus voltage between the energy storage and power converter.
 * units: volt (V)
 */
@unit("V")
float internalVoltage;

/**
 * Estimate of present charge as a fraction of rated capacity.  0=empty and 1=full.
 * Change History:
 *  1) Version: 2.0.1, Change: modify, Interoperability: No-xtypes
 *     Made the field optional.
 */
@range(min=0, max=1)
@optional
float stateOfCharge;

/**
 * Estimate of present energy available for discharging to the microgrid.
 * units: joule (J)
 */
@unit("J")
float availableEnergy;

/**
 * Estimate of how long the energy storage can operate at the present power rate.
 * units: second (s)
 */
@unit("s")
float holdTime;

/**
 * Estimate of how long the energy storage can charge at different power levels, without exceeding the ↩
      highStateOfCharge.
 * units: watt, second (W, s)
 */
@unit("W,␣s")
Curve2D chargeTime;

/**
 * Estimate of how long the energy storage can discharge at different power levels, without exceeding the ↩
      lowStateOfCharge.
 * units: watt, second (W, s)
 */
@unit("W,␣s")
Curve2D dischargeTime;

/**
 * Estimate of how long the energy storage can charge at different power levels, with battle short enabled.  Must↩
       be greater than or equal to the chargeTime.
 * units: watt, second (W, s)
 */
@unit("W,␣s")
Curve2D maxChargeTime;

/**
 * Estimate of how long the energy storage can discharge at different power levels, with battle short enabled.  ↩
      Must be greater than or equal to the dischargeTime.
 * units: watt, second (W, s)
 */
@unit("W,␣s")
Curve2D maxDischargeTime;

/**
 * The maximum rate of charge based on current temperatures and state of charge. 0 to Inf.
 * units: watt (W)
 * Change History:
 *  1) Version: 2.0.1, Change: modify, Interoperability: No-xtypes
 *     Made the field optional.
 */
@min(0)
@unit("W")
@optional
float maxChargeRate;

/**
 * The maximum rate of discharge based on current temperatures and state of charge. -Inf to 0.
 * units: watt (W)
 * Change History:
 *  1) Version: 2.0.1, Change: modify, Interoperability: No-xtypes
 *     Made the field optional.
 */
@max(0)
@unit("W")
@optional
```

```
    float maxDischargeRate;
}; // end struct StorageUpdate


/**
 * Design information for control services on microgrid controllers and dashboards.
 */
@nested
@extensibility(APPENDABLE)
struct ControlServiceInfo {

    /**
     * Microgrid Controller information.
     */
    @optional
    MicrogridControllerInfo mc;

    /**
     * Microgrid Dashboard information.
     */
    @optional
    MicrogridDashboardInfo md;
}; // end struct ControlServiceInfo


/**
 * Design information for a power device which is any TMS device role that generates, distributes or consumes power. ↩
      SRC, STOR, LOAD, DIST and CONV.
 */
@nested
@extensibility(APPENDABLE)
struct PowerDeviceInfo {

    /**
     * 1 for SRC and STOR and LOAD, 2+ for DIST and CONV
     */
    @try_construct(TRIM)
    PowerPortInfoSequence powerPorts;

    /**
     * Grounding port information.
     */
    @try_construct(TRIM)
    GroundingInfoSequence grounds;

    /**
     * Conversion information. Required for the CONV device role and empty otherwise.
     */
    @optional
    ConversionInfo conversion;

    /**
     * Distribution device information. Required for the DIST device role and empty otherwise.
     */
    @optional
    DistributionInfo distribution;

    /**
     * Power source device information. Required for the SRC device role and empty otherwise.
     */
    @optional
    SourceInfo source;

    /**
     * Power storage device information. Required for the STOR device role and empty otherwise.
     */
    @optional
    StorageInfo storage;

    /**
     * Power load information. Required for the LOAD device role and empty otherwise.
     */
    @optional
    LoadInfo load;
}; // end struct PowerDeviceInfo


/**
 * Design information for the device.
 * Topic Usage:
 * - TOPIC_DEVICE_INFO
 */
@topic
@extensibility(APPENDABLE)
struct DeviceInfo {

