INTERNATIONAL STANDARD

ISO/IEEE 11073-20601

First edition
2010-05-01

Health informatics — Personal health device communication —

Part 20601: Application profile — Optimized exchange protocol

Informatique de santé — Communication entre dispositifs de santé personnels —

Partie 20601: Profil d'application — Protocole d'échange optimisé

Reference number
ISO/IEEE 11073-20601:2010(E)

PDF disclaimer

This PDF file may contain embedded typefaces. In accordance with Adobe's licensing policy, this file may be printed or viewed but shall not be edited unless the typefaces which are embedded are licensed to and installed on the computer performing the editing. In downloading this file, parties accept therein the responsibility of not infringing Adobe's licensing policy. Neither the ISO Central Secretariat nor IEEE accepts any liability in this area.

Adobe is a trademark of Adobe Systems Incorporated.

Details of the software products used to create this PDF file can be found in the General Info relative to the file; the PDF-creation parameters were optimized for printing. Every care has been taken to ensure that the file is suitable for use by ISO member bodies and IEEE members. In the unlikely event that a problem relating to it is found, please inform the ISO Central Secretariat or IEEE at the address given below.

ISO copyright office
Case postale 56 • CH-1211 Geneva 20
Tel. + 41 22 749 01 11
Fax + 41 22 749 09 47
E-mail copyright@iso.org
Web www.iso.org

Institute of Electrical and Electronics Engineers, Inc.
3 Park Avenue, New York • NY 10016-5997, USA
E-mail stds.ipr@ieee.org
Web www.ieee.org

Published in Switzerland

Contents

Page

Foreword

ISO (the International Organization for Standardization) is a worldwide federation of national standards bodies (ISO member bodies). The work of preparing International Standards is normally carried out through ISO technical committees. Each member body interested in a subject for which a technical committee has been established has the right to be represented on that committee. International organizations, governmental and non-governmental, in liaison with ISO, also take part in the work. ISO collaborates closely with the International Electrotechnical Commission (IEC) on all matters of electrotechnical standardization.

IEEE Standards documents are developed within the IEEE Societies and the Standards Coordinating Committees of the IEEE Standards Association (IEEE-SA) Standards Board. The IEEE develops its standards through a consensus development process, approved by the American National Standards Institute, which brings together volunteers representing varied viewpoints and interests to achieve the final product. Volunteers are not necessarily members of the Institute and serve without compensation. While the IEEE administers the process and establishes rules to promote fairness in the consensus development process, the IEEE does not independently evaluate, test, or verify the accuracy of any of the information contained in its standards.

The main task of technical committees is to prepare International Standards. Draft International Standards adopted by the technical committees are circulated to the member bodies for voting. Publication as an International Standard requires approval by at least 75 % of the member bodies casting a vote.

Attention is called to the possibility that implementation of this standard may require the use of subject matter covered by patent rights. By publication of this standard, no position is taken with respect to the existence or validity of any patent rights in connection therewith. ISO/IEEE is not responsible for identifying essential patents or patent claims for which a license may be required, for conducting inquiries into the legal validity or scope of patents or patent claims or determining whether any licensing terms or conditions provided in connection with submission of a Letter of Assurance or a Patent Statement and Licensing Declaration Form, if any, or in any licensing agreements are reasonable or non-discriminatory. Users of this standard are expressly advised that determination of the validity of any patent rights, and the risk of infringement of such rights, is entirely their own responsibility. Further information may be obtained from ISO or the IEEE Standards Association.

ISO/IEEE 11073-20601 was prepared by the 11073 Committee of the Engineering in Medicine and Biology Society of the IEEE (as IEEE Std 11073-20601-2008). It was adopted by Technical Committee ISO/TC 215, *Health informatics*, in parallel with its approval by the ISO member bodies, under the “fast-track procedure” defined in the Partner Standards Development Organization cooperation agreement between ISO and IEEE. Both parties are responsible for the maintenance of this document.

ISO/IEEE 11073 consists of the following parts, under the general title *Health informatics — Personal health device communication (text in parentheses gives a variant of subtitle)*:

— *Part 10101: (Point-of-care medical device communication) Nomenclature*

— *Part 10201: Domain information model*

— *Part 10404: Device specialization — Pulse oximeter*

— *Part 10407: Device specialization — Blood pressure monitor*

— *Part 10408: (Point-of-care medical device communication) Device specialization — Thermometer*

— *Part 10415: (Point-of-care medical device communication) Device specialization — Weighing scale*

— *Part 10417: Device specialization — Glucose meter*

— *Part 10471: (Point-of-care medical device communication) Device specialization — Independant living activity hub*

— *Part 20101: (Point-of-care medical device communication) Application profiles — Base standard*

— *Part 20601: (Point-of-care medical device communication) Application profile — Optimized exchange protocol*

— *Part 30200: (Point-of-care medical device communication) Transport profile — Cable connected*

— *Part 30300: (Point-of-care medical device communication) Transport profile — Infrared wireless*

Introduction

ISO and IEEE 11073 standards enable communication between medical devices and external computer systems. This standard and corresponding IEEE 11073-104zz standards address a need for a simplified and optimized communication approach for personal health devices, which may or may not be regulated devices. These standards align with, and draw upon, the existing clinically focused standards to provide easy management of data from either a clinical or personal health device.

This document addresses a need for an openly defined, independent standard for converting the collected information into an interoperable transmission format so the information can be exchanged between agents and managers.

Other closely related standards include the following:

— ISO/IEEE P11073-00103 [B8][a] provides an overview of the personal health space and defines the underlying use cases and usage models.

— ISO/IEEE 11073-10101 [B12] documents the nomenclature terms that can be used.

— ISO/IEEE 11073-10201:2004 [B13] documents the extensive domain information model (DIM) leveraged by this standard.

— ISO/IEEE 11073-104zz standards define specific device specializations. For example, ISO/IEEE P11073-10404 [B9] defines how interoperable pulse oximeters work.

— ISO/IEEE 11073-20101:2004 [B14] defines the medical device encoding rules (MDER) used in this standard.

[a] The numbers in brackets correspond to the numbers of the bibliography in Annex K.

Health informatics — Personal health device communication —

Part 20601:
Application profile — Optimized exchange protocol

1. Overview

1.1 Scope

Within the context of the ISO/IEEE 11073 family of standards for device communication, this standard defines a common framework for making an abstract model of personal health data available in transport-independent transfer syntax required to establish logical connections between systems and to provide presentation capabilities and services needed to perform communication tasks. The protocol is optimized to personal health usage requirements and leverages commonly used methods and tools wherever possible.

1.2 Purpose

This document addresses a need for an openly defined, independent standard for converting the information profile into an interoperable transmission format so the information can be exchanged to and from personal telehealth devices and compute engines (e.g., cell phones, personal computers, personal health appliances, and set top boxes).

1.3 Context

Figure 1 shows categories and typical devices supporting the personal health space. Agents (e.g., blood pressure monitors, weighing scales, and pedometers) collect information about a person (or persons) and transfer the information to a manager (e.g., cell phone, health appliance, or personal computer) for collection, display, and possible later transmission. The manager may also forward the data to remote support services for further analysis. The information is available from a range of domains including disease management, health and fitness, or aging independently applications.

The communication path between agent and manager is assumed to be a logical point-to-point connection. Generally, an agent communicates with a single manager at any point in time. A manager may communicate with multiple agents simultaneously using separate point-to-point connections.

The overlay shows the focus area of the IEEE 11073™ Personal Health Devices Working Group. The primary concentration is the interface and data exchange between the agents and manager. However, this interface cannot be created in isolation by ignoring the remainder of the solution space. Remaining cognizant of the entire system helps to ensure that data can reasonably move from the agents all the way to the remote support services when necessary. This path may include converting the data format, exchange protocols, and transport protocols across different interfaces. Much of the standardization effort is outside of the scope of the Personal Health Devices Working Group; however, aligning all standardization efforts allows data to flow seamlessly through the overall set of systems.

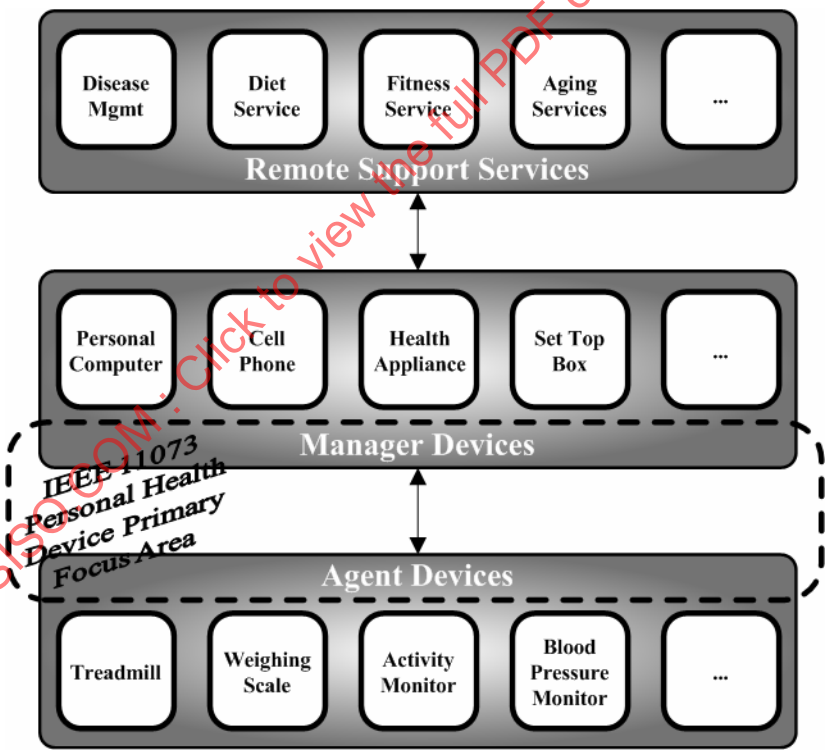


Figure 1 —Overall context of work

Figure 2 shows a hierarchical view of the architecture of an agent or manager superimposed with a view of the related standards. The application layers are, for the most part, not specific to any particular transport. Where necessary, this standard identifies assumptions that require direct support by a transport or a "shim" layer above the transport. This approach allows support for various transports. The definition of the transports is outside the scope of this standard and the working group.

Above the transport layer is the Optimized Exchange Protocol (described in this standard). This protocol consists of two aspects: the application layer services and the definition of the data exchange protocol between agents and managers. The application layer services provide the protocol for connection management and reliable transfer of actions and data between agent and manager. The data exchange protocol defines the commands, agent configuration information, data format, and overall protocol. The Optimized Exchange Protocol provides the basis to support any type of agent. For a specific device type, the reader is directed to the device specialization for that agent to understand the capabilities of the device and its implementation according to this standard. The device specialization indicates which aspects of this standard to comprehend and where further information to implement the device is found.

Above the exchange protocol are device specializations that describe specific details relative to the particular agent (e.g., blood pressure monitor, weighing scale, or pedometer). The specializations describe the details of how these agents work and act as a detailed description for creating a specific type of agent. Additionally, they provide reference to a related standard for further details. The standard numbers reserved for device specializations range from IEEE Std 11073-10401 through IEEE Std 11073-10499, inclusive. When the collection of standards is being referenced, the term *IEEE 11073-104zz* is used where zz could be any number in the range from 01 to 99, inclusive.

The ISO/IEEE P11073-00103 [B8][1] technical report describes the overall personal health space with further definition of the underlying use cases and usage models.

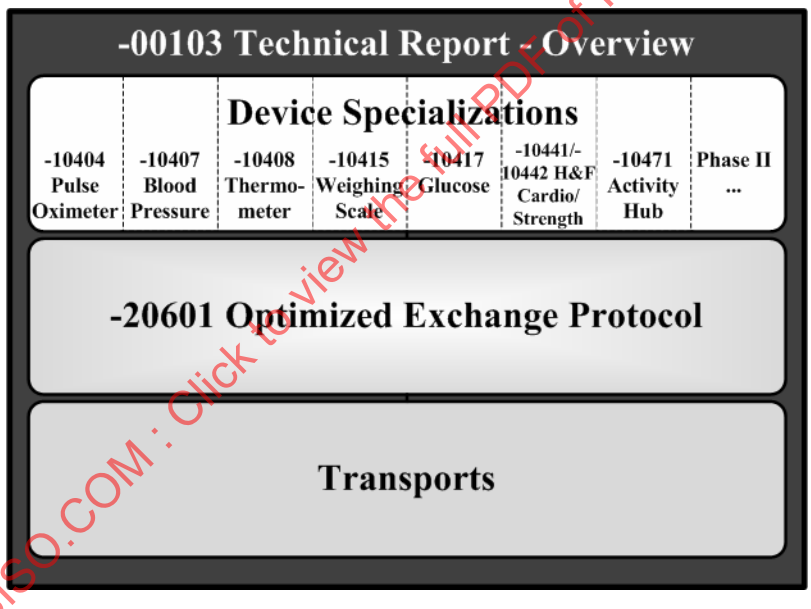


Figure 2 —Document map

The personal health device specializations are not being created independently of all other standards. There are a number of existing standards generated for clinical environments upon which these standards draw. Figure 3 shows the relationship to the remainder of the IEEE 11073 documents. There are two types of relationships:

[1] The numbers in brackets correspond to the numbers of the bibliography in Annex K.

— Drawing ideas and/or content from the other documents (dashed lines)

— Leveraging information from the other document and introducing new content into that document to support this standard (solid lines)

This standard imports information from ISO/IEEE 11073-10201:2004 [B13] and ISO/IEEE 11073-20101:2004 [B14] as normative annexes. If there is a discrepancy between these standards, this standard takes priority. Because of the reuse of constructs from these standards, some of the names appear to be more clinically focused [e.g., medical device system (MDS) instead of personal health device system]; however, to maintain consistency, the traditional names have been preserved.

This standard replicates relevant portions of ISO/IEEE 11073-10101 [B12] and incorporates new nomenclature codes.

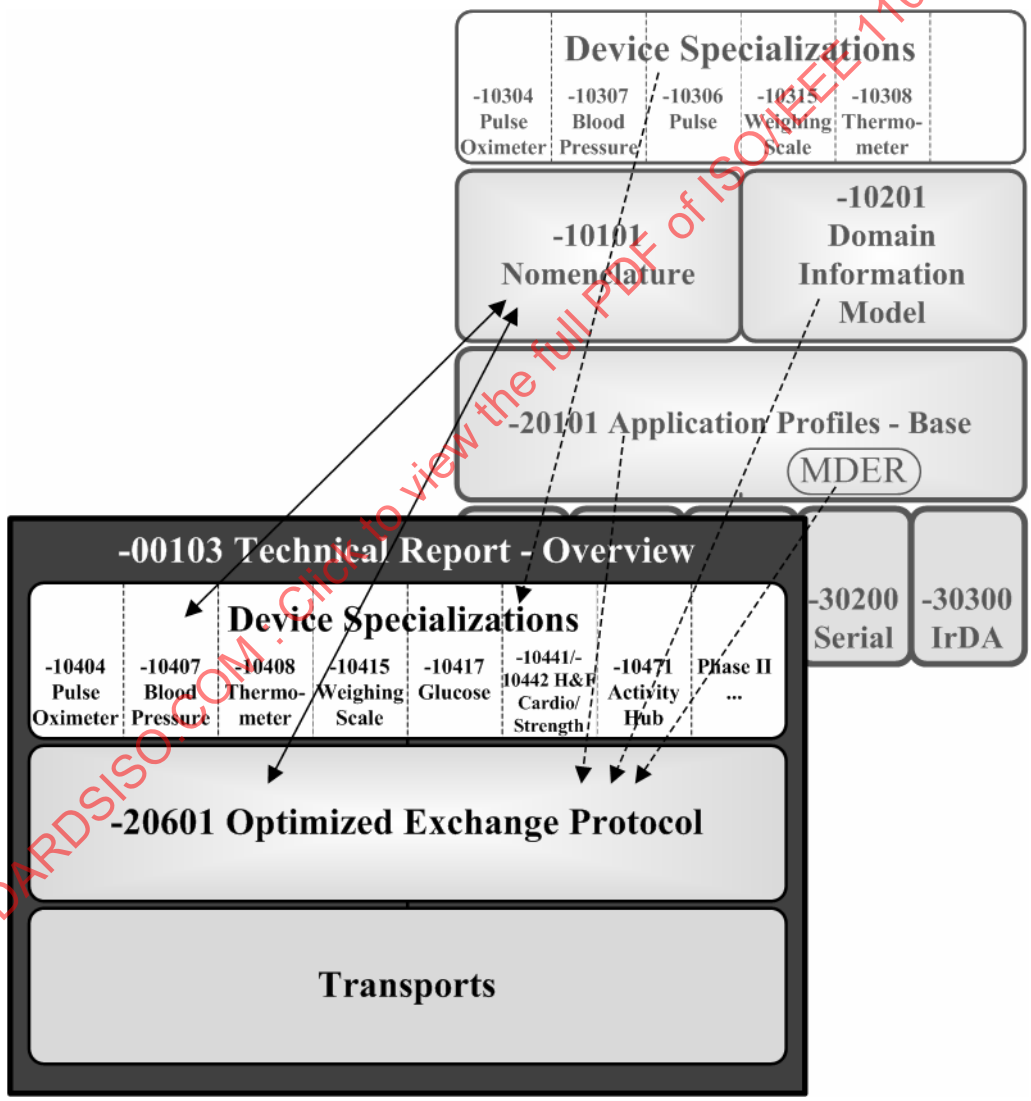


Figure 3 —Relationship to other IEEE 11073 documents

2. Normative references

The following referenced documents are indispensable for the application of this standard (i.e., they must be understood and used; therefore, each referenced document is cited in the text and its relationship to this standard is explained). For dated references, only the edition cited applies. For undated references, the latest edition of the referenced document (including any amendments or corrigenda) applies.

IEEE Std 802®-2001, IEEE Standard for Local and Metropolitan Area Networks: Overview and Architecture.[2]

ITU-T Rec. X.667 (Sept. 2004), Information technology – Open Systems Interconnection – Procedures for the operation of OSI Registration Authorities: Generation and registration of universally unique identifiers (UUIDs) and their use as ASN.1 object identifier components.[3]

3. Definitions, acronyms, and abbreviations

3.1 Definitions

For the purposes of this standard, the following terms and definitions apply. *The Authoritative Dictionary of IEEE Standards* [B6] should be referenced for terms not defined in this clause.

3.1.1 agent: A node that collects and transmits personal health data to an associated manager.

3.1.2 compute engine: *See:* **manager**.

3.1.3 confirmed: An application-level, completion notification service mechanism. For EVENT REPORT services (i.e., the data plane), confirmation allows the agent to know when the manager has "accepted responsibility" for a piece of data so that the agent can delete that data. For the ACTION, GET, and SET services (i.e., the control plane), confirmation allows the manager to know when the agent has "completed" the requested transaction.

3.1.4 device: A physical device implementing either an agent or manager role.

3.1.5 handle: An unsigned 16-bit number that is locally unique and identifies one of the object instances within an agent.

3.1.6 manager: A node receiving data from one or more agent systems. Examples of managers include a cellular phone, health appliance, set top box, or computer system.

3.1.7 personal health device: A device used in personal health applications.

3.1.8 personal telehealth device: *See:* **personal health device**.

3.2 Acronyms and abbreviations

ASCII	American Standard Code for Information Interchange[4]
ASN.1	Abstract Syntax Notation One
APDU	application protocol data unit

[2] IEEE publications are available from the Institute of Electrical and Electronics Engineers, 445 Hoes Lane, Piscataway, NJ 08854, USA (http://standards.ieee.org/).
[3] ITU-T publications are available from the International Telecommunications Union, Place des Nations, CH-1211, Geneva 20, Switzerland/Suisse (http://www.itu.int/).
[4] Note that throughout this standard the term ASCII is used to mean the character set as defined in ISO/IEC 646 (1991) [B7].

AVA	attribute value assertion
BER	binary encoding rules
DIM	domain information model
EUI-64	extended unique identifier (64 bits)
GMDN	Global Medical Device Nomenclature
ICS	implementation conformance statement
ID	identifier
LSB	least significant bit
MDER	medical device encoding rules
MDNF	medical device numeric format
MDS	medical device system
MOC	medical object class
MSB	most significant bit
NaN	not a number
NBO	network byte order
NRes	not at this resolution
NTP	Network Time Protocol
OID	object identifier
OUI	organizationally unique identifier
PDU	protocol data unit
PER	packed encoding rules
POC	*personal health device domain information model* object and class
RC_{assoc}	retry count: association procedure
RTC	real-time clock
RT-SA	real-time sample array
SNTP	Simple Network Time Protocol
TCP	Transmission Control Protocol
TO_{assoc}	timeout: association procedure
TO_{ca}	timeout: confirmed action service
$TO_{cer-mds}$	timeout: confirmed event report service for the MDS object
$TO_{cer-pms}$	timeout: confirmed event report service for the PM-store object
$TO_{cer-scan}$	timeout: confirmed event report service for the scanner object
$TO_{clr-pms}$	timeout: confirmed action service to clear the PM-store object
TO_{config}	timeout: configuration procedure
TO_{cs}	timeout: confirmed set service
TO_{get}	timeout: get service
$TO_{release}$	timeout: association release procedure
TO_{sp-mds}	timeout: special interservice timeout for the MDS object
TO_{sp-pms}	timeout: special segment transfer timeout for the PM-segment object
UTC	universal time coordinated
UUID	universally unique identifier
USB	universal serial bus
XER	Extensible Markup Language (XML) encoding rules

4. Guiding principles

This standard and the other personal health device standards fit in the larger context of the ISO/IEEE 11073 family of standards. The full suite of standards enables agents to interconnect and interoperate with managers and with computerized healthcare information systems.

The communication profile defined in this standard takes into account the specific requirements of personal health agents and managers, which are typically used outside a clinical setting, e.g., mobile or in a person's home:

— Personal health agents typically have very limited computing capabilities.

— Personal health agents typically have a fixed configuration, and they are used with a single manager device.

— Personal health agents are frequently battery powered, mobile devices, using a wireless communication link. Therefore, energy efficiency of the protocol is an important aspect.

— Personal health agents are often not permanently active. For example, a weighing scale may provide data only once or twice a day. An efficient connection procedure is needed for minimum overhead for such devices.

— Personal health managers tend to have more processing power, memory, and storage space so the protocol intentionally places more load on the managers.

— Personal health agents and managers convey information that could be useful to clinical professionals. As such, the quality of the data may be considered to have clinical merit even if acquired in a personal health or remote monitoring environment.

The ISO/IEEE 11073 family of standards is based on an object-oriented systems management paradigm. Data (measurement, state, and so on) are modeled in the form of information objects that are accessed and manipulated using an object access service protocol.

To address the unique requirements of personal health devices, a specialized application profile is defined in this standard. This profile leverages concepts from the ISO/IEEE 11073 family of standards and industry best practices to define an optimized communication profile for this domain:

— Where possible, the communication profile is not specific to any particular transport.

— The information model of the communication profile is built on the ISO/IEEE 11073 domain information model (DIM) and includes optimizations where possible.

— An optimized communication protocol is defined to reduce message size, run-time packet construction, and parsing overheads. This is possible due to the lower complexity of the devices in the personal health domain.

— Required definitions for a protocol implementation are included in this standard, rather than referenced. This approach facilitates easier adoption of this standard. In the case of discrepancies between the normative inclusions and a referenced document, this standard takes precedence.

Where possible, versions of this standard are fully backward compatible with at least two major versions.

NOTE—It is expected that any new additions to the DIM or other relevant parts of the ISO/IEEE 11073 family of standards will be adopted and reflected in future revisions of those standards.[5]

5. Introduction to IEEE 11073 personal health devices

5.1 General

The overall ISO/IEEE 11073 system model is divided into three principal components: the DIM, the service model, and the communications model. These three models work together to represent data, define data access and command methodologies, and communicate the data from an agent to a manager. Because of the tight relationship between the models, the DIM, service model, and communications model are briefly introduced in 5.2, 5.3, and 5.4, respectively, so that when they are described in more detail, in Clause 6, Clause 7, and Clause 8, respectively, the basic concepts are familiar.

[5] Notes in text, tables, and figures of a standard are given for information only and do not contain requirements needed to implement the standard.

5.2 Domain information model (DIM)

The DIM, described in detail in Clause 6, characterizes information from an agent as a set of objects. Each object has one or more attributes. Attributes describe measurement data that are communicated to a manager as well as elements that control behavior and report on the status of the agent.

5.3 Service model

The service model, described in detail in Clause 7, provides data access primitives that are sent between the agent and manager to exchange data from the DIM. These primitives include commands such as Get, Set, Action, and Event Report.

5.4 Communication model

The communication model, described in detail in Clause 8, supports the topology of one or more agents communicating over point-to-point connections to a single manager. For each point-to-point connection, the dynamic system behavior is defined by a connection state machine. The connection state machine defines the states and substates an agent and manager pair passes through, including states related to connection, association, and operation. The communication model also defines in detail the entry, exit, and error conditions for the respective states including various operating procedures for measurement data transmission. The communication model also includes assumptions regarding the underlying communication layers' behavior.

Another function of the communication model is to convert the abstract data modeling (abstract syntax) used in the DIM into a transfer syntax, for example, to binary messages using medical device encoding rules (MDER), that are sent using the communication model.

6. Personal health device DIM

6.1 General

Personal health devices, within this standard, are defined using an object-oriented model. This DIM defines several classes for modeling an agent. The model describes an agent device as a set of objects that represent the data sources, as the elements that a manager can use to control the behavior of the agent, and as the mechanism the agent uses to report updates to the status of agent representation. Agent device objects have attributes that represent information and status for the object.

Manager devices communicate with agent device objects through the use of well-defined methods, such as GET and SET, and are defined in each subclause describing an object. Information, such as measurements, is sent from agent data objects to the manager device using event reports.

The information model for the domain of personal health devices is an object-oriented model defining data objects of agents, including their attributes and methods. The use of an object-oriented information model supports the following:

— Separation of specification from implementation through the principle of encapsulation

— Support for evolution through the principle of inheritance

— Support for backward compatibility through the principle of polymorphism

The objects derived from classes defined in the information model represent all data that an agent system can communicate to a manager system by means of the application protocol defined in this standard. Such data are modeled in the form of object attributes. Furthermore, the information model defines specific data access services in the form of methods that are used for data exchange between agent and manager systems. These services model the application protocol messages (data access primitives) defined in this standard.

Objects define the structure and the capabilities of the agent system. The manager system accesses these objects to retrieve data and/or to control the agent system. This standard does not define an information model of the manager system.

The information model is a hierarchical model that defines the logical structure and capability of a personal health device. At a top level, the MDS object represents the properties and services of the device itself, independent of its health data capabilities. Properties of the MDS object include attributes for device identification and further technical descriptive and state data. The application-specific data (e.g., health data and measurement data) provided by the personal health device are modeled in the form of further information objects that are logically contained in the MDS object. The set of object attributes, together with this containment relation, describes the configuration and, as such, the capabilities of the personal health device.

Note that while the definitions in this standard make use of object orientation to define the information model, this practice does not imply use of object-oriented technologies (e.g., object-oriented programming languages) for the implementation of this standard in a particular device. The model is used to define data structures and access methods (protocol messages) in a consistent and maintainable way. Conformance to these definitions is at a communication protocol message level only. Specifically, definitions in this standard are optimized to allow very simple agent implementations (e.g., by use of pre-defined transmission templates). Likewise, the implementation of the manager device is free to choose a design that makes use of the information objects versus other design alternatives.

This standard makes use of information classes and objects that are defined in ISO/IEEE 11073-10201:2004 [B13], but adapts these to the domain of personal health device communication in the following ways:

— The definition of attributes that are mandatory, optional, or conditional may be different.

— Additional object services may be defined.

— Additional attributes may be defined.

— Some features of the original model might not be used.

6.2 Nomenclature usage

A key aspect of the DIM is that object classes and attributes are referenced using nomenclature codes found in ISO/IEEE 11073-10101 [B12]. By using a consistent nomenclature, interoperability is enhanced as all implementations maintain the same semantic meaning for the numeric codes. Using nomenclature codes also assists with international implementations as the use of strings is reduced.

The ISO/IEEE 11073 nomenclature is defined as a set of context-dependent partitions. The nomenclature code in each context-dependent partition is defined by a 16-bit code that supports up to 65 536 independent terms per partition. The partitions are referenced by a 16-bit partition code. When the partition of the nomenclature code is defined through context, then it is possible to use only the 16-bit term code. If the context is not defined or a context-independent term code is required, then this situation is specified by a

32-bit code constructed from the 16-bit partition code together with the 16-bit term code. Table 1 shows the partitions that are defined in this standard and/or ISO/IEEE 11073-10101 [B12].

Term codes from 0xF000 – 0xFFFF in each partition in the nomenclature are reserved for private (vendor-specified) nomenclature codes.

For each nomenclature term, ISO/IEEE 11073-10101 [B12] defines a systematic name that explains the term, a unique code value, and a reference identifier (ID). The reference ID has the form MDC_XXX_YYY (with MDC referring to "medical device communication"). Throughout this standard, nomenclature terms and nomenclature codes are referenced by the reference ID.

Table 1—Partitions in the nomenclature

Partition number	Nomenclature category
1	Object-oriented (OO)
2	Supervisory control and data acquisition (SCADA)
3	Events
4	Dimensions (units of measurement)
5	Virtual attributes
6	Parameter groups
7	[Body] sites
8	Infrastructure
9–127	Reserved
128	Personal health devices disease management
129	Personal health devices health and fitness
130	Personal health devices aging independently
131–254	Reserved
255	Return codes
256	External nomenclature references
257–1023	Reserved
1024	Private
1025–65 535	Reserved

6.3 Personal health object class definitions

6.3.1 General

The diagram in Figure 4 uses Unified Modeling Language (UML) to represent the information objects of a personal health agent along with class relationships. The top-most object represents the MDS information and its status (see 6.3.2). There are zero or more numeric, real-time sample array (RT-SA), enumeration, scanner, or PM-store objects associated with an MDS object. There are zero or more PM-segments that contain persistent metrics associated with a PM-store. Numeric, RT-SA, and enumeration are derived from a common metric base class that contains common and shared attributes (see 6.3.3). In general, numeric objects represent episodic measurements (see 6.3.4), RT-SA objects represent continuous samples or wave forms (see 6.3.5), enumeration objects represent event annotations (see 6.3.6), and PM-stores (see 6.3.7) along with PM-segments (see 6.3.8) provide a persistent storage mechanism for metrics that are accessed by the manager at a later time. In addition, a scanner object (further defined in 6.3.9) facilitates the reporting of agent-initiated data transfers.

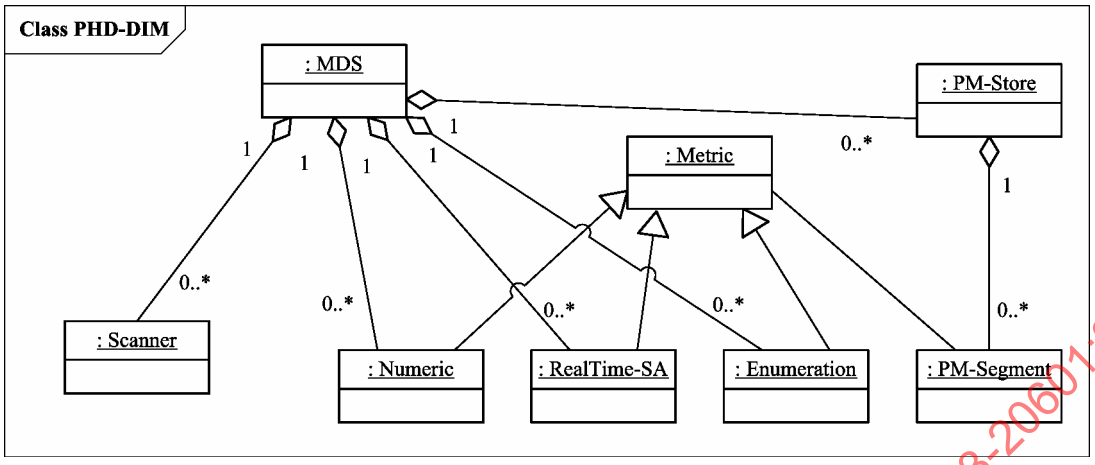


Figure 4 —Personal health device – DIM

Subclauses 6.3.2 through 6.3.9 describe the classes of the personal health device DIM. Each subclause uses the following format:

— The nomenclature code used to identify the class. This code is used during the configuration event to report the class for each object and allows the manager to learn whether the object being specified is numeric, RT-SA, or any of the other classes.

— The attributes defined by the class.