    /**
     * The device described by this structure.
     */
```

```
        @key
        Identity deviceId;

        /**
         * TMS role implemented by this device.
         */
        @try_construct(USE_DEFAULT)
        DeviceRole role;

        /**
         * Product information.
         */
        ProductInfo product;

        /**
         * communications interfaces implemented by this device
         */
        TopicInfo topics;

        /**
         * Information about the device controller.  May apply to both control services and power devices.
         */
        @optional
        ControlHardwareInfo controlHardware;

        /**
         * Information about the power hardware, if applicable.
         */
        @optional
        PowerHardwareInfo powerHardware;

        /**
         * A complete list of read-write control parameters that must be present if the device implements the ↩
              ControlParameterState topic.
         */
        @optional
        @try_construct(TRIM)
        ParameterMetadataSequence controlParameters;

        /**
         * A complete list of read-only metric parameters that must be present if the device implements the ↩
              MetricParameterState topic.
         */
        @optional
        @try_construct(TRIM)
        ParameterMetadataSequence metricParameters;

        /**
         * Control role-specific information. Required for MC or MD device roles and empty otherwise.
         */
        @optional
        ControlServiceInfo controlService;

        /**
         * Power device role-specific information. Required for CONV, STOR, DIST, SRC, or LOAD device roles and empty ↩
              otherwise.
         */
        @optional
        PowerDeviceInfo powerDevice;
    }; // end struct DeviceInfo

}; //end module tms
```

E.2 Quality of Service XML

The OMG DDS QoS Library XML file for TMS is attached to this PDF as . This file is non-normative. Only standard OMG DDS QoS policies with non-default values are included. The following is a syntax-highlighted, verbatim copy of the file contents. Automatic line wraps are indicated by a ↩.

```
<?xml version="1.0" encoding="UTF-8"?>
<!-- TMS_VERSION 2.0.3-prerelease2 -->
<dds xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
        xmlns="http://www.omg.org/spec/DDS-XML"
        xsi:schemaLocation="http://www.omg.org/spec/DDS-XML␣./schema/dds-xml_dds_system_definitions.xsd">
        <qos_library name="TMS">
                <qos_profile name="PublishLast">
                        <datareader_qos>
                                <deadline>
                                        <period>
                                                <sec>DURATION_INFINITE_SEC</sec>
                                                <nanosec>DURATION_INFINITE_NSEC</nanosec>
                                        </period>
```

```
                </deadline>
                <durability>
                        <kind>TRANSIENT_LOCAL_DURABILITY_QOS</kind>
                </durability>
                <history>
                        <kind>KEEP_LAST_HISTORY_QOS</kind>
                        <depth>1</depth>
                </history>
                <liveliness>
                        <kind>AUTOMATIC_LIVELINESS_QOS</kind>
                        <lease_duration>
                                <sec>DURATION_INFINITE_SEC</sec>
                                <nanosec>DURATION_INFINITE_NSEC</nanosec>
                        </lease_duration>
                </liveliness>
                <reliability>
                        <kind>RELIABLE_RELIABILITY_QOS</kind>
                </reliability>
                <ownership>
                        <kind>EXCLUSIVE_OWNERSHIP_QOS</kind>
                </ownership>
        </datareader_qos>
        <datawriter_qos>
                <deadline>
                        <period>
                                <sec>DURATION_INFINITE_SEC</sec>
                                <nanosec>DURATION_INFINITE_NSEC</nanosec>
                        </period>
                </deadline>
                <durability>
                        <kind>TRANSIENT_LOCAL_DURABILITY_QOS</kind>
                </durability>
                <history>
                        <kind>KEEP_LAST_HISTORY_QOS</kind>
                        <depth>1</depth>
                </history>
                <liveliness>
                        <kind>AUTOMATIC_LIVELINESS_QOS</kind>
                        <lease_duration>
                                <sec>DURATION_INFINITE_SEC</sec>
                                <nanosec>DURATION_INFINITE_NSEC</nanosec>
                        </lease_duration>
                </liveliness>
                <reliability>
                        <kind>RELIABLE_RELIABILITY_QOS</kind>
                </reliability>
                <ownership>
                        <kind>EXCLUSIVE_OWNERSHIP_QOS</kind>
                </ownership>
                <ownership_strength>
                        <value>1</value> <!-- Note:  This value must increase each time the device ↩
                             restarts.  -->
                </ownership_strength>
        </datawriter_qos>
</qos_profile>