— The methods available.

— The potential events generated by objects instantiated from the class.

— The available services such as getting or setting attributes.

Each attribute data type is defined using an Abstract Syntax Notation One (ASN.1). The ASN.1 definitions for all data types and exchange formats are found in Annex A.

The attributes for each class are defined in tables that specify the name of the attribute, its nomenclature reference ID, its type, a description of the attribute, and its qualifier. The qualifiers mean O — Attribute is Optional, M — Attribute is Mandatory, and C — Attribute is Conditional and depends on the condition stated in the Remark column. Optional attributes may be implemented on an agent. Mandatory attributes shall be implemented by an agent. Conditional attributes shall be implemented if the condition applies and may be implemented otherwise.

The nomenclature code of the object class (e.g., numeric, RT-SA) is sent to the manager at configuration time to create a mirrored object representation. Each object has a Handle attribute that is used to identify the object for operations (to or from the object) and other attributes to represent and convey information on the physical device and its data sources. Attributes are accessed and modified using methods such as GET and SET. Data are transmitted using EVENTs.

6.3.2 MDS class

6.3.2.1 General

Each personal health device agent is defined by an object-oriented model as shown in Figure 4. The top-level object of each agent is instantiated from the MDS class. Each agent has one MDS object. The MDS represents the identification and status of the agent through its attributes.

6.3.2.2 MDS class identification

The nomenclature code to identify the MDS class is MDC_MOC_VMS_MDS_SIMP.

6.3.2.3 MDS class attributes

Table 2 defines the set of MDS attributes that are supported for personal health agent communication. An MDS object shall support all mandatory attributes, but may have a subset of the conditional and optional attributes present.

Table 2—MDS attributes

Attribute name	Attribute ID	Attribute type	Remark	Qual.
Handle	MDC_ATTR_ID_ HANDLE	HANDLE	The Handle attribute represents a reference ID for this object. The value of the MDS Handle attribute shall be 0.	M
System-Type	MDC_ATTR_SYS_TYPE	TYPE	This attribute defines the type of the agent, as defined in nomenclature (e.g., weighing scale). The values shall come from ISO/IEEE 11073-10101 [B12], nom-part-object partition, and subpartition MD-Gen (Medical Device – Generic). Either this attribute or System-Type-Spec-List shall be present. This attribute shall remain unchanged after the configuration is agreed upon.	C
System-Model	MDC_ATTR_ID_MODEL	SystemModel	This attribute defines manufacturer and model number of the agent device. This attribute shall remain unchanged after the configuration is agreed upon.	M

Table 2—MDS attributes

Attribute name	Attribute ID	Attribute type	Remark	Qual.
System-Id	MDC_ATTR_SYS_ID	OCTET STRING	This attribute is an IEEE EUI-64, which consists of a 24-bit unique organizationally unique identifier (OUI) followed by a 40-bit manufacturer-defined ID. The OUI shall be a value assigned by the IEEE Registration Authority (http://standards.ieee.org/regauth/index.html) and shall be used in accordance with IEEE Std 802-2001.[6] This attribute shall remain unchanged after the configuration is agreed upon.	M
Dev-Configuration-Id	MDC_ATTR_DEV_CONFIG_ID	ConfigId	This attribute defines the identification of the agent device configuration. This Dev-Configuration-Id is static during the lifetime of an association; it is normally exchanged during the association procedure. The manager can GET this attribute during operation. If this attribute is queried prior to when the agent and manager agree upon a configuration, the agent shall return the configuration ID that is being offered at that time. For more information on this attribute, see 8.7.6. This attribute shall remain unchanged after the configuration is agreed upon.	M
Attribute-Value-Map	MDC_ATTR_ATTRIBUTE_VAL_MAP	AttrValMap	This attribute defines the attributes that are reported in the fixed format data update messages (see 7.4.5 for more information). Usage of this attribute is mandatory if the agent device uses fixed format value messages to report dynamic data for the object.	C
Production-Specification	MDC_ATTR_ID_PROD_SPECN	ProductionSpec	This attribute defines component revisions, serial numbers, and so on in a manufacturer-specific format. This attribute shall remain unchanged after the configuration is agreed upon.	O
Mds-Time-Info	MDC_ATTR_MDS_TIME_INFO	MdsTimeInfo	This attribute defines the time handling capabilities and the status of the MDS. Usage of this attribute is required if synchronization or settable time is supported.	C

[6] For information on references, see Clause 2.

Table 2—MDS attributes

Attribute name	Attribute ID	Attribute type	Remark	Qual.
Date-and-Time	MDC_ATTR_TIME_ABS	AbsoluteTime	This attribute defines the date and time of an agent with resolution of 1/100 of a second, if available. For more information on this attribute, see 8.12. If the agent reports AbsoluteTime in any other message, it shall report its current value of AbsoluteTime in this attribute.	C
Relative-Time	MDC_ATTR_TIME_REL	RelativeTime	If the agent reports RelativeTime in any other message, it shall report its current value of RelativeTime in this attribute.	C
HiRes-Relative-Time	MDC_ATTR_TIME_REL_HI_RES	HighResRelativeTime	If the agent reports HighResRelativeTime in any other message, it shall report its current value of HighResRelativeTime in this attribute.	C
Date-and-Time-Adjustment	MDC_ATTR_TIME_ABS_ADJUST	AbsoluteTimeAdjust	This attribute reports any date and time adjustments that occur either due to a person's changing the clock or events such as daylight savings time. This is used in event reports only. If queried with Get MDS Object command, this value shall be not present or 0. If the agent ever adjusts the date and time, this attribute is used in an event report to report such adjustment.	C
Power-Status	MDC_ATTR_POWER_STAT	PowerStatus	This attribute reports whether power is being drawn from battery or main power lines and the status of charging.	O
Battery-Level	MDC_ATTR_VAL_BATT_CHARGE	INT-U16	This attribute reports the percentage of battery capacity remaining, which is undefined if value > 100.	O
Remaining-Battery-Time	MDC_ATTR_TIME_BATT_REMAIN	BatMeasure	This attribute represents the predicted amount of operational time left on the batteries. The BatMeasure's unit shall be set to one of MDC_DIM_MIN, MDC_DIM_HR, or MDC_DIM_DAY for minutes, hours, or days, respectively.	O

Table 2—MDS attributes

Attribute name	Attribute ID	Attribute type	Remark	Qual.
Reg-Cert-Data-List	MDC_ATTR_REG_CERT_DATA_LIST	RegCertDataList	This attribute lists various regulatory and/or certification compliance items to which the agent claims adherence as an informative statement. The Implementation Conformace Statements (see Clause 9) take precedence over this attribute and are the legally binding claims. This attribute shall remain unchanged after the configuration is agreed upon.	O
System-Type-Spec-List	MDC_ATTR_SYS_TYPE_SPEC_LIST	TypeVerList	This attribute reports the type(s) of the agent, as defined in nomenclature (e.g., weighing scale). The values shall come from ISO/IEEE 11073-10101 [B12], nom-part-infrastruct partition, subpartition DEVspec, and reference ISO/IEEE 11073-104zz specializations. If an agent does not follow any specialization, the list shall be left blank. This list shall also contain the version of the specialization. Either this attribute or System-Type shall be present. If the agent is multifunction, this attribute shall be present. This attribute shall remain unchanged after the configuration is agreed upon.	C
Confirm-Timeout	MDC_ATTR_CONFIRM_TIMEOUT	RelativeTime	This informational timeout attribute defines the minimum time that the agent shall wait for a Response message from the manager after issuing a Confirmed Event Report invoke message before timing out and transitioning to the Unassociated state. This is an informational attribute for the benefit of the manager. If this attribute is supplied, it shall match the actual timeout value that the agent uses for the Confirmed Event Report generated from the MDS object. This attribute is informational for the manager in the sense that the manager does not use this attribute in an actual implementation of the protocol (i.e., the manager does not time out on an agent-generated Confirmed Event Report).	O

Table 2—MDS attributes

Attribute name	Attribute ID	Attribute type	Remark	Qual.
			However, the manager might wish to use this information to prioritize its handling of a "short" timeout agent over that of a "long" timeout agent.	

The attribute data types are defined in Annex A.

6.3.2.4 MDS object methods

Table 3 defines the methods (actions) available for the MDS object. These methods are invoked using the ACTION service. In Table 3, the Method/Action column defines the name of the method. The Mode column defines whether the method is invoked as an unconfirmed action (i.e., roiv-cmip-action from A.10.2) or a confirmed action (i.e., roiv-cmip-confirmed-action). The Action-type column defines the nomenclature ID to use in the action-type field of an action request and response (see A.10.6). The action-info-args column defines the associated data structure to use in the action message for the action-info-args field of the request. The Resulting action-info-args column defines the structure to use in the action-info-args of the response.

Table 3—MDS object methods

Method/Action	Mode	Action-type	action-info-args	Resulting action-info-args
MDS-Data-Request	Confirmed	MDC_ACT_DATA_REQUEST	DataRequest	DataResponse
Set-Time	Confirmed	MDC_ACT_SET_TIME	SetTimeInvoke	None

— **MDS-Data-Request:**
This method allows the manager system to enable or disable measurement data transmission from the agent (see 8.9.3.3.3 for a description).

— **Set-Time:**
This method allows the manager system to set a real-time clock (RTC) with the absolute time. The agent indicates whether the Set-Time command is valid by using the mds-time-capab-set-clock bit in the Mds-Time-Info attribute (see Table 2).

6.3.2.5 MDS object events

Table 4 defines the potential events sent by the MDS object. A manager shall support all methods defined in Table 4.

Table 4—MDS object events

Event	Mode	Event-type	Event-info parameter	Event-reply-info
MDS-Configuration-Event	Confirmed or unconfirmed	MDC_NOTI_CONFIG	ConfigReport	ConfigReportRsp
MDS-Dynamic-Data-Update-Var	Confirmed or unconfirmed	MDC_NOTI_SCAN_REPORT_VAR	ScanReportInfoVar	-
MDS-Dynamic-Data-Update-Fixed	Confirmed or unconfirmed	MDC_NOTI_SCAN_REPORT_FIXED	ScanReportInfoFixed	-
MDS-Dynamic-Data-Update-MP-Var	Confirmed or unconfirmed	MDC_NOTI_SCAN_REPORT_MP_VAR	ScanReportInfoMPVar	-
MDS-Dynamic-Data-Update-MP-Fixed	Confirmed or unconfirmed	MDC_NOTI_SCAN_REPORT_MP_FIXED	ScanReportInfoMPFixed	-

— **MDS-Configuration-Event:**
This event is sent by the agent during the configuring state of startup if the manager does not already know the agent's configuration from past associations. The event provides static information about the supported measurement capabilities of the agent.

— **MDS-Dynamic-Data-Update-Var:**
This event provides dynamic data (typically measurements) from the agent for some or all of the objects that the agent supports. Data for reported objects are reported using a generic attribute list variable format (see 7.4.5 for details on event report formats). The event is triggered by an MDS-Data-Request from the manager system, or it is sent as an unsolicited message by the agent. For agents that support manager-initiated measurement data transmission, refer to 8.9.3.3.3 for information on controlling the activation and/or period of the data transmission. For agents that do not support manager-initiated measurement data transmission, refer to 8.9.3.3.2 for information on the limited control a manager can assert.

— **MDS-Dynamic-Data-Update-Fixed:**
This event provides dynamic data (typically measurements) from the agent for some or all of the metric objects or the MDS object that the agent supports. Data are reported in the fixed format defined by the Attribute-Value-Map attribute for reported metric objects or the MDS object (see 7.4.5 for details on event report formats). The event is triggered by an MDS-Data-Request from the manager system (i.e., a manager-initiated measurement data transmission), or it is sent as an unsolicited message by the agent (i.e., an agent-initiated measurement data transmission). For agents that support manager-initiated measurement data transmission, refer to 8.9.3.3.3 for information on controlling the activation and/or period of the data transmission. For agents that do not support manager-initiated measurement data transmission, refer to 8.9.3.3.2 for information on the limited control a manager can assert.

— **MDS-Dynamic-Data-Update-MP-Var:**
This is the same as MDS-Dynamic-Data-Update-Var, but allows inclusion of data from multiple persons.

— **MDS-Dynamic-Data-Update-MP-Fixed:**
This is the same as MDS-Dynamic-Data-Update-Fixed, but allows inclusion of data from multiple persons.

6.3.2.6 Other MDS services

6.3.2.6.1 GET service

Any agent supporting two-way communication links shall support the GET service to retrieve the values of all implemented MDS object attributes. The GET service can be invoked as soon as the agent receives the Association Response and moves to the Associated state, including the Operating and Configuring substates.

The manager may request the MDS object attributes of the agent in which case the manager shall send the "Remote Operation Invoke | Get" command (see roiv-cmip-get in A.10.2) with the reserved handle value of 0. The agent shall respond by reporting its MDS object attributes to the manager using the "Remote Operation Response | Get" response (see rors-cmip-get in A.10.2). In the response to a Get MDS Object command, only attributes implemented by the agent are returned.

6.3.2.6.2 SET service

There are currently no settable attributes.

6.3.3 Metric class

6.3.3.1 General

The metric class is the base class for all objects representing measurements, status, and context data. The metric class is not instantiated; therefore, it is never part of the agent configuration. As a base class, it defines all attributes, methods, events, and services that are common for all objects representing measurements.

6.3.3.2 Metric class identification

The nomenclature code to identify the metric class is MDC_MOC_VMO_METRIC. This nomenclature code is not used in an agent or a manager implementation as the metric class is just a base class for other classes.

6.3.3.3 Metric class attributes

Table 5 defines the set of metric attributes that are supported for personal health device communication and that are inherited by all metric-derived classes.

Table 5—Metric attributes

Attribute name	Attribute ID	Attribute type	Remark	Qual.
Handle	MDC_ATTR_ID_HANDLE	HANDLE	The Handle attribute represents a reference ID for this object. Each object shall have a unique ID assigned by the agent. The handle identifies the object in event reports sent to the manager. This attribute shall remain unchanged after the configuration is agreed upon.	M
Type	MDC_ATTR_ID_TYPE	TYPE	This attribute defines a specific static type of this object as defined in the nomenclature (e.g., pulse rate for a specific numeric object instance). The Type attribute contains the nomenclature partition and term code IDs for context-free, extensible identification. This attribute shall remain unchanged after the configuration is agreed upon.	M

Table 5—Metric attributes

Attribute name	Attribute ID	Attribute type	Remark	Qual.
Supplemental-Types	MDC_ATTR_ SUPPLEMENTAL_TYPES	SupplementalTypeList	This attribute may be used to convey supplemental information about the object beyond the Type and Metric-Id attributes. Supplemental information covers conditions like the location of the sensor or the rate at which the object reacts to changes. Device specializations define the expected usage of this attribute. For example, IEEE Std 11073-10471 [B5] defines location nomenclature for specifying the location of a sensor in a home and ISO/IEEE P11073-10404 [B9] defines three supplemental types for fast response, slow response, and spot checking of the pulse rate or blood oxygenation. This attribute shall remain unchanged after the configuration is agreed upon.	O
Metric-Spec-Small	MDC_ATTR_METRIC_SPEC_ SMALL	MetricSpecSmall	This attribute describes the characteristics of the measurements.	M
Metric-Structure-Small	MDC_ATTR_METRIC_ STRUCT_SMALL	MetricStructureSmall	This attribute describes the structure of the measurement. If not present, the manager shall assume MetricStructureSmall := {ms-struct-simple, 0}.	O
Measurement-Status	MDC_ATTR_MSMT_STAT	MeasurementStatus	This attribute indicates the validity of a particular value or set of samples.	O