<qos_profile name="Rare">
        <datareader_qos>
                <durability>
                        <kind>VOLATILE_DURABILITY_QOS</kind>
                </durability>
                <history>
                        <kind>KEEP_LAST_HISTORY_QOS</kind>
                        <depth>1</depth>
                </history>
                <liveliness>
                        <kind>AUTOMATIC_LIVELINESS_QOS</kind>
                        <lease_duration>
                                <sec>DURATION_INFINITE_SEC</sec>
                                <nanosec>DURATION_INFINITE_NSEC</nanosec>
                        </lease_duration>
                </liveliness>
                <reliability>
                        <kind>BEST_EFFORT_RELIABILITY_QOS</kind>
                </reliability>
                <ownership>
                        <kind>EXCLUSIVE_OWNERSHIP_QOS</kind>
                </ownership>
                <deadline>
                        <period>
                                <sec>2000</sec>
                                <nanosec>0</nanosec>
                        </period>
                </deadline>
        </datareader_qos>
        <datawriter_qos>
                <durability>
                        <kind>VOLATILE_DURABILITY_QOS</kind>
                </durability>
                <history>
                        <kind>KEEP_LAST_HISTORY_QOS</kind>
                        <depth>1</depth>
```

```
                </history>
                <liveliness>
                        <kind>AUTOMATIC_LIVELINESS_QOS</kind>
                        <lease_duration>
                                <sec>DURATION_INFINITE_SEC</sec>
                                <nanosec>DURATION_INFINITE_NSEC</nanosec>
                        </lease_duration>
                </liveliness>
                <reliability>
                        <kind>BEST_EFFORT_RELIABILITY_QOS</kind>
                </reliability>
                <ownership>
                        <kind>EXCLUSIVE_OWNERSHIP_QOS</kind>
                </ownership>
                <ownership_strength>
                        <value>1</value> <!-- Note:  This value must increase each time the device ↩
                             restarts.  -->
                </ownership_strength>
                <deadline>
                        <period>
                                <sec>2000</sec>
                                <nanosec>0</nanosec>
                        </period>
                </deadline>
        </datawriter_qos>
</qos_profile>

<!--  -->
<qos_profile name="Slow">
        <datareader_qos>
                <durability>
                        <kind>VOLATILE_DURABILITY_QOS</kind>
                </durability>
                <history>
                        <kind>KEEP_LAST_HISTORY_QOS</kind>
                        <depth>1</depth>
                </history>
                <liveliness>
                        <kind>AUTOMATIC_LIVELINESS_QOS</kind>
                        <lease_duration>
                                <sec>DURATION_INFINITE_SEC</sec>
                                <nanosec>DURATION_INFINITE_NSEC</nanosec>
                        </lease_duration>
                </liveliness>
                <reliability>
                        <kind>BEST_EFFORT_RELIABILITY_QOS</kind>
                </reliability>
                <ownership>
                        <kind>EXCLUSIVE_OWNERSHIP_QOS</kind>
                </ownership>
                <deadline>
                        <period>
                                <sec>20</sec>
                                <nanosec>0</nanosec>
                        </period>
                </deadline>
        </datareader_qos>
        <datawriter_qos>
                <durability>
                        <kind>VOLATILE_DURABILITY_QOS</kind>
                </durability>
                <history>
                        <kind>KEEP_LAST_HISTORY_QOS</kind>
                        <depth>1</depth>
                </history>
                <liveliness>
                        <kind>AUTOMATIC_LIVELINESS_QOS</kind>
                        <lease_duration>
                                <sec>DURATION_INFINITE_SEC</sec>
                                <nanosec>DURATION_INFINITE_NSEC</nanosec>
                        </lease_duration>
                </liveliness>
                <reliability>
                        <kind>BEST_EFFORT_RELIABILITY_QOS</kind>
                </reliability>
                <ownership>
                        <kind>EXCLUSIVE_OWNERSHIP_QOS</kind>
                </ownership>
                <ownership_strength>
                        <value>1</value> <!-- Note:  This value must increase each time the device ↩
                             restarts.  -->
                </ownership_strength>
                <deadline>
                        <period>
                                <sec>20</sec>
                                <nanosec>0</nanosec>
                        </period>
                </deadline>
        </datawriter_qos>
</qos_profile>

<!--  -->
```

```
<qos_profile name="Medium">
        <datareader_qos>
                <durability>
                        <kind>VOLATILE_DURABILITY_QOS</kind>
                </durability>
                <history>
                        <kind>KEEP_LAST_HISTORY_QOS</kind>
                        <depth>1</depth>
                </history>
                <liveliness>
                        <kind>AUTOMATIC_LIVELINESS_QOS</kind>
                        <lease_duration>
                                <sec>DURATION_INFINITE_SEC</sec>
                                <nanosec>DURATION_INFINITE_NSEC</nanosec>
                        </lease_duration>
                </liveliness>
                <reliability>
                        <kind>BEST_EFFORT_RELIABILITY_QOS</kind>
                </reliability>
                <ownership>
                        <kind>EXCLUSIVE_OWNERSHIP_QOS</kind>
                </ownership>
                <deadline>
                        <period>
                                <sec>3</sec>
                                <nanosec>0</nanosec>
                        </period>
                </deadline>
        </datareader_qos>
        <datawriter_qos>
                <durability>
                        <kind>VOLATILE_DURABILITY_QOS</kind>
                </durability>
                <history>
                        <kind>KEEP_LAST_HISTORY_QOS</kind>
                        <depth>1</depth>
                </history>
                <liveliness>
                        <kind>AUTOMATIC_LIVELINESS_QOS</kind>
                        <lease_duration>
                                <sec>DURATION_INFINITE_SEC</sec>
                                <nanosec>DURATION_INFINITE_NSEC</nanosec>
                        </lease_duration>
                </liveliness>
                <reliability>
                        <kind>BEST_EFFORT_RELIABILITY_QOS</kind>
                </reliability>
                <ownership>
                        <kind>EXCLUSIVE_OWNERSHIP_QOS</kind>
                </ownership>
                <ownership_strength>
                        <value>1</value> <!-- Note:  This value must increase each time the device ↩
                             restarts.  -->
                </ownership_strength>
                <deadline>
                        <period>
                                <sec>3</sec>
                                <nanosec>0</nanosec>
                        </period>
                </deadline>
        </datawriter_qos>
</qos_profile>

<!--  -->
<qos_profile name="Continuous">
        <datareader_qos>
                <durability>
                        <kind>VOLATILE_DURABILITY_QOS</kind>
                </durability>
                <history>
                        <kind>KEEP_LAST_HISTORY_QOS</kind>
                        <depth>1</depth>
                </history>
                <liveliness>
                        <kind>AUTOMATIC_LIVELINESS_QOS</kind>
                        <lease_duration>
                                <sec>DURATION_INFINITE_SEC</sec>
                                <nanosec>DURATION_INFINITE_NSEC</nanosec>
                        </lease_duration>
                </liveliness>
                <reliability>
                        <kind>BEST_EFFORT_RELIABILITY_QOS</kind>
                </reliability>
                <ownership>
                        <kind>EXCLUSIVE_OWNERSHIP_QOS</kind>
                </ownership>
                <deadline>
                        <period>
                                <sec>2</sec>
                                <nanosec>0</nanosec>
                        </period>
                </deadline>
```

```
        </datareader_qos>
        <datawriter_qos>
                <durability>
                        <kind>VOLATILE_DURABILITY_QOS</kind>
                </durability>
                <history>
                        <kind>KEEP_LAST_HISTORY_QOS</kind>
                        <depth>1</depth>
                </history>
                <liveliness>
                        <kind>AUTOMATIC_LIVELINESS_QOS</kind>
                        <lease_duration>
                                <sec>DURATION_INFINITE_SEC</sec>
                                <nanosec>DURATION_INFINITE_NSEC</nanosec>
                        </lease_duration>
                </liveliness>
                <reliability>
                        <kind>BEST_EFFORT_RELIABILITY_QOS</kind>
                </reliability>
                <ownership>
                        <kind>EXCLUSIVE_OWNERSHIP_QOS</kind>
                </ownership>
                <ownership_strength>
                        <value>1</value> <!-- Note:  This value must increase each time the device ↩
                             restarts.  -->
                </ownership_strength>
                <deadline>
                        <period>
                                <sec>2</sec>
                                <nanosec>0</nanosec>
                        </period>
                </deadline>
        </datawriter_qos>
</qos_profile>