Table 5—Metric attributes

Attribute name	Attribute ID	Attribute type	Remark	Qual.
Metric-Id	MDC_ATTR_ID_PHYSIO	OID-Type	This attribute may be used to hold an identification that is more specific than the generic ID in the Type attribute. If the Metric-Id-Partition attribute is valued, it defines the nomenclature partition for this attribute. Otherwise, the OID-Type is taken from the same nomenclature partition as defined in the partition field of the Type attribute. This attribute is needed only if identification changes during operation and the Type attribute does not contain full identification. For example, if the Type attribute contains a generic temperature code (MDC_TEMP), this attribute could report a specific, but changing, identification such as MDC_TEMP_ORAL or MDC_TEMP_RECT. Only one attribute of Metric-Id and Metric-Id-List shall be present.	O
Metric-Id-List	MDC_ATTR_ID_PHYSIO_LIST	MetricIdList	This attribute shall be used if a compound observed value is used and does not incorporate the Metric-Id directly (e.g., Compound-Simple-Nu-Observed-Value or Compound-Basic-Nu-Observed-Value) so that elements in the observed value list can be identified individually. The order of the Metric-Id-List shall correspond to the order of the elements in the compound observed value. Only one attribute of Metric-Id and Metric-Id-List shall be present.	C
Metric-Id-Partition	MDC_ATTR_METRIC_ID_PART	NomPartition	This attribute may be used to define the partition from which the Metric-Id or Metric-Id-List nomenclature terms were taken. If not present, the partition is the same as the nomenclature partition defined in the partition field of the Type attribute.	O
Unit-Code	MDC_ATTR_UNIT_CODE	OID-Type	This attribute defines the nomenclature code for the units of measure from the nom-part-dim partition (e.g., MDC_DIM_KILO_G).	O

Table 5—Metric attributes

Attribute name	Attribute ID	Attribute type	Remark	Qual.
Attribute-Value-Map	MDC_ATTR_ATTRIBUTE_VAL_MAP	AttrValMap	This attribute defines the attributes that are reported in the fixed format data update messages. Usage of this attribute is mandatory if the agent uses fixed format value messages to report dynamic measurement data for the object.	C
Source-Handle-Reference	MDC_ATTR_SOURCE_HANDLE_REF	HANDLE	This attribute establishes a relation of this object instance to a source object (e.g., pulse references sourcing SpO2). This attribute is used whenever it is required to model an explicit relation between object instances to define dependencies. The usage of this attribute is defined by device specializations.	O
Label-String	MDC_ATTR_ID_LABEL_STRING	OCTET STRING	This attribute defines the textual representation of Type attribute in printable ASCII. The value of this attribute is at the complete discretion of the agent's manufacturer. It could potentially be useful for a manager as a display string or as an aid in deciding how to behave when it does not understand the MDC_ATTR_ID_TYPE as reported by the agent.	O
Unit-LabelString	MDC_ATTR_UNIT_LABEL_STRING	OCTET STRING	This attribute defines the textual representation of Unit-Code dimension in printable ASCII. The value of this attribute is at the complete discretion of the agent's manufacturer. It could potentially be useful for a manager as a display string or as an aid in deciding how to behave when it does not understand the MDC_ATTR_UNIT_CODE as reported by the agent.	O

Table 5—Metric attributes

Attribute name	Attribute ID	Attribute type	Remark	Qual.
Absolute-Time-Stamp	MDC_ATTR_TIME_STAMP_ABS	AbsoluteTime	This attribute defines the date and time of observation with resolution of 1/100 of a second, if available. For more information on this attribute, see 8.12. If an agent stores data (either in a PM-store object or as a "temporarily stored measurement"), it shall associate a time stamp (Absolute-Time-Stamp, Relative-Time-Stamp, or HiRes-Time-Stamp) with the data.	C
Relative-Time-Stamp	MDC_ATTR_TIME_STAMP_REL	RelativeTime	This attribute defines the time of observation (time stamp in a relative time format/number of clock ticks as defined by RelativeTime data type). If an agent stores data, it shall associate a time stamp (Absolute-Time-Stamp, Relative-Time-Stamp, or HiRes-Time-Stamp) with the data.	C
HiRes-Time-Stamp	MDC_ATTR_TIME_STAMP_REL_HI_RES	HighResRelativeTime	This attribute defines the time of observation (time stamp in a high-resolution relative time format/number of clock ticks as defined by a HighResRelativeTime data type). If an agent stores data, it shall associate a time stamp (Absolute-Time-Stamp, Relative-Time-Stamp, or HiRes-Time-Stamp) with the data.	C
Measure-Active-Period	MDC_ATTR_TIME_PD_MSMT_ACTIVE	FLOAT-Type	This attribute defines the time duration of the observation period in seconds.	O

6.3.3.4 Metric object methods

None.

6.3.3.5 Metric object events

Objects that derive from the metric class do not report their observations directly; rather, the observations are reported through another object, such as the MDS object, a scanner object, or a PM-store object.

6.3.3.6 Other metric services

None

6.3.4 Numeric class

6.3.4.1 General

An instance of the numeric class represents a numerical measurement. The values of a numeric object are sent from the agent to the manager using the EVENT REPORT service (see 7.3). This class is derived from the metric base class.

6.3.4.2 Numeric class identification

The nomenclature code to identify the numeric class is MDC_MOC_VMO_METRIC_NU.

6.3.4.3 Numeric class attributes

Table 6 defines the set of numeric attributes that are supported for personal health device communication.

Table 6—Numeric attributes

Attribute name	Attribute ID	Attribute type	Remark	Qual.
Simple-Nu-Observed-Value	MDC_ATTR_NU_VAL_OBS_SIMP	SimpleNuObsValue	This attribute defines the numerical observed value of the object, without any further embedded status information as found in Nu-Observed-Value. One of Simple-Nu-Observed-Value, Basic-Nu-Observed-Value, Nu-Observed-Value, Compond-Nu-Observed-Value, Compound-Simple-Nu-Observed-Value, or Compound-Basic-Nu-Observed-Value shall be present.	C
Compound-Simple-Nu-Observed-Value	MDC_ATTR_NU_CMPD_VAL_OBS_SIMP	SimpleNuObsValueCmp	This attribute represents an array of Simple-Nu-Observed-Values. One of Simple-Nu-Observed-Value, Basic-Nu-Observed-Value, Nu-Observed-Value, Compond-Nu-Observed-Value, Compound-Simple-Nu-Observed-Value, or Compound-Basic-Nu-Observed-Value shall be present.	C

Table 6—Numeric attributes

Attribute name	Attribute ID	Attribute type	Remark	Qual.
Basic-Nu-Observed-Value	MDC_ATTR_NU_VAL_OBS_BASIC	BasicNuObsValue	This attribute defines the numerical observed value of the object, without any further embedded status information, but with a smaller numerical representation compared to Simple-Nu-Observed-Value. One of Simple-Nu-Observed-Value, Basic-Nu-Observed-Value, Nu-Observed-Value, Compond-Nu-Observed-Value, Compound-Simple-Nu-Observed-Value, or Compound-Basic-Nu-Observed-Value shall be present.	C
Compound-Basic-Nu-Observed-Value	MDC_ATTR_NU_CMPD_VAL_OBS_BASIC	BasicNuObsValueCmp	This attribute represents an array of Basic-Nu-Observed-Values. One of Simple-Nu-Observed-Value, Basic-Nu-Observed-Value, Nu-Observed-Value, Compond-Nu-Observed-Value, Compound-Simple-Nu-Observed-Value, or Compound-Basic-Nu-Observed-Value shall be present.	C
Nu-Observed-Value	MDC_ATTR_NU_VAL_OBS	NuObsValue	This attribute defines the numerical observed value of the object and combines it with measurement status and unit information. It is used when status/unit are dynamic and are always provided together with the value. One of Simple-Nu-Observed-Value, Basic-Nu-Observed-Value, Nu-Observed-Value, Compond-Nu-Observed-Value, Compound-Simple-Nu-Observed-Value, or Compound-Basic-Nu-Observed-Value shall be present.	C

Table 6—Numeric attributes

Attribute name	Attribute ID	Attribute type	Remark	Qual.
Compound-Nu-Observed-Value	MDC_ATTR_NU_CMPD_VAL_OBS	NuObsValueCmp	This attribute combines an array of value, status, and unit. This attribute is available for use only in variable format event reports. One of Simple-Nu-Observed-Value, Basic-Nu-Observed-Value, Nu-Observed-Value, Compond-Nu-Observed-Value, Compound-Simple-Nu-Observed-Value, or Compound-Basic-Nu-Observed-Value shall be present.	C
Accuracy	MDC_ATTR_NU_ACCUR_MSMT	FLOAT-Type	This attribute defines the maximum deviation of actual value from reported observed value (if it can be specified). This attribute shall remain unchanged after the configuration is agreed upon.	O

The attributes Compound-Simple-Nu-Obs-Value, Compound-Basic-Nu-Obs-Value, and Compound-Nu-Observed-Value represent a list concept for observed values. This concept should be used whenever a strong relationship between the individual observed values is given, which might be nomenclature and/or application dependent. The compound observed values share the same static attribution context except for the identification of elements. An example is the blood pressure application, where the nomenclature base term expresses “Blood Pressure” and more specific terms express “Blood Pressure Systolic,” “Blood Pressure Diastolic,” and “Blood Pressure Mean.” The corresponding DIM would contain only a single instance of a numeric object, which would use one of the compound numeric observed value formats to represent the “systolic,” “diastolic,” and “mean” parts of the “Blood Pressure.”

6.3.4.4 Numeric object methods

None

6.3.4.5 Numeric object events

None

6.3.4.6 Other numeric services

None

6.3.5 RT-SA class

6.3.5.1 General

An instance of the RT-SA class represents a wave form measurement. The values of the RT-SA object are sent from the agent to the manager using the EVENT REPORT service (see 7.3). This class is derived from the metric base class.

6.3.5.2 RT-SA class identification

The nomenclature code to identify the RT-SA class is MDC_MOC_VMO_METRIC_SA_RT.

6.3.5.3 RT-SA class attributes

Table 7 defines the set of RT-SA attributes that are supported for personal health device communication.

Table 7—RT-SA attributes

Attribute name	Attribute ID	Attribute type	Remark	Qual.
Sample-Period	MDC_ATTR_TIME_PD_SAMP	RelativeTime	This attributes defines time interval between successive samples given in 1/8 of a millisecond. Thus, 8000 = 1 s. This attribute shall remain unchanged after the configuration is agreed upon.	M
Simple-Sa-Observed-Value	MDC_ATTR_SIMP_SA_OBS_VAL	OCTET STRING	This byte array contains the samples that are reported by the agent in the format that is described by the Sa-Specification and Scale-and-Range-Specification. The length shall be even with padding bytes at the end. Sa-Specification defines the actual number of utilized bytes.	M
Scale-and-Range-Specification	MDC_ATTR_SCALE_SPECN_I8 MDC_ATTR_SCALE_SPECN_I16 MDC_ATTR_SCALE_SPECN_I32	ScaleRangeSpec8 ScaleRangeSpec16 ScaleRangeSpec32	This attribute defines mapping between samples and actual values as well as measurement range. The type depends on sample resolution (sample-size field within sample-type field of Sa-Specification). Exactly one of the three specifications shall be included. This attribute shall remain unchanged after the configuration is agreed upon.	M
Sa-Specification	MDC_ATTR_SA_SPECN	SaSpec	This attribute describes the sample array and sample types. This attribute shall remain unchanged after the configuration is agreed upon.	M

Characteristics of the RT-SA object can be gained by examination of the Sa-Specification attribute. This attribute defines the number of elements in the array and the size of an element and is more thoroughly described in A.3.4.

For agents that support manager-initiated measurement data transmission, refer to 8.9.3.3.3 for information on controlling the activation and/or period of the data transmission. For agents that do not support manager-initiated measurement data transmission, refer to 8.9.3.3.2 for information on the limited control a manager can assert.

— **Scale-and-Range-Specification:**
The Scale-and-Range-Specification attribute defines the coefficients for an algorithm to map the scaled values into their absolute values. The manager shall apply the following algorithm:

$$Y = M \times X + B$$

where

Y = the converted absolute value
M = (upper-absolute-value – lower-absolute-value) / (upper-scaled-value – lower-scaled-value)
B = upper-absolute-value – (M × upper-scaled-value)
X = the scaled value

An example of this algorithm in use can be found in Annex B.

Note that the term *absolute-value* does not refer to the mathematical absolute value in which all values are positive, but rather to the actual, measured value.

6.3.5.4 RT-SA object methods

None

6.3.5.5 RT-SA object events

None

6.3.5.6 Other RT-SA services

None

6.3.6 Enumeration class

6.3.6.1 General

An instance of the enumeration class represents status information and/or annotation information. The values of the enumeration object are coded in the form of normative codes (as defined in ISO/IEEE 11073-10101 [B12]) or in the form of free text. The values of the enumeration object are sent from the agent to the manager using the EVENT REPORT service (see 7.3). This class is derived from the metric base class.

6.3.6.2 Enumeration class identification

The nomenclature code to identify the enumeration class is MDC_MOC_VMO_METRIC_ENUM.

6.3.6.3 Enumeration class attributes

Table 8 defines the set of enumeration attributes that are supported for personal health device communication.

Table 8—Enumeration attributes

Attribute name	Attribute ID	Attribute type	Remark	Qual.
Enum-Observed-Value-Simple-OID	MDC_ATTR_ENUM_OBS_VAL_SIMP_OID	OID-Type	The value is reported as a nomenclature code. If the Enum-Observed-Value-Partition attribute is valued, it defines the nomenclature partition for this attribute. Otherwise, the OID-Type is taken from the same nomenclature partition as defined in the partition field of the Type attribute. One of Enum-Observed-Value-Simple-OID, Enum-Observed-Value-Simple-Bit-Str , Enum-Observed-Value-Basic-Bit-Str, Enum-Observed-Value-Simple-Str, or Enum-Observed-Value shall be present.	C
Enum-Observed-Value-Simple-Bit-Str	MDC_ATTR_ENUM_OBS_VAL_SIMP_BIT_STR	BITS-32	The value is reported as a bit string of 32-bits. One of Enum-Observed-Value-Simple-OID, Enum-Observed-Value-Simple-Bit-Str , Enum-Observed-Value-Basic-Bit-Str, Enum-Observed-Value-Simple-Str, or Enum-Observed-Value shall be present.	C
Enum-Observed-Value-Basic-Bit-Str	MDC_ATTR_ENUM_OBS_VAL_BASIC_BIT_STR	BITS-16	The value is reported as a bit string of 16-bits. One of Enum-Observed-Value-Simple-OID, Enum-Observed-Value-Simple-Bit-Str, Enum-Observed-Value-Basic-Bit-Str, Enum-Observed-Value-Simple-Str, or Enum-Observed-Value shall be present.	C

Table 8—Enumeration attributes

Attribute name	Attribute ID	Attribute type	Remark	Qual.
Enum-Observed-Value-Simple-Str	MDC_ATTR_ENUM_OBS_VAL_SIMP_STR	EnumPrintableString	The value is reported as an ASCII printable string. One of Enum-Observed-Value-Simple-OID, Enum-Observed-Value-Simple-Bit-Str , Enum-Observed-Value-Basic-Bit-Str, Enum-Observed-Value-Simple-Str, or Enum-Observed-Value shall be present.	C
Enum-Observed-Value	MDC_ATTR_VAL_ENUM_OBS	EnumObsValue	This attribute defines a structured observed value that permits additional flexibility about the data type of the reported value. One of Enum-Observed-Value-Simple-OID, Enum-Observed-Value-Simple-Bit-Str , Enum-Observed-Value-Basic-Bit-Str, Enum-Observed-Value-Simple-Str, or Enum-Observed-Value shall be present.	C
Enum-Observed-Value-Partition	MDC_ATTR_ENUM_OBS_VAL_PART	NomPartition	This attribute may be used to define the partition from which the Enum-Observed-Value-Simple-OID or the Enum-Observed-Value's observation OID nomenclature term was taken. If not present, the partition is the same as the nomenclature partition defined in the partition field of the Type attribute.	O

6.3.6.4 Enumeration object methods

None

6.3.6.5 Enumeration object events

None

6.3.6.6 Other enumeration services

None

6.3.7 PM-store class

6.3.7.1 General

An instance of the PM-store class provides long-term storage capabilities for metric data. Data are stored in a variable number of PM-segment objects (see 6.3.8). The stored data of the PM-store object are requested from the agent by the manager using object access services (see 7.3). Anybody not familiar with the PM-store concept may wish to read Annex C for a conceptual overview prior to reading the following subclauses.

6.3.7.2 PM-store class identification

The nomenclature code to identify the PM-store class is MDC_MOC_VMO_PMSTORE.