<!--  -->
<qos_profile name="Command">
        <datareader_qos>
                <deadline>
                        <period>
                                <sec>DURATION_INFINITE_SEC</sec>
                                <nanosec>DURATION_INFINITE_NSEC</nanosec>
                        </period>
                </deadline>
                <durability>
                        <kind>VOLATILE_DURABILITY_QOS</kind>
                </durability>
                <history>
                        <kind>KEEP_LAST_HISTORY_QOS</kind>
                        <depth>1</depth>
                </history>
                <liveliness>
                        <kind>AUTOMATIC_LIVELINESS_QOS</kind>
                        <lease_duration>
                                <sec>DURATION_INFINITE_SEC</sec>
                                <nanosec>DURATION_INFINITE_NSEC</nanosec>
                        </lease_duration>
                </liveliness>
                <reliability>
                        <kind>RELIABLE_RELIABILITY_QOS</kind>
                </reliability>
                <ownership>
                        <kind>EXCLUSIVE_OWNERSHIP_QOS</kind>
                </ownership>
        </datareader_qos>
        <datawriter_qos>
                <deadline>
                        <period>
                                <sec>DURATION_INFINITE_SEC</sec>
                                <nanosec>DURATION_INFINITE_NSEC</nanosec>
                        </period>
                </deadline>
                <durability>
                        <kind>VOLATILE_DURABILITY_QOS</kind>
                </durability>
                <history>
                        <kind>KEEP_LAST_HISTORY_QOS</kind>
                        <depth>1</depth>
                </history>
                <liveliness>
                        <kind>AUTOMATIC_LIVELINESS_QOS</kind>
                        <lease_duration>
                                <sec>DURATION_INFINITE_SEC</sec>
                                <nanosec>DURATION_INFINITE_NSEC</nanosec>
                        </lease_duration>
                </liveliness>
                <reliability>
                        <kind>RELIABLE_RELIABILITY_QOS</kind>
                </reliability>
                <ownership>
                        <kind>EXCLUSIVE_OWNERSHIP_QOS</kind>
                </ownership>
```

```
                <ownership_strength>
                        <value>1</value> <!-- Note:  This value must increase each time the device ↩
                             restarts.  -->
                </ownership_strength>
        </datawriter_qos>
</qos_profile>

<!--  -->
<qos_profile name="Response">
        <datareader_qos>
                <deadline>
                        <period>
                                <sec>DURATION_INFINITE_SEC</sec>
                                <nanosec>DURATION_INFINITE_NSEC</nanosec>
                        </period>
                </deadline>
                <durability>
                        <kind>VOLATILE_DURABILITY_QOS</kind>
                </durability>
                <history>
                        <kind>KEEP_LAST_HISTORY_QOS</kind>
                        <depth>1</depth>
                </history>
                <liveliness>
                        <kind>AUTOMATIC_LIVELINESS_QOS</kind>
                        <lease_duration>
                                <sec>DURATION_INFINITE_SEC</sec>
                                <nanosec>DURATION_INFINITE_NSEC</nanosec>
                        </lease_duration>
                </liveliness>
                <reliability>
                        <kind>RELIABLE_RELIABILITY_QOS</kind>
                </reliability>
                <ownership>
                        <kind>EXCLUSIVE_OWNERSHIP_QOS</kind>
                </ownership>
        </datareader_qos>
        <datawriter_qos>
                <deadline>
                        <period>
                                <sec>DURATION_INFINITE_SEC</sec>
                                <nanosec>DURATION_INFINITE_NSEC</nanosec>
                        </period>
                </deadline>
                <durability>
                        <kind>VOLATILE_DURABILITY_QOS</kind>
                </durability>
                <history>
                        <kind>KEEP_LAST_HISTORY_QOS</kind>
                        <depth>1</depth>
                </history>
                <liveliness>
                        <kind>AUTOMATIC_LIVELINESS_QOS</kind>
                        <lease_duration>
                                <sec>DURATION_INFINITE_SEC</sec>
                                <nanosec>DURATION_INFINITE_NSEC</nanosec>
                        </lease_duration>
                </liveliness>
                <reliability>
                        <kind>RELIABLE_RELIABILITY_QOS</kind>
                </reliability>
                <ownership>
                        <kind>EXCLUSIVE_OWNERSHIP_QOS</kind>
                </ownership>
                <ownership_strength>
                        <value>1</value> <!-- Note:  This value must increase each time the device ↩
                             restarts.  -->
                </ownership_strength>
        </datawriter_qos>
</qos_profile>

<!--  -->
<qos_profile name="Reply">
        <datareader_qos>
                <deadline>
                        <period>
                                <sec>DURATION_INFINITE_SEC</sec>
                                <nanosec>DURATION_INFINITE_NSEC</nanosec>
                        </period>
                </deadline>
                <durability>
                        <kind>VOLATILE_DURABILITY_QOS</kind>
                </durability>
                <history>
                        <kind>KEEP_LAST_HISTORY_QOS</kind>
                        <depth>128</depth>
                </history>
                <liveliness>
                        <kind>AUTOMATIC_LIVELINESS_QOS</kind>
                        <lease_duration>
                                <sec>DURATION_INFINITE_SEC</sec>
                                <nanosec>DURATION_INFINITE_NSEC</nanosec>
```

```
                        </lease_duration>
                    </liveliness>
                    <reliability>
                        <kind>RELIABLE_RELIABILITY_QOS</kind>
                    </reliability>
                    <ownership>
                        <kind>EXCLUSIVE_OWNERSHIP_QOS</kind>
                    </ownership>
                </datareader_qos>
                <datawriter_qos>
                    <deadline>
                        <period>
                            <sec>DURATION_INFINITE_SEC</sec>
                            <nanosec>DURATION_INFINITE_NSEC</nanosec>
                        </period>
                    </deadline>
                    <durability>
                        <kind>VOLATILE_DURABILITY_QOS</kind>
                    </durability>
                    <history>
                        <kind>KEEP_LAST_HISTORY_QOS</kind>
                        <depth>128</depth>
                    </history>
                    <liveliness>
                        <kind>AUTOMATIC_LIVELINESS_QOS</kind>
                        <lease_duration>
                            <sec>DURATION_INFINITE_SEC</sec>
                            <nanosec>DURATION_INFINITE_NSEC</nanosec>
                        </lease_duration>
                    </liveliness>
                    <reliability>
                        <kind>RELIABLE_RELIABILITY_QOS</kind>
                    </reliability>
                    <ownership>
                        <kind>EXCLUSIVE_OWNERSHIP_QOS</kind>
                    </ownership>
                    <ownership_strength>
                        <value>1</value> <!-- Note:  This value must increase each time the device ↩
                            restarts.  -->
                    </ownership_strength>
                </datawriter_qos>
            </qos_profile>
        </qos_library>
</dds>
```

E.3 Topics Table

A comma-separated values (CSV) file is attached to this PDF as `mil-std-3071_topics.csv`. This file is non-normative. The information in this file was used to generate Table B.1, Table B.2, and the topic definition tables in each section of Chapter B.

For convenience of the following explanation, tables from Section B.5 are copied into Figure E.1.

Table E.1 provides an overview of how each topic is used.

Table E.1: Description of the Device Announcement topics.

Topic	Description
DeviceInfo	Product information for the device. Used to discover TMS compliant devices.