6.3.7.3 PM-store class attributes

Table 9 defines the set of PM-store attributes that are supported for personal health device communication:

Table 9—PM-store attributes

Attribute name	Attribute ID	Attribute type	Remark	Qual.
Handle	MDC_ATTR_ID_HANDLE	HANDLE	The Handle attribute represents a reference ID for this object. Each object shall have a unique ID assigned by the agent. The handle identifies the object in event reports sent to the manager and to address the object instance in messages invoking object methods. This attribute shall remain unchanged after the configuration is agreed upon.	M
PM-Store-Capab	MDC_ATTR_PM_STORE_CAPAB	PmStoreCapab	This attribute defines basic capabilities of the PM-store object instance. This attribute shall remain unchanged after the configuration is agreed upon.	M
Store-Sample-Algorithm	MDC_ATTR_METRIC_STORE_SAMPLE_ALG	StoSampleAlg	This attribute describes how the sample values stored in the PM-segment have been processed. The StoSampleAlg structure describes the available sampling algorithms. If there is no specific sampling algorithm used (in other words the sample values are raw data), then this attribute shall have a value of st-alg-no-downsampling. This attribute shall remain unchanged after the configuration is agreed upon.	M

Table 9—PM-store attributes

Attribute name	Attribute ID	Attribute type	Remark	Qual.
Store-Capacity-Count	MDC_ATTR_METRIC_STORE_CAPAC_CNT	INT-U32	This attribute is the maximum number of stored PM-segment entries (entries in all contained PM-segments). This attribute shall remain unchanged after the configuration is agreed upon.	O
Store-Usage-Count	MDC_ATTR_METRIC_STORE_USAGE_CNT	INT-U32	This attribute is the actual number of currently stored PM-segment entries (entries in all contained PM-segments).	O
Operational-State	MDC_ATTR_OP_STAT	OperationalState	The attribute indicates if new entries are currently being inserted in any of the contained PM-segments. If any PM-segment contained by this PM-store is having data actively added to it, this attribute shall be set to enabled. Otherwise, it shall be set to disabled.	M
PM-Store-Label	MDC_ATTR_PM_STORE_LABEL_STRING	OCTET STRING	This attribute is an application-dependent label for the PM-store in printable ASCII to indicate its intended use and may be used for display purposes. This attribute shall remain unchanged after the configuration is agreed upon.	O
Sample-Period	MDC_ATTR_TIME_PD_SAMP	RelativeTime	This attribute determines the frequency at which entries are added to the PM-segments. This attribute shall be present either in the PM-store (in which case it applies to all periodically storing PM-segments in the PM-store) or alternatively in each PM-segment, if values are sampled periodically, so the time difference for two entries in the Fixed-Segment-Data is constant (i.e., the pmsc-peri-seg-entries bit in the Pm-Store-Capab attribute is set). This attribute shall remain unchanged after the configuration is agreed upon.	C
Number-Of-Segments	MDC_ATTR_NUM_SEG	INT-U16	This attribute is the number of currently instantiated PM-segments contained in the PM-store. Note that the PM-segment attribute Instance-Number is NOT related to this number (i.e., does not need to be in the range from 0 to Number-Of-Segments), but shall be retrieved with the Get-Segment-Info method.	M

Table 9—PM-store attributes

Attribute name	Attribute ID	Attribute type	Remark	Qual.
Clear-Timeout	MDC_ATTR_CLEAR_ TIMEOUT	RelativeTime	This timeout attribute defines the minimum time that the manager shall wait for the completion of a PM-store clear command. If, after the manager sends a Confirmed Action(Clear Segments) invoke command, the timeout expires before the manager receives the corresponding Confirmed Action response message, the manager shall transition to the Unassociated state as described in 8.9.5.6.	M

The attributes Handle and PM-Store-Capab are part of the agent configuration; therefore, the manager knows the corresponding attribute values after the Configuring procedure.

6.3.7.4 PM-store object methods

Table 10 defines the methods (actions) of a PM-store object. These methods can be invoked using the ACTION service.

Table 10—PM-store object methods

Method/Action	Mode	Action-type	action-info-args	Resulting action-info-args
Clear-Segments	Confirmed	MDC_ACT_SEG_CLR	SegmSelection	(empty)
Get-Segment-Info	Confirmed	MDC_ACT_SEG_GET_ INFO	SegmSelection	SegmentInfoList
Trig-Segment-Data-Xfer	Confirmed	MDC_ACT_SEG_ TRIG_XFER	TrigSegmDataXfer-Req	TrigSegmDataXferRsp

If an agent supports the PM-store class, the support of the Get-Segment-Info and Trig-Segment-Data-Xfer methods is mandatory. Support for the Clear-Segments method is optional and is indicated in the PM-Store-Capab attribute.

— **Clear-Segments:**
This method allows the manager to delete the data currently stored in one or more selected PM-segments. All entries in the selected PM-segments are deleted. If the agent supports a variable number of PM-segments, the agent may delete empty PM-segments. Additionally, the agent may clear PM-segments without direction from the manager (e.g., the user of the agent could choose to delete data stored on the agent); however, if doing so while in an Associated state, the Instance-Number shall remain empty for the duration of the association. The Instance-Number of all other PM-segments shall be unaffected by clearing a segment. If this method is invoked on a PM-segment that has the Operational-State attribute set to enabled, the agent shall reply with a not-allowed-by-object error (roer) with a return code of MDC_RET_CODE_OBJ_BUSY.

Note that the behavior of the Clear-Segments method is application specific. The method may remove

all entries from the specified PM-segment, leaving it empty, or it may remove the defined PM-segment completely. This behavior is defined in the PM-Store-Capab attribute. For specific applications, recommendations are defined in corresponding device specializations, making use of the PM-store.

— **Get-Segment-Info:**
This method allows the manager to retrieve PM-segment attributes of one or more PM-segments, with the exception of the Fixed-Segment-Data attribute which contains the actual stored data and is retrieved by using the Trig-Segment-Data-Xfer method. In particular, the Get-Segment-Info method allows the manager to retrieve the Instance-Number attributes of the PM-segment object instances and their data contents.

— **Trig-Segment-Data-Xfer:**
This method allows the manager to start the transfer of the Fixed-Segment-Data attribute of a specified PM-segment. The agent indicates in the response if it accepts or denies this request. If the agent accepts the request, the agent sends Segment-Data-Event messages as described in 6.3.7.5. If this method is invoked on a PM-segment that has the Operational-State attribute set to enabled, the agent shall reply with a not-allowed-by-object error (roer) with a return code of MDC_RET_CODE_OBJ_BUSY.

6.3.7.5 PM-store object events

Table 11 defines the potential events sent by a PM-store object:

Table 11 —PM-store object events

Event	Mode	Event-type	Event-info parameter	Event-reply-info
Segment-Data-Event	Confirmed	MDC_NOTI_ SEGMENT_DATA	SegmentDataEvent	SegmentDataResult

— **Segment-Data-Event:**
This event sends data stored in the Fixed-Segment-Data of a PM-segment from the agent to the manager. The event is triggered by the manager by the Trig-Segment-Data-Xfer method. Once the data transfer is triggered, the agent sends Segment-Data-Event messages until the complete Fixed-Segment-Data is transferred or the transfer is aborted by the manager or agent. See Transfer PM-segment content in 8.9.3.4.2 for a full description.

It is encouraged to place as many segment entries contained in a Segment-Data-Event as possible to reduce the number of messages required for the transfer of the segment.

Support for the event by the agent is mandatory if the agent supports PM-store objects.

6.3.7.6 Other PM-store services

6.3.7.6.1 GET service

Support for the GET service shall be provided by any agent that supports one or more PM-store objects only while in the Operating state. The manager uses the GET service to retrieve the values of all PM-store object attributes.

The manager may request the PM-store object attributes of the agent in which case the manager shall send the "Remote Operation Invoke | Get" command (see roiv-cmip-get in A.10.2) with the handle value of the PM-store object, as defined in the agent's configuration. The agent shall respond by reporting its PM-store object attributes to the manager using the "Remote Operation Response | Get" response (see rors-cmip-get in A.10.2).

6.3.8 PM-segment class

6.3.8.1 General

An instance of the PM-segment class represents a persistently stored episode of measurement data. A PM-segment object is not part of the static agent configuration because the number of instantiated PM-segment instances may dynamically change. The manager accesses PM-segment objects indirectly by methods and events of the PM-store object.

6.3.8.2 PM-segment class identification

The nomenclature code to identify the PM-segment class is MDC_MOC_PM_SEGMENT.

6.3.8.3 PM-segment class attributes

Table 12 defines the set of PM-segment attributes that are supported for personal health device communication.

Table 12—PM-segment attributes

Attribute name	Attribute ID	Attribute type	Remark	Qual.
Instance-Number	MDC_ATTR_ID_INSTNO	InstNumber	The Instance-Number is the ID of a specific PM-segment object instance. It is used by the manager to address a PM-segment.	M
PM-Segment-Entry-Map	MDC_ATTR_PM_SEG_MAP	PmSegmentEntryMap	This attribute defines the format and contents of one stored entry. An entry has an optional header containing information applicable to all elements in the entry. The entry then contains one or more elements, defined by the class, metric ID, handle, and an attribute value map defining the object attributes for each element in the PM-segment.	M

Table 12—PM-segment attributes

Attribute name	Attribute ID	Attribute type	Remark	Qual.
PM-Seg-Person-Id	MDC_ATTR_PM_SEG_ PERSON_ID	PersonId	This standard supports devices that have simple support for data from multiple persons. A person ID is used to differentiate different persons. If the PM-store is able to store data for multiple persons, it shall set the pmsc-multi-person bit in the PM-Store-Capab attribute. If this bit is set, all PM-segment instances contained in the PM-store shall support the PM-Seg-Person-Id attribute. Otherwise, this attribute is not defined.	C
Operational-State	MDC_ATTR_OP_STAT	OperationalState	This attribute indicates if new entries are currently being inserted into this PM-segment. If this PM-segment is having data actively added to it, this attribute shall be set to enabled. Otherwise, it shall be set to disabled.	M
Sample-Period	MDC_ATTR_TIME_PD_ SAMP	RelativeTime	This attribute defines the frequency at which entries are added to the PM-segment. This attribute shall be present either in the PM-store (in which case it applies to all periodically storing PM-segments in the PM-store) or alternatively in each PM-segment. If values are sampled, then the pmsc-peri-seg-entries bit in the PM-Store-Capab attribute shall be set.	C
Segment-Label	MDC_ATTR_PM_SEG_ LABEL_STRING	OCTET STRING	This attribute is an application-dependent label in printable ASCII for the segment to indicate its intended use and may be used for display purposes.	O
Segment-Start-Abs-Time	MDC_ATTR_TIME_START_ SEG	AbsoluteTime	This attribute defines the start time of segment.	O
Segment-End-Abs-Time	MDC_ATTR_TIME_END_ SEG	AbsoluteTime	This attribute defines the end time of segment.	O
Date-and-Time-Adjustment	MDC_ATTR_TIME_ABS_ ADJUST	AbsoluteTimeAdjust	This attribute reports any date and time adjustments that occur either due to a person's changing the clock or events such as daylight savings time. If the agent ever adjusts the date and time, this attribute is reports such adjustment.	C

Table 12—PM-segment attributes

Attribute name	Attribute ID	Attribute type	Remark	Qual.
Segment-Usage-Count	MDC_ATTR_SEG_USAGE_CNT	INT-U32	This attribute gives the actual (current) number of stored entries.	O
Segment-Statistics	MDC_ATTR_SEG_STATS	SegmentStatistics	This attribute defines the array for reporting minimum, mean, maximum statistics for each element to be tagged.	O
Fixed-Segment-Data	MDC_ATTR_SEG_FIXED_DATA	N/A The data is stored internal to the device and so this data type never occurs in any protocol definition directly.	This attribute defines the segment data transferred as an array of entries in a format as specified in the PM-Segment-Entry-Map attribute. This is defined here as an opaque data structure without a defined data type. Note that this attribute is not directly accessible; it is only retrievable by the manager using the PM-store Trig-Segm-Data-Xfer method.	M
Confirm-Timeout	MDC_ATTR_CONFIRM_TIMEOUT	RelativeTime	This informational timeout attribute defines the minimum time that the agent shall wait for a Response message from the manager after issuing a Confirmed Event Report invoke message before timing out and transitioning to the Unassociated state. This is an informational attribute for the benefit of the manager. If this attribute is supplied, it shall match the actual timeout value that the agent uses for the Confirmed Event Report generated from the PM-store object. This attribute is informational for the manager in the sense that the manager does not use this attribute in an actual implementation of the protocol (i.e., the manager does not time out on an agent-generated Confirmed Event Report). However, the manager might wish to use this information to prioritize its handling of a “short” timeout agent over that of a “long” timeout agent.	O

Table 12—PM-segment attributes

Attribute name	Attribute ID	Attribute type	Remark	Qual.
Transfer-Timeout	MDC_ATTR_TRANSFER_TIMEOUT	RelativeTime	This timeout attribute defines the minimum time that the manager shall wait for the complete transfer of PM-segment information. If the timeout expires prior to the reception of the complete PM-segment, the manager shall transition to the Unassociated state as described in 8.9.5.6.	M

The Fixed-Segment-Data attribute stores an array of identically formatted entries. In cases where a measurement is not available at the required time, then the value for a numeric measurement represented by the (S)FLOAT-type data type shall use the special NaN (not a number) value to indicate an unavailable value.

The Fixed-Segment-Data attribute may hold very large amounts of data, depending on the agent capabilities and the application. An agent may choose to restrict the maximum size of the Fixed-Segment-Data attribute in a way that is aligned with the maximum transmission unit of the transport system. In order to support this type of behavior, a manager shall be able to support the transfer of Fixed-Segment-Data attributes in multiple application messages.

6.3.8.4 PM-segment object methods

None

6.3.8.5 PM-segment object events

None

6.3.8.6 Other PM-segment services

None

6.3.9 Scanner classes

6.3.9.1 General

A scanner object is an observer and “summarizer” of object attribute values. It observes attributes of metric objects (e.g., numeric objects) and generates summaries in the form of notification event reports. See Figure 5 for the class hierarchy of the scanner classes. Each class is described in 6.3.9.2 through 6.3.9.5, respectively.

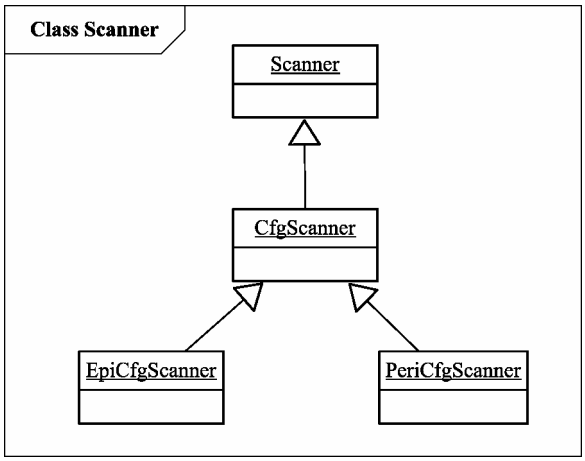


Figure 5 —Personal health device – DIM – scanner model

The different scanner classes (periodic and episodic) as well as the different instances should be used to distribute different data types across one or many data flows represented by a scanner instance. A pulse oximetry application might choose to have a periodic configurable scanner for RT-SA objects with a Reporting-Interval of 50 ms, a periodic configurable scanner with a Reporting-Interval of 1 s for numeric and enumeration objects, and an episodic configurable scanner for beat-to-beat metric objects (numeric or enumeration objects).

6.3.9.2 Scanner class

6.3.9.2.1 General

The scanner class is an abstract class defining attributes, methods, events, and services that are common for its subclasses. As such, it cannot be instantiated.

The scanner concept provides three different event report notifications: variable format, fixed format, and grouped format. See 7.4.5 for the reporting of observed object attributes. The event report formats are described further in 6.3.9.4.5, 6.3.9.5.5, and A.11.5, respectively.

More specialized scanner classes are derived from the scanner base class.

6.3.9.2.2 Scanner class identification

The nomenclature code to identify the scanner class is MDC_MOC_SCAN.

6.3.9.2.3 Scanner class attributes

Table 13 defines the set of scanner attributes that are supported for personal health device communication.