Table E.2 specifies the publishers and subscribers for each topic as well as usage requirements as described in Section A.8.7.

Table E.2: Participants of the Device Announcement topics.

Topic	Usage	MD	MC	MON	SRC	STOR	DIST	LOAD	CONV
DeviceInfo	Required	Pub Sub	Pub Sub	Sub	Pub Sub	Pub Sub	Pub Sub	Pub Sub	Pub Sub

This participants table has the following interpretation.

1. DeviceInfo shall be published by MD, MC, SRC, STOR, DIST, LOAD, and CONV devices.
2. DeviceInfo may be subscribed by MD, MC, MON, SRC, STOR, DIST, LOAD, and CONV devices.

Table E.3 specifies the timing for when sample updates are sent to each topic. These parameters are defined in Section A.8.5.

Table E.3: Timing of the Device Announcement topics.

`Topic`	Data Trigger	Rate Class	Burst Size	QoS Profile
DeviceInfo	At Start	Once	1	PublishLast

Table E.4 specifies the data types that are used as sample values for each topic.

Table E.4: Data types of the Device Announcement topics.

Topic	`Data Type`
DeviceInfo	`tms::DeviceInfo`

Topic-level data types are defined in the order shown in Table E.4. Then nested data types are defined in the order of appearance. Cross-references are provided for data types that have already been defined.

Figure E.1: Example Topic Definition Tables

The columns in the attached topics table CSV file have the following interpretation.

1. PDF # – Section number where this information appears. Last column in Table B.1 and Table B.2.
2. PDF Heading – Section name where this information appears.
3. Topic – Name of the topic and first column in every table.
4. Description – Column in topic description tables such as Table E.1.
5. Usage – Column in Table B.2 and topic participants tables such as Table E.2.
6. Condition – Column in Table B.2 and item listed after topic participants tables such as Table E.2. (Note: DeviceInfo is not conditional and so this entry is empty in the table.)
7. MD – Column in Table B.1 and topic participants tables such as Table E.2.
8. MC – Column in Table B.1 and topic participants tables such as Table E.2.
9. MON – Column in Table B.1 and topic participants tables such as Table E.2.
10. SRC – Column in Table B.1 and topic participants tables such as Table E.2.
11. STOR – Column in Table B.1 and topic participants tables such as Table E.2.
12. DIST – Column in Table B.1 and topic participants tables such as Table E.2.
13. LOAD – Column in Table B.1 and topic participants tables such as Table E.2.
14. CONV – Column in Table B.1 and topic participants tables such as Table E.2.
15. Data Trigger – Column in topic timing tables such as Table E.3.
16. Rate Class – Column in topic timing tables such as Table E.3.
17. Burst Size – Column in topic timing tables such as Table E.3.
18. QoS Profile – Column in topic timing tables such as Table E.3.
19. Data Type – Column in data type tables such as Table E.4.

E.4 Data Flows Table

A comma-separated values (CSV) file is attached to this PDF as `mil-std-3071_data_flows.csv`. This file is non-normative. The information in this file was used to generate all data flow diagrams as described in Section A.8.3.

For convenience of the following explanation, Figure A.9 is copied into Figure E.2.

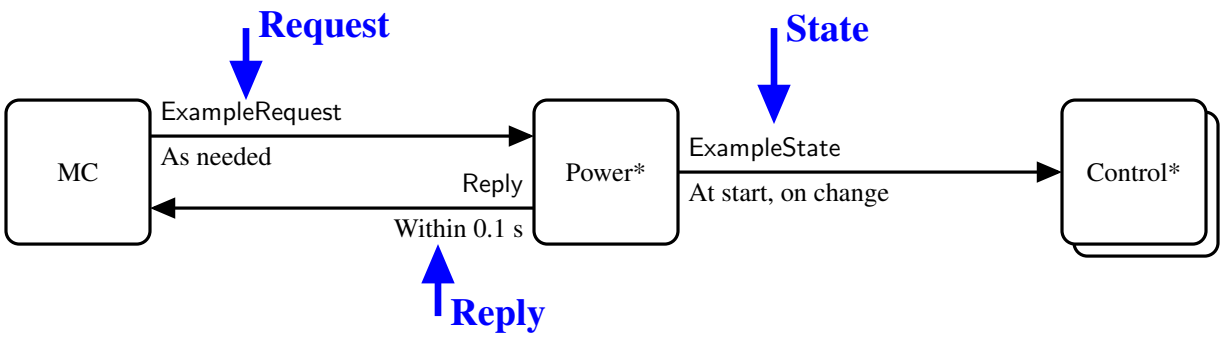


Figure E.2: Example Data Flow

The columns in the attached data flow table CSV file have the following interpretation.

1. PDF # – Section number where this information appears. Last column in Table B.1 and Table B.2.
2. PDF Heading – Section name where this information appears.
3. Requesting Device – Device role(s) in the left box, or blank if unused.
4. Operating Device – Device role(s) in the center box, or blank if unused.
5. Observing Device – Device role(s) in the right device box, or blank if unused.
6. Request Topic – Name of the topic from Requesting Device to Operating Device.
7. Reply Topic – Name of the topic from Operating Device back to Requesting Device.
8. Status Topic – Name of the topic from Operating Device to Observing Device.
9. Result Topic – Name of an additional topic, only used in custom patterns. See Section A.8.3.4.

Change History

This chapter lists all technical changes to this appendix. Editorial changes are not listed. See Section A.12 for further explanation.

Versions are listed from newest to oldest

Version 2.0.3

add

- DDS Security, 342
- Machine Readable Files
  - Data Flows CSV, 418
  - Topics CSV, 415
- Platform representative, 54
- SuspectParameterNumber
  - =SPN_RANGE_J1939, 219
  - =SPN_RANGE_PROPRIETARY, 219
  - =SPN_RANGE_TMS, 220
- tms::DiagnosticTroubleCode
  - .affectedIdentityId, 218
- tms::PowerConnectorType
  - =CONNECTOR_VOLTARIUS, 157
- tms::ProductInfo
  - .platformRepresentative, 132

modify

- AcSummaryMeasurementUpdate without external meter, 259
- Data Accuracy and Completeness, 87
- LOAD may subscribe to PowerSwitchRequest, 250
- Network Interoperability, 321
- tms::DeviceIcon
  - .deviceId, 184
- Updated DDS standards, 321

Version 2.0.2

modify

- OMG DDS QoS OWNERSHIP_STRENGTH, 326

tms::PowerPortInfo
.maxFrequency, 152
.minFrequency, 152
Type namespace separator, 78

Version 2.0.1

add

Microgrid controller selection priorityRanking, 60
tms::MicrogridControllerInfo
.priorityRanking, 148
tms::ReplyCode
=REPLY_PERMISSION_DENIED, 123

modify

tms::ac::PowerLineMeasurement
.frequency, 263
.phaseOffset, 263
tms::ControlHardwareInfo
.maxTemperature, 134
.minTemperature, 134
tms::ControlHardwareUpdate
.memoryUsage, 187
.networkUsage, 187
.processorUsage, 187
.storageUsage, 188
.temperature, 188
tms::EnergyStorageInfo
.highStateOfCharge, 139
.lowStateOfCharge, 139
.maxTemperature, 140
.minTemperature, 139
.nomTemperature, 140
tms::EnergyStorageState
.averageTemperature, 195
.maxTemperature, 195
.minTemperature, 194
tms::EngineInfo
.maxEngineSpeed, 137
.maxOilPressure, 136
.minEngineSpeed, 136
.minOilPressure, 136
.minWetStackMitigationLoad, 137
.minWetStackPreventionLoad, 137
tms::EngineState

.coolantTemperature, 191
.runTime, 192
.speed, 191
tms::FuelInfo
.maxFuelLevel, 138
tms::GeneratorInfo
.maxStatorTemperature, 139
tms::PowerElectronicsInfo
.maxTemperature, 140
.minTemperature, 140
tms::PowerElectronicsState
.temperature, 197
tms::StorageCellState
.stateOfCharge, 196
.temperature, 196
tms::StorageUpdate
.maxChargeRate, 201
.maxDischargeRate, 201
.stateOfCharge, 200

CONCLUDING MATERIAL

Custodians:

Army – CR4
Navy – YD
Air Force – 11

Preparing activity:

U.S. Army DEVCOM C5ISR Center

Review activities:

Army – CR, AV, MI, MT, TE
Navy – AS, CG
Air Force – 03, 19, 50, 84
DLA – CC, GS
GSA – FAS

NOTE: The activities listed above were interested in this document as of the date of this document. Since organizations and responsibilities can change, you should verify the currency of the information above using the ASSIST Online database at https://assist.dla.mil.