Table 13—Scanner attributes

Attribute name	Attribute ID	Attribute type	Remark	Qual.
Handle	MDC_ATTR_ID_HANDLE	HANDLE	Scanners are identified by handles. This attribute shall remain unchanged after the configuration is agreed upon.	M
Operational-State	MDC_ATTR_OP_STAT	OperationalState	This attribute defines if scanner is active and can be set by the manager.	M
Scan-Handle-List	MDC_ATTR_SCAN_ HANDLE_LIST	HANDLEList	This attribute defines the metric-derived objects that might be reported in the Unbuf-Scan-Report-Var, Buf-Scan-Report-Var, Unbuf-Scan-Report-Fixed, Buf-Scan-Report-Fixed, or any of the four multiple-person equivalents. For episodic scanners, a particular object is included in an event report whenever there are changes in one or more attribute values. For periodic scanners, the scanned objects and attribute values are reported in each period. The manager shall not assume the order of the objects contained in the event reports is the same as the order of the Scan-Handle-List. This attribute shall be present if any of these eight reporting styles are used by the scanner.	C
Scan-Handle-Attr-Val-Map	MDC_ATTR_SCAN_ HANDLE_ATTR_VAL_MAP	HandleAttrValMap	This attribute defines the metric-derived objects, the attributes, and the order in which objects and attribute values are reported in a Unbuf-Scan-Report-Grouped, Buf-Scan-Report-Grouped, Unbuf-Scan-Report-MP-Grouped, or Buf-Scan-Report-MP-Grouped. All values shall be present to maintain a consistent layout of message. This attribute shall be present if any of these four reporting styles are used.	C

6.3.9.2.4 Scanner object methods

None

6.3.9.2.5 Scanner object events

See the derived class event descriptions in 6.3.9.4.5 and 6.3.9.5.5.

6.3.9.2.6 Other scanner services

— **SET service:**
Agents that have scanner derived objects shall support the SET service for the Operational-State attribute of the scanner objects.

6.3.9.3 CfgScanner class

6.3.9.3.1 General

The CfgScanner class is an abstract class defining attributes, methods, events, and services that are common for its subclasses. In particular, it defines the communication behavior of a configurable scanner object. As such, it cannot be instantiated.

6.3.9.3.2 Configurable scanner class identification

The nomenclature code to identify the configurable scanner class is MDC_MOC_SCAN_CFG.

6.3.9.3.3 Configurable scanner class attributes

Table 14 defines the set of scanner attributes that are supported for personal health device communication.

Table 14—Configurable scanner attributes

Attribute name	Attribute ID	Attribute type	Remark	Qual.
Confirm-Mode	MDC_ATTR_CONFIRM_MODE	ConfirmMode	This attribute defines whether event reports are sent confirmed or unconfirmed.	M
Confirm-Timeout	MDC_ATTR_CONFIRM_TIMEOUT	RelativeTime	This informational timeout attribute defines the minimum time that the agent shall wait for a Response message from the manager after issuing a Confirmed Event Report invoke message before timing out and transitioning to the Unassociated state. This is an informational attribute for the benefit of the manager. If this attribute is supplied, it shall match the actual timeout value that the agent uses for the Confirmed Event Report generated from the scanner object. This attribute is informational for the manager in the sense that the manager does not use this attribute in an actual	C

Table 14—Configurable scanner attributes

Attribute name	Attribute ID	Attribute type	Remark	Qual.
			implementation of the protocol (i.e., the manager does not time out on an agent-generated Confirmed Event Report). However, the manager might wish to use this information to prioritize its handling of a "short" timeout agent over that of a "long" timeout agent.	
Transmit-Window	MDC_ATTR_TX_WIND	INT-U16	This attribute defines informative data provided by the agent that may help a manager optimize its configuration. The Transmit-Window represents the number of unacknowledged confirmed event reports that the agent will allow to be outstanding. For this version of this standard, the attribute shall have only a value of 1.	O

Figure 6 illustrates the handling of the optional transmit queue when Transmit-Window is greater than 1.

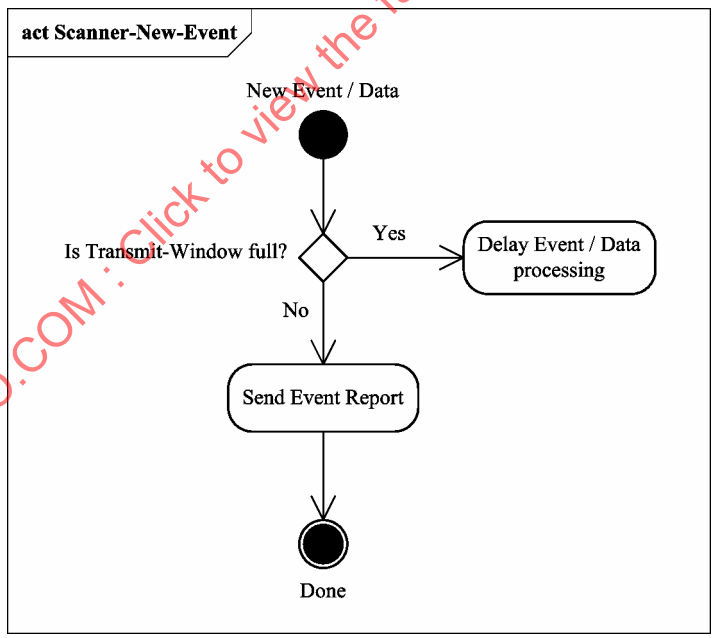


Figure 6 —Configurable scanner Transmit-Window handling

6.3.9.3.4 Configurable scanner object methods

None

6.3.9.3.5 Configurable scanner object events

See the derived class event descriptions in 6.3.9.4.5 and 6.3.9.5.5.

6.3.9.3.6 Other configurable scanner services

None

6.3.9.4 EpiCfgScanner class

6.3.9.4.1 General

The EpiCfgScanner class represents a class that can be instantiated. EpiCfgScanner objects are used to send reports containing episodic data, that is, data not having a fixed period between each data value. A report is sent whenever one of the observed attributes changes value; however, two consecutive event reports shall not have a time interval less than the value of the Min-Reporting-Interval attribute.

6.3.9.4.2 Episodic configurable scanner class identification

The nomenclature code to identify the episodic configurable scanner class is MDC_MOC_SCAN_CFG_EPI.

6.3.9.4.3 Episodic configurable scanner class attributes

Table 15 defines the set of episodic configurable scanner attributes that are supported for personal health device communication

Table 15—Episodic configurable scanner attributes

Attribute name	Attribute ID	Attribute type	Remark	Qual.
Min-Reporting-Interval	MDC_ATTR_SCAN_REP_PD_MIN	RelativeTime	This attribute provides an estimate of the expected minimum time between any two successive event reports.	O

6.3.9.4.4 Episodic configurable scanner object methods

None

6.3.9.4.5 Episodic configurable scanner object events

Table 16 defines the potential events sent by an episodic configurable scanner object. The event reports are classified as unbuffered since the agent sends the event when the episode occurs and it does not need to buffer the information waiting for the next periodic transmission. If an agent supports an episodic configurable scanner, it shall support at least one of the events identified in Table 16.

Table 16—Episodic configurable scanner object events

Event	Mode	Event-type	Event-info parameter	Event-reply-info
Unbuf-Scan-Report-Var	Confirmed or unconfirmed	MDC_NOTI_UNBUF_SCAN_REPORT_VAR	ScanReportInfoVar	—
Unbuf-Scan-Report-Fixed	Confirmed or unconfirmed	MDC_NOTI_UNBUF_SCAN_REPORT_FIXED	ScanReportInfoFixed	—
Unbuf-Scan-Report-Grouped	Confirmed or unconfirmed	MDC_NOTI_UNBUF_SCAN_REPORT_GROUPED	ScanReportInfoGrouped	—
Unbuf-Scan-Report-MP-Var	Confirmed or unconfirmed	MDC_NOTI_UNBUF_SCAN_REPORT_MP_VAR	ScanReportInfoMPVar	—
Unbuf-Scan-Report-MP-Fixed	Confirmed or unconfirmed	MDC_NOTI_UNBUF_SCAN_REPORT_MP_FIXED	ScanReportInfoMPFixed	—
Unbuf-Scan-Report-MP-Grouped	Confirmed or unconfirmed	MDC_NOTI_UNBUF_SCAN_REPORT_MP_GROUPED	ScanReportInfoMPGrouped	—

NOTE 1—For variable and fixed format reports, if no attribute of an object changes its value, then no data of this object are included in the scan report.

NOTE 2—Because an episodic scanner does not buffer any changes and does not have an update period specification attribute (this is not needed because updates are sent on value changes), attribute change notifications shall be sent at a rate that ensures no data loss. For example, to ensure that no metric value changes more than once between scans of object attributes, the episodic scanner implementation should check for changes at a rate at least as fast as the shortest update period of the metric instances in the scanner's scan list.

— **Unbuf-Scan-Report-Var:**
This event style reports summary data about any objects and attributes that the scanner monitors. The event is triggered whenever data values change and the variable message format (type/length/value) is used when reporting data that changed.

— **Unbuf-Scan-Report-Fixed:**
This event style is used whenever data values change and the fixed message format of each object is used to report data that changed.

— **Unbuf-Scan-Report-Grouped:**
This style is used when the scanner object is used to send the data in its most compact format. The Handle-Attr-Val-Map attribute describes the objects and attributes that are included and the format of the message.

— **Unbuf-Scan-Report-MP-Var:**
This is the same as Unbuf-Scan-Report-Var, but allows inclusion of data from multiple persons.

— **Unbuf-Scan-Report-MP-Fixed:**
This is the same as Unbuf-Scan-Report-Fixed, but allows inclusion of data from multiple persons.

— **Unbuf-Scan-Report-MP-Grouped:**
This is the same as Unbuf-Scan-Report-Grouped, but allows inclusion of data from multiple persons.

6.3.9.4.6 Other episodic configurable scanner services

None

6.3.9.5 PeriCfgScanner class

6.3.9.5.1 General

The PeriCfgScanner class represents a class that can be instantiated. PeriCfgScanner objects are used to send reports containing Periodic data, that is, data sampled during fixed periods. It buffers any data value changes to be sent as part of a periodic report. Event reports shall be sent with a time interval equal to the Reporting-Interval attribute value.

The number of observations for each metric object is dependent on the metric object's update interval and the scanner's Reporting-Interval.

Example: A periodic configurable scanner is set up to "scan" two metric objects with a Reporting-Interval of 1 s. The two objects update their corresponding observed value periodically with an interval of 1 s and ½ s, respectively. The periodic configurable scanner then issues event reports every second containing one observation scan of metric object #1 and two observation scans of metric object #2.

6.3.9.5.2 Periodic configurable scanner object identification

The nomenclature code to identify the periodic configurable scanner class is MDC_MOC_SCAN_CFG_PERI.

6.3.9.5.3 Periodic configurable scanner object attributes

Table 17 defines the set of scanner object attributes that are supported for personal health device communication.

Table 17—Periodic configurable scanner object attributes

Attribute name	Attribute ID	Attribute type	Remark	Qual.
Reporting-Interval	MDC_ATTR_SCAN_REP_PD	RelativeTime	Reporting period of the event reports.	M

6.3.9.5.4 Periodic configurable scanner object methods

None

6.3.9.5.5 Periodic configurable scanner object events

Table 18 defines the potential events sent by a periodic configurable scanner object. If an agent supports a periodic configurable scanner, it shall support at least one of the events identified in Table 18.

Table 18—Periodic configurable scanner object events

Event	Mode	Event-type	Event-info parameter	Event-reply-info
Buf-Scan-Report-Var	Confirmed or unconfirmed	MDC_NOTI_BUF_SCAN_REPORT_VAR	ScanReportInfoVar	—
Buf-Scan-Report-Fixed	Confirmed or unconfirmed	MDC_NOTI_BUF_SCAN_REPORT_FIXED	ScanReportInfoFixed	—
Buf-Scan-Report-Grouped	Confirmed or unconfirmed	MDC_NOTI_BUF_SCAN_REPORT_GROUPED	ScanReportInfoGrouped	—
Buf-Scan-Report-MP-Var	Confirmed or unconfirmed	MDC_NOTI_BUF_SCAN_REPORT_MP_VAR	ScanReportInfoMPVar	—
Buf-Scan-Report-MP-Fixed	Confirmed or unconfirmed	MDC_NOTI_BUF_SCAN_REPORT_MP_FIXED	ScanReportInfoMPFixed	—
Buf-Scan-Report-MP-Grouped	Confirmed or unconfirmed	MDC_NOTI_BUF_SCAN_REPORT_MP_GROUPED	ScanReportInfoMPGrouped	—

All of the event report styles listed in Table 18 are buffered equivalents to their unbuffered counterparts in 6.3.9.4.5. One difference is that the scanner buffers data over the reporting interval and sends a single message at the end of the interval. A second difference is that the same objects and attributes are included in each report regardless of whether their values have changed.

6.3.9.5.6 Other periodic configurable scanner services

None

6.4 Information model extensibility rules

The information model is extended in an implementation by using additional object attributes for the objects defined in this standard that are defined in ISO/IEEE 11073-10201:2004 [B13].

Another extension available is to use private (e.g., manufacturer-specific) object attributes and/or methods for the objects defined in this standard. Private attributes shall be identified by assigning nomenclature codes from the private numbering space (0xF000 – 0xFFFF) within the corresponding partition as defined in ISO/IEEE 11073-10101 [B12].

An implementation of a manager system shall process a message fully by skipping any unknown attributes (e.g., vendor-specified attributes) and ignoring the assigned data values of such attributes, without protocol errors. The implementation may log the occurrence of such attributes (e.g., in log files) as appropriate.

7. Personal health device service model

7.1 General

The service model defines the conceptual mechanisms for data exchange services. These services are mapped to messages that are exchanged between the agent and manager. Protocol messages within the ISO/IEEE 11073 family of standards are defined in ASN.1. The messages defined in this standard can coexist with messages defined in other standard profiles defined in the ISO/IEEE 11073 family of standards.

The protocol messages are structured as follows:

— The upper layer protocol frame structure separates the connection-management-related command messages (association messages) from the upper layer object-related messages (data and service communication).

— The upper layer frame structure, in particular, provides a message type and length field.

— The protocol, when using MDER, allows agents to store predefined transmission templates and modify just the fixed location, varying parts before sending.

7.2 Association service

The association service manages the association between an agent and manager. The following messages are part of the association service:

— An Association Request establishes an upper layer connection over an existing transport connection.

— An Association Response accepts the Association Request if the connection is bidirectional.

— A Release Request terminates an upper layer association gracefully.

— A Release Response confirms termination of the upper layer association if the connection is bidirectional.

— An Abort terminates an upper layer association immediately and without response. This is usually sent as a result of a failure.

7.3 Object access services

Object access services are used to access the information objects defined in the DIM. These services are, in particular, used for the data reporting and data access functions provided by an agent.

The following generic object access services are supported:

— GET service: used by the manager to retrieve the values of the agent MDS object and PM-store attributes. The list of MDS object attributes is given in 6.3.2.3, and the list of PM-store attributes is given in 6.3.7.3

— SET service: used by the manager to set values of attributes of the agent's object. Currently, only the scanner objects support the SET service (see 6.3.9.2.6).

— EVENT REPORT service: used by the agent to send configuration updates and measurement data to the manager. The list of event reports is given in 6.3.2.5, 6.3.7.5, 6.3.9.4.5, and 6.3.9.5.5.

— ACTION service: used by the manager to invoke actions (or methods) supported by the agent. An example is MDS-Data-Request action, which is used to request measurement data from the agent. The list of methods is given in 6.3.2.4 and 6.3.7.4.

Access to agent objects via the Get request shall be considered invalid unless one of the following conditions is true:

— The agent is in the Operating state, and the GET references the MDS object or an object handle that has been declared during configuration.

— The agent is in the Associated state, and the GET references the MDS object.

A manager receiving a confirmed event report from an agent shall respond with either a rors-cmip-confirmed-event-report or an appropriate roer error message with a suitable return code.

If a request for a confirmed action is received by an agent that does not support the action, the agent shall reply with an error (roer) with an error value of no-such-action. If an error occurs in executing a confirmed action, then the error may be indicated by returning an error (roer) with an appropriate error value and, where appropriate, additional information on the error may be included in the parameter field using one of the return codes from the return codes partition.

7.4 Specific application of object access EVENT REPORT services for personal health devices

7.4.1 General

The EVENT REPORT service is the primary mechanism for the agent to report both measurement and configuration data. Event reports in this standard are a property of the MDS and scanner objects. These specific event reports can have various forms and properties, as defined in 7.4.2 through 7.4.7.

7.4.2 Confirmed and unconfirmed event reports

The sender of an event report may optionally require a confirmation from the receiver.

7.4.3 Configuration event report

7.4.3.1 General

Subclauses 7.4.3.2 through 7.4.3.4 describe configurations, configuration event reports, and device specializations used to describe the objects in the agent.

7.4.3.2 Agent device configuration

The set of objects and attributes that exists in an agent denotes the agent device configuration and is associated with a Dev-Configuration-Id value (see Table 2). In case an agent owns multiple device configurations, the assigned Dev-Configuration-Id values shall be locally unique. During the lifetime of an association, the configuration of an agent shall remain fixed, that is, the set of objects shall remain fixed. However, the agent may add new attributes to an object or change attribute values as described in 7.4.3.3. An agent that requires a different configuration shall release the association and establish a new association with the desired configuration.

7.4.3.3 Configuration event report

The configuration that the agent wants to use for the duration of the association to a manager is indicated by using the Dev-Configuration-Id value for the dev-config-id field in Association Request message. If the manager does not already know the agent's device configuration (e.g., based on a previous association phase), the manager asks for the agent's device configuration. The agent transfers its configuration to the manager using a configuration event report. The report describes all the objects of the agent's device configuration along with the associated Dev-Configuration-Id value. For the duration of the association, the agent's configuration is fixed with respect to the number of objects. In case the agent intends to use a different configuration or wants to change the existing configuration by adding or removing objects, the agent shall release the association and re-associate with a new configuration.

For each object, the configuration also contains attributes used by the object. Typically, the infrequently changing attributes are included within the configuration report and dynamic values, such as measurements, are sent in later measurement event reports. The agent may add new attributes to an object or change attribute values during the association without sending a new configuration. Adding new attributes can only be achieved using a variable format event report (see 7.4.5 for details on event report formats). Changing attribute values may use variable, fixed, or grouped event reports depending on the configuration.

Changes to an existing configuration, whether extended or standard, are effective only for the duration of that association and are not considered persistent changes to the configuration. Therefore, the Dev-Configuration-Id represents the configuration as agreed upon at configuration time. In subsequent associations when a previously used Dev-Configuration-Id is specified, the configuration being referenced does not include any changes made during a prior association. Persistent changes to a configuration shall be made only by re-associating and specifying a different Dev-Configuration-Id and the new configuration desired at configuration time.

A manager uses the configuration information to create an equivalent model of the agent's information. This information is then updated by the agent as measurements are collected.

7.4.3.4 Device specializations

The ISO/IEEE 11073-104zz device specializations define mandatory objects and attributes that shall exist within an agent's configuration. Furthermore, each of the specializations defines mandatory elements (e.g., including mandatory actions and methods) of the service and communication models, which have to be supported by an agent following that specialization.

7.4.3.5 Types of configuration

To reduce transmission message sizes, this standard introduces the ability to inform the manager of the agent's configuration in an efficient fashion. There are two types of configuration: standard and extended.

7.4.3.5.1 Standard configuration

A standard configuration is one that is specified in one of the ISO/IEEE 11073-104zz specializations and that has a Dev-Configuration-Id value assigned from the range between standard-config-start and standard-config-end, inclusive. That range is further subdivided by reserving 100 IDs for each ISO/IEEE 11073-104zz specialization in the range from zz × 100 to zz × 100 + 99, inclusive. For example, the range 1500–1599 is reserved for IEEE Std 11073-10415 [B4]. All unused values in the standard range are reserved for future use. A manager that supports one of the ISO/IEEE 11073-104zz device specializations knows all the standard device configurations specified in that particular specialization. Every time an agent requests to associate with that manager using a Dev-Configuration-Id value of a standard configuration, the manager can accept the association without asking for the agent's configuration since it is already known. After successful association, both manager and agent enter the Operating mode.

It is important to note that standard configuration devices are required to send their configuration, if requested. This requirement covers a case where an agent associates with a manager that does not have preconfigured knowledge of the standard configuration (e.g., the manager is version 1.0 and the device specialization is version 2.0 or greater). How well the manager is able to utilize the configuration depends on the manager's implementation.

If an agent uses a Dev-Configuration-Id value assigned to a standard configuration, it shall also fulfill all additional mandatory elements (e.g., including mandatory actions and methods) of the service and communication models as defined in the corresponding device specialization.

7.4.3.5.2 Extended configuration

In extended configurations, the agent's configuration is not standard; it might have a different set of objects, different attributes present, and/or different attribute values. An agent implementing extended configuration(s) shall select a unique Dev-Configuration-Id value from the range between extended-config-start and extended-config-end, inclusive for each extended configuration. At association time, the agent sends the Dev-Configuration-Id to identify the agent's selected configuration for the duration of the association. If the manager already understands that configuration either because it was preloaded via an installation program or the agent previously associated with the manager, then the manager shall respond with the configuration accepted response, and no further configuration information needs to be sent. However, if the manager does not know the agent's configuration, the manager shall respond with an accepted-unknown-config response, and the agent shall transmit its configuration information by sending a configuration event report. See 8.7 and 8.8 for full details on associating and configuring procedures. Once the manager has the configuration, the agent may transmit measurement data. To save association time, the same Dev-Configuration-Id should be used by an agent for subsequent associations provided the device configuration remains the same.

Unlike standard configurations, two agents with the same extended Dev-Configuration-Id do not necessarily represent the same configuration. A manager shall differentiate extended configurations on a per-agent basis. An agent's System-Id may be used to differentiate extended configurations since System-Id is mandatory, required to be unique, and sent during association; however, other techniques such as manufacturer/model/serial number may be used instead as long as they do not lead a manager to use an incorrect configuration for an agent.

In principle, an agent having an extended configuration supports zero, one, or multiple device specializations as defined in the ISO/IEEE 11073-104zz specifications. In case it supports one or more device specializations, it shall implement all mandatory and a valid choice of conditional items (including objects, attributes, actions, and methods) specified in the respective specializations.

7.4.4 Agent- and manager-initiated measurement data transmission

Agent-initiated measurement data transmission is sent by the agent, for example, as a result of a new measurement that is taken.

Manager-initiated measurement data transmission is explicitly requested by the manager by issuing a command (i.e., MDS-Data-Request) to instruct the agent to start or stop sending measurement data. Dependent on the capability of the agent, the time period while this reporting mode is active is configurable (e.g., fixed period or always while associated).

A manager shall support receipt of both agent- and manager-initiated measurement data transmission from an agent. An agent may support generation of either one or both agent- and manager-initiated measurement data transmission.

In both agent- and manager-initiated measurement data transmission, event reports are used to carry the measurement data.

7.4.5 Variable, fixed, and grouped format event reports

Event reporting can take on three styles: variable format, fixed format, or grouped format. Figure 7 shows the relationship between each of the message formats.

The variable format event report explicitly defines each reported attribute by including the attribute identification field, the value length, and the value in the message. This approach provides flexibility for including a different set of attributes per event report but at the expense of message overhead.

The fixed format event report is optimized by defining the message format in the Attribute-Value-Map of the object in a previous configuration message before transfer commences. When an agent transmits data in a fixed format event report, it shall report the object handle and the attribute values in the same order and size as specified in the Attribute-Value-Map. In this way, the overhead of sending attribute identification and length in each event report is avoided. On receipt of a fixed format event report, the manager uses the object handle to retrieve the Attribute-Value-Map previously given at configuration time to know how to extract the data.

The grouped format event report is further optimized by defining an event report's message format, containing one or more objects, in the scanner object's Handle-Attr-Val-Map in a separate configuration message before transfer commences. When an agent transmits data in a grouped format event report, it shall report the scanner object's handle along with the scanned objects' attribute values in the same order and size as specified in the Handle-Attr-Val-Map. In this way, the overhead of sending the scanned object's handles, their attribute identification, and data lengths in each event report is avoided. On receipt of a grouped format event report, the manager uses the scanner object's handle to retrieve the Handle-Attr-Val-Map previously given at configuration time to know how to extract the data.

A manager shall support variable format, fixed format, and grouped format event reports. An agent may support any or all of variable format, fixed format, and grouped format event reports. The manager learns which format(s) the agent might use by inspecting the Attribute-Value-Map of objects defined in the MDS-Configuration-Event from the agent or by inspecting the Handle-Attr-Val-Map attribute for scanner objects defined in the MDS-Configuration-Event.

7.4.6 Single-person and multiple-person event reports

Agents designed to operate in an environment where data may be collected from multiple people may use the multiple-person event report to transmit all the data from all the people in a single event. Where the functionality is not required, the agent may use the single-person event report for reduced overhead.

A manager shall support both single-person and multiple-person event reports. An agent may support either one or both single-person and multiple-person event reports. Subclause A.11.5 describes the formats for single-person and multiple-person event reports.

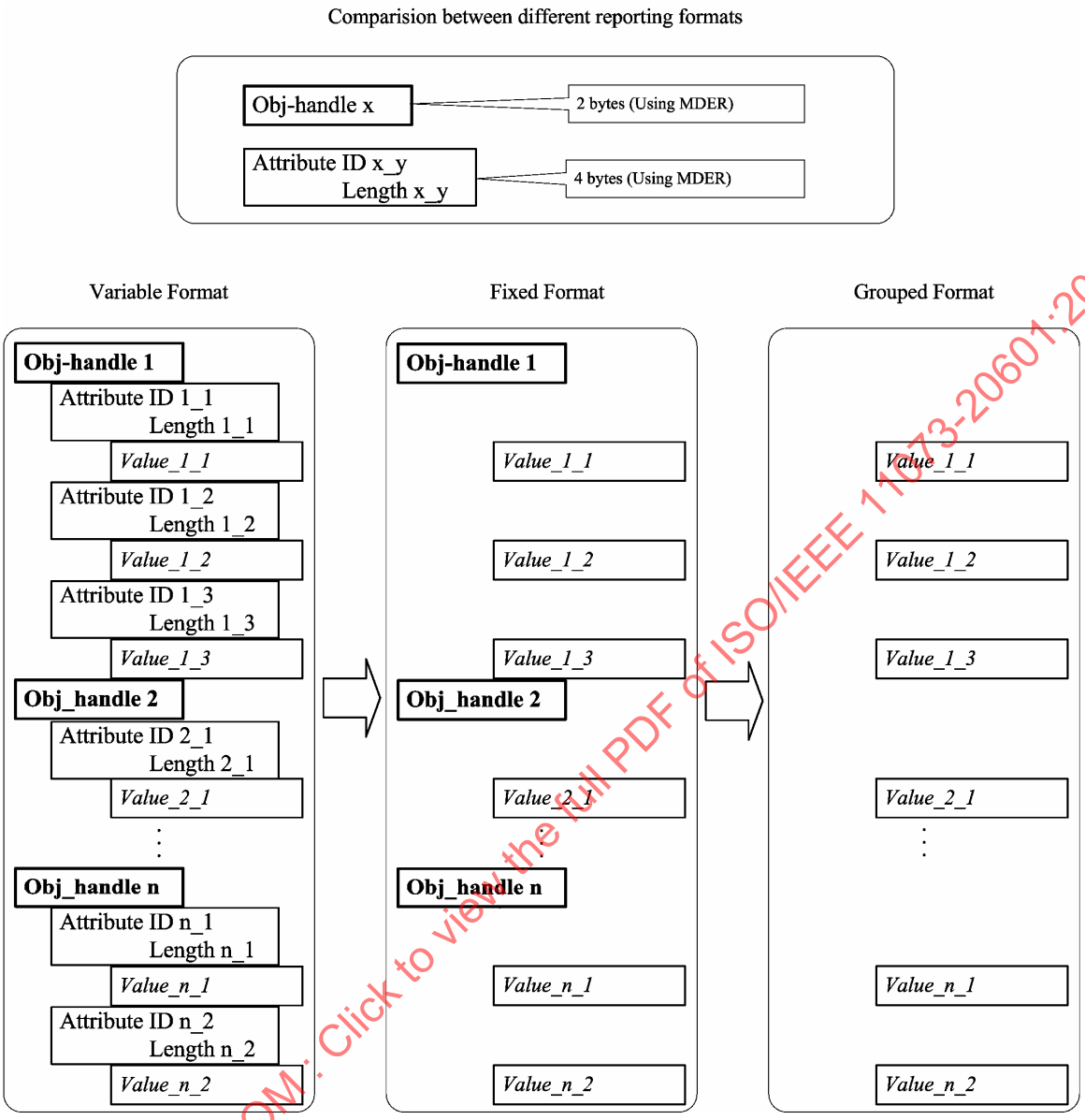


Figure 7 — Variable format, fixed format, and grouped format relationships

7.4.7 Temporarily stored measurements

An agent may optionally store a small number of measurements in local memory while it is not connected to a manager system (i.e., temporarily stored measurements). When the agent can subsequently establish a connection with the manager, all previously stored measurements are transferred to the manager.

NOTE—A typical example for temporarily stored measurements is a weighing scale: new measurements are performed infrequently. The scale is not connected to a manager, and it powers down after the measurement rather than waiting indefinitely for the manager and using up energy.

For the support of temporarily stored measurements, the following behavior shall apply for the agent system:

— Only metric-derived objects that are not RT-SAs (e.g., numeric and enumeration objects) are supported as temporarily stored measurements.

— The use of time stamp attributes (i.e., Date-and-Time, Relative-Time, or HiRes-Relative-Time) is required for temporarily stored measurements.

— The agent shall not send temporarily stored measurements if the time stamp information is known to be inaccurate (e.g., if the time base used to time stamp the values has changed between measurements by an amount significant for the type of measurement) unless it includes the appropriate Date-and-Time-Adjustment at the beginning of the event report.

— Temporarily stored measurements are included in any of the defined event report mechanisms (manager- or agent-initiated; grouped, fixed, or variable format; and single- or multiple-person).

— After transmitting the temporarily stored measurements to the manager, the agent should delete the stored measurements from its local memory. The agent should ensure ownership of the measurements is successfully transferred to the manager by using confirmed event reports.

— To limit the amount of data transported by this mechanism, the agent shall provide no more than 25 temporarily stored measurements in any one event report. If storage of more than 25 measurements is required, the PM-store mechanism should be used for archiving measurements.

8. Communication model

8.1 General

Generally speaking, the expected topology is one or more agents communicating over point-to-point connections to a manager. If a manager wants to support multiple and simultaneous agents (e.g., using a Bluetooth piconet), then the manager shall be capable of handling multiple connection indications and separate associations from each of these agents.

Any agent that supports multiple modalities (device specializations) may choose to generate a single connection and association to a manager or to generate multiple connection indications and associations (e.g., one for each modality) to a manager. However, if an agent chooses to implement multiple connection indications and associations, object instances in the different associations shall be completely independent just as if the associations were implemented by different devices. As an example, the MDS object for each connection indication and association must act as separate, independent agents.

8.2 System context

The communication profile defined in this standard takes into account the specific requirements of personal health agents and managers that are typically used in mobile environments or a person's home. The following assumptions are made regarding the services and features that shall be provided by the transport layers. Additionally, the context of the system outside of this communications profile (i.e., the other non-personal health device/supporting application layer functionality) and its relationship to the assumptions of the transport layers are also covered.

This standard assumes that transport technologies are feature rich and takes a generic view of transport technologies to allow their features to be used, leveraged, and exploited natively. If the transport is not natively intelligent, then a "shim" is added to meet the required characteristics.

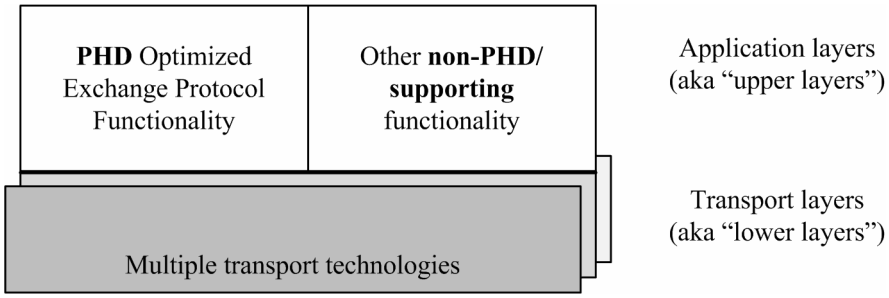


Figure 8 —System context

This standard utilizes the concept of a "type" to group and differentiate the services offered by available transport technologies that have been profiled for use by the ISO/IEEE 11073 family of standards. Specifically, the ISO/IEEE 11073 family of standards recognizes the following transport profile types:

— Type 1: Transport profiles that contain both "reliable" and "best-effort" transport services, where there shall be one or more virtual channels of reliable transport services and zero or more virtual channels of best-effort transport services

— Type 2: Transport profiles that contain only a unidirectional transport service

— Type 3: Transport profiles that contain only a best-effort transport service, where there shall be one or more virtual channels of best-effort transport services

The reason the transport profile types are significant is that the different transport services offered by the transport profile types have an effect on the implementation of some upper layer functionality. In particular, they have an effect on the implementation of this standard's confirmed service mechanism. This standard is defined for use only with Type 1 transport profiles.

For a more complete description of the various transport profile types and how they interact with the confirmed and unconfirmed service mechanisms, refer to Annex D.

8.3 Communications characteristics

8.3.1 General

For this standard, Type 1 transport profiles shall be used.

For this standard, each device shall support a primary virtual channel. A primary virtual channel shall be a reliable virtual channel (i.e., a reliable transport service) from the Type 1 transport profile.

The primary virtual channel shall be used for the following:

— All messages related to the association procedure

  — aare, aarq, rlre, rlrq, abrt

— All messages related to the confirmed service mechanism

- prst.ro**iv**-cmip-confirmed-**action**, prst.ro**iv**-cmip-confirmed-**event-report**, prst.ro**iv**-cmip-**get**, prst.ro**iv**-cmip-confirmed-**set**
- prst.ro**rs**-cmip-confirmed-**action**, prst.ro**rs**-cmip-confirmed-**event-report**, prst.ro**rs**-cmip-**get**, prst.ro**rs**-cmip-confirmed-**set**

— All messages related to fault or abnormal conditions

- roer, rorj

For this standard, each device may support one or more secondary virtual channels. Each secondary virtual channel may be either a reliable virtual channel or a best-effort virtual channel from the Type 1 transport profile.

The primary virtual channel or any secondary channel(s) may be used for messages related to the unconfirmed service mechanism.

— prst.ro**iv**-cmip-**action**, prst.ro**iv**-cmip-**event-report**, prst.ro**iv**-cmip-**set**

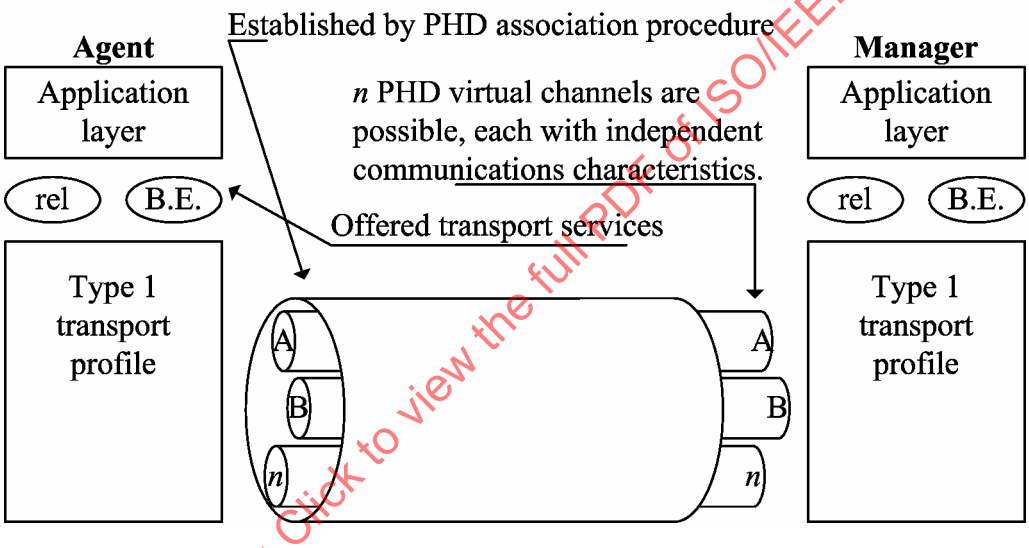


Figure 9 —General communications model

In general, the term *metadata* means data about data. In the context of IEEE 11073-20601 communications characteristics, metadata is used to mean supporting information or data relating to the application protocol data unit (APDU). Examples include the following:

— Transport-technology-specific address for the delivery of a given APDU to a given agent or manager

— Reliability and/or latency needs of a given APDU

— Size or length of a given APDU

Some metadata describe communication characteristics that are represented as a single value that encompasses a wide range of possible values. With respect to the general metadata examples above, some specific examples are as follows:

— APDU-metadata.address (for an USB end point) = 1 – 1023

— APDU-metadata.address (for an IPv4 network) = 0.0.0.0 – 255.255.255.255

— APDU-metadata.size = 1 – 64512

Other metadata describe communication characteristics that are represented as a single value but that only have a few discrete, possible values. With respect to the general metadata examples above, some specific examples are as follows:

— APDU-metadata.latency = (10ms | 100ms | 1sec | 10sec)

— APDU-metadata.reliability = (high | medium | low)

— APDU-metadata.bandwidth = (100bps | 1Kbps | 10Kbps | 100Kbps | 1Mbps)

The following subclauses describe the common characteristics (see 8.3.2) and the unique characteristics of reliable (see 8.3.3) and best-effort (see 8.3.4) of the virtual channels as applied to this standard.

8.3.2 Common communications characteristics

A number of common communications characteristics are applicable to both reliable and best-effort communications:

a) An APDU may be processed in any manner (e.g., part by part as the APDU arrives or as a complete buffered APDU in memory), but the APDU shall be processed so that its effects are as an atomic transaction.

b) APDUs may be segmented and reassembled during transport, or they may be sent as a complete unit.

c) APDUs, in the agent-to-manager direction, shall be no larger than 63K (64 512) bytes in size. Specific device specializations or implementations may evaluate the messages exchanged to determine a specific implementation size for a manager receive buffer that is smaller than the maximum agent-to-manager APDU size. If a manager receives a larger APDU, it shall reply with an error (roer) code of protocol-violation.

d) APDUs, in the manager-to-agent direction, shall be no larger than 8K (8192) bytes in size. Specific device specializations or implementations may evaluate the messages exchanged to determine a specific implementation size for an agent receive buffer that is smaller than the maximum manager-to-agent APDU size. If an agent receives a larger APDU, it shall reply with an error (roer) code of protocol-violation.

e) The overall length of the APDU shall be passed to and from the communications layers as metadata.

f) The communications layer shall indicate the overall length of the APDU to its peer communications layer.

8.3.3 Reliable communications characteristics

For a communications technology/method to be considered reliable and usable by the Optimized Exchange Protocol, the following characteristics apply:

g) APDUs shall be received in the order they are sent.

h) APDUs shall be free of detectable errors.

i) APDUs shall not be duplicated.

j) APDUs shall not be missing.

k) APDUs are generally sent in an expeditious manner, but may be delayed due to retries.

l) The communications layers should provide a mechanism to indicate to the application layer when a complete APDU has been received.

m) The communications layers shall provide a mechanism to indicate to the application layer when a connection path between an agent and a manager is established.

n) The communications layers should provide a mechanism to indicate to the application layer when a connection is terminated or disconnected.

o) The communications layers shall provide a mechanism to indicate to the application layer when it is unable to send an APDU.

p) Flow control between the sending and receiving application shall be supported for complete APDUs. The lower layers may implement flow control for smaller subsets of the APDU.

8.3.4 Best-effort communications characteristics

When a communications technology does not meet the criterion of a reliable communications channel as described above, it is termed best-effort by the Optimized Exchange Protocol. The following characteristics are typical of a best-effort channel:

a) An APDU may not be delivered in the order in which it was sent. It is possible for the communication channel itself, independent of the operation of a personal health device transmitter, to misorder packets.

b) An APDU may be lost or duplicated.

c) APDUs may arrive at a rate that causes buffer exhaustion at the receiver.

8.4 State machines

8.4.1 Agent state machine

Figure 10 shows an overview of the agent state machine.

The detailed agent state table is described in Annex E.2.

Table 19 provides a description of each of the states.

Table 20 provides a description of each of the state transitions.

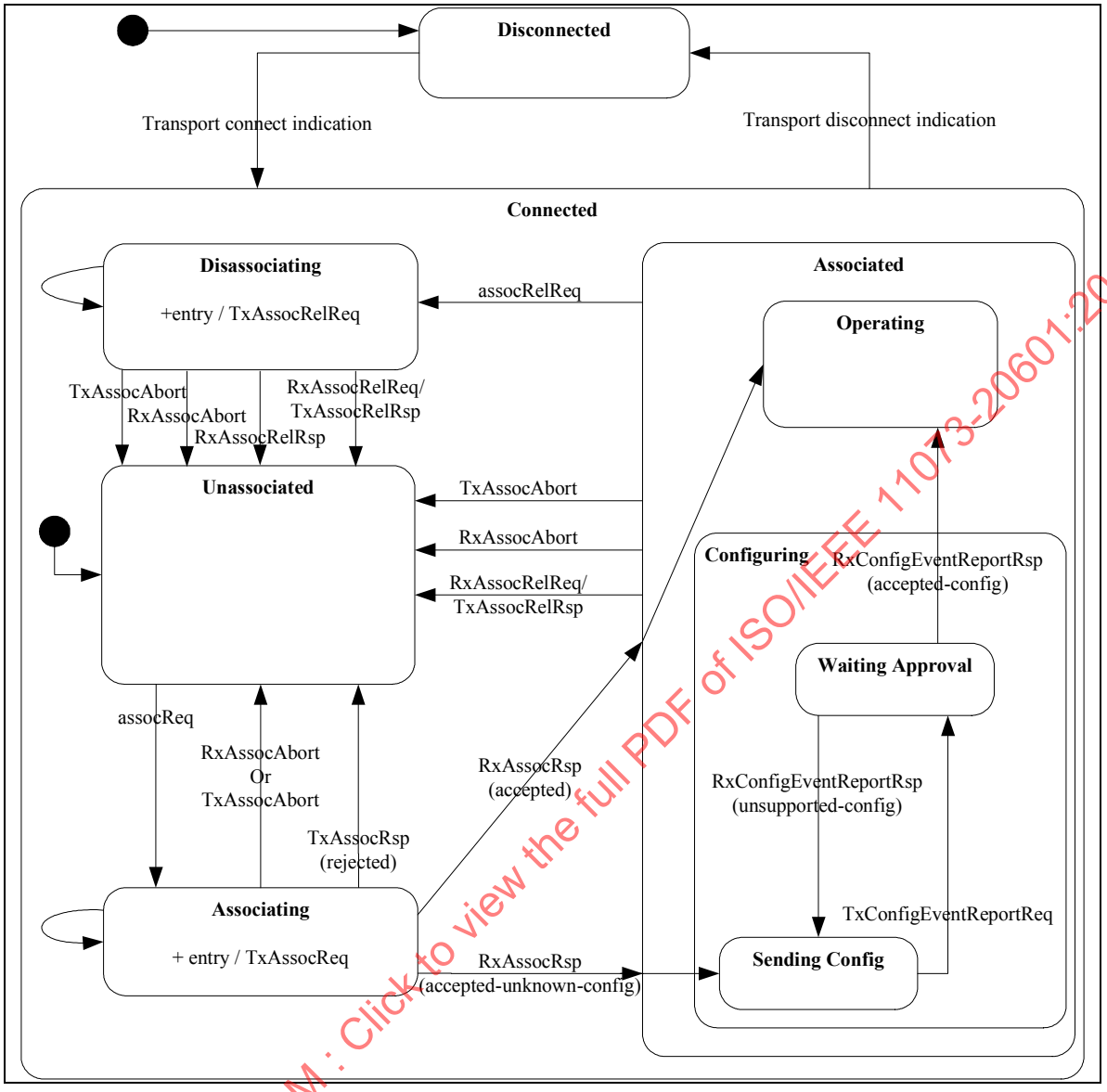


Figure 10 —Agent state machine diagram

Table 19—Agent state description

State	Description
Disconnected	When an agent initially powers on, it is assumed to start in the Disconnected state, which indicates that a transport connection between the agent and manager was not established. After a transport connection is established, it is possible to return to the Disconnected state if the transport connection is intentionally terminated or unintentionally disconnected.
Connected	When a transport connection is established, the agent receives a Transport Connection Indication from the transport layer, causing a transition into the Connected state (see 8.4.3). The agent remains in the Connected state as long as there is a transport connection established. Initially, the agent starts in the Unassociated state, a substate of the Connected state.

Table 19—Agent state description

State	Description
Unassociated	The agent is in the Unassociated state whenever it does not have an application layer association with a manager. This situation can occur due to any of the following: — A new connection was just established. — The manager rejects an Association Request. — Either party releases or aborts an active association at any time while connected. The agent remains in the Unassociated state until the agent determines that it should begin associating with the manager.
Associating	Whenever the agent determines it should create an association, the agent moves to the Associating state and sends an Association Request to the manager (see 8.6). If an association fails, but alternative association parameters are possible, the agent may attempt to associate with each new set of association parameters. In the case of timeout, the agent shall attempt to associate up to the maximum retry count is reached or association is successful.
Associated	When the manager determines that the agent and manager share common versions and protocols, it sends an Association Response with an "accepted" parameter (see 8.7.3.3) to the agent. When the agent receives this message, it moves to the Associated state. It remains in this state until the agent sends or receives a release or abort request for the association. The initial substate when entering the Associated state depends on whether the manager responded to the Association Request with an indication that the agent's configuration is recognized or not.
Operating	When a manager recognizes an agent's configuration, it informs the agent with an Association Response with an "accepted" parameter to cause the agent to move into the Operating state. Alternatively, if the configuration is not recognized, the configuration is transferred. If the configuration is accepted, the agent enters the Operating state. See 8.9 for a description of possible procedures while in the Operating state.
Configuring	When the manager does not recognize the agent's configuration, it informs the agent by sending an Association Response with an "accepted-unknown-config" parameter to indicate that the association was accepted but that the configuration needs to be transmitted. The agent remains in the Configuring state until the agent transfers the configuration information and the manager acknowledges the configuration (see 8.7.6).
Disassociating	Whenever the agent determines it should release the current association, the agent moves to the Disassociating state and sends an Association Release Request to the manager (see 8.10). In cases of a timeout, the agent sends an abort request and moves to the unassociated state.

Table 20—Agent state transition description

Transition	Description
Transport Connection Indication	The Transport Connection Indication transition occurs whenever the transport (or a supporting shim layer) indicates that a connection has been established.
assocReq	Whenever the agent determines that it wishes to attempt associating with the manager, it transitions to the Associating state.
RxAssocRsp(accepted or accepted-unknown-config)	As the agent attempts to associate with the manager, it sends an Association Request message (or multiple in timeout conditions) and awaits an Association Response from the manager. When it receives an approval of association, the agent transitions to the Associated state.
RxAssocRsp(rejected)	If the manager determines that it is not able to associate with the agent after receiving an Association Request, it sends an Association Response with an AssociateResult code of rejected, either permanently or temporarily, to cause the agent to transition back to the Unassociated state.
RxAssocRelReq/TxAssocRelRsp	When an agent is associated with a manager and receives an Association Release Request, the agent responds and transitions to the Unassociated state.

Table 20—Agent state transition description

Transition	Description
RxAssocAbort or TxAssocAbort	Any time the agent and manager are associating, associated, or disassociating, the agent can either send or receive an Association Abort message. When this event occurs, the agent transfers from its current state into the Unassociated state. If the agent is associated, it can send an Association Abort message to inform the manager that a serious failure has occurred. This message should be a last resort with a preference toward sending an Association Release Request to move to the Unassociated state gracefully. If the agent receives an Association Abort message, it does not need to respond since this message is received only when the manager is aborting (e.g., a crash).
assocRelReq	When an agent decides to stop an association, it transitions to the Disassociating state and sends an Association Release Request. This transition is used during a normal shutdown sequence by sending a ReleaseRequestReason of normal or, if the agent's configuration has changed and requires the agent to release the association, the agent uses the ReleaseRequestReason of configuration-changed. Either way, the next time the agent associates, it indicates the configuration to use in the Association Request, and the manager determines whether it knows about the configuration.
RxAssocRelRsp	This transition indicates that the request to release the current association has been granted. In the case where the agent sent the Association Release Request, this indicates the agent has received an Association Release Response indicating that termination is approved by the manager.
Transport Disconnect Indication	At any point, the agent or manager can terminate the transport connection, or the connection may be lost due to fault conditions. When the indication that the transport has been disconnected is received, the agent transitions to the Disconnected state.

8.4.2 Manager state machine

An overview of the manager state machine is shown in Figure 11. The majority of the states and transitions are symmetric with the items described for an agent in Table 19 and Table 20. The key differences are as follows:

— The manager shall wait in the Waiting for Config state for at least TO_{config} seconds before sending an Association Release Request or Association Abort message.

— If the manager does not accept the configuration, it shall send a configuration response with an unsupported-config result.

— If the manager accepts the configuration, it shall send a configuration response with an accepted-config result.

The detailed manager state table is described in Annex E.3.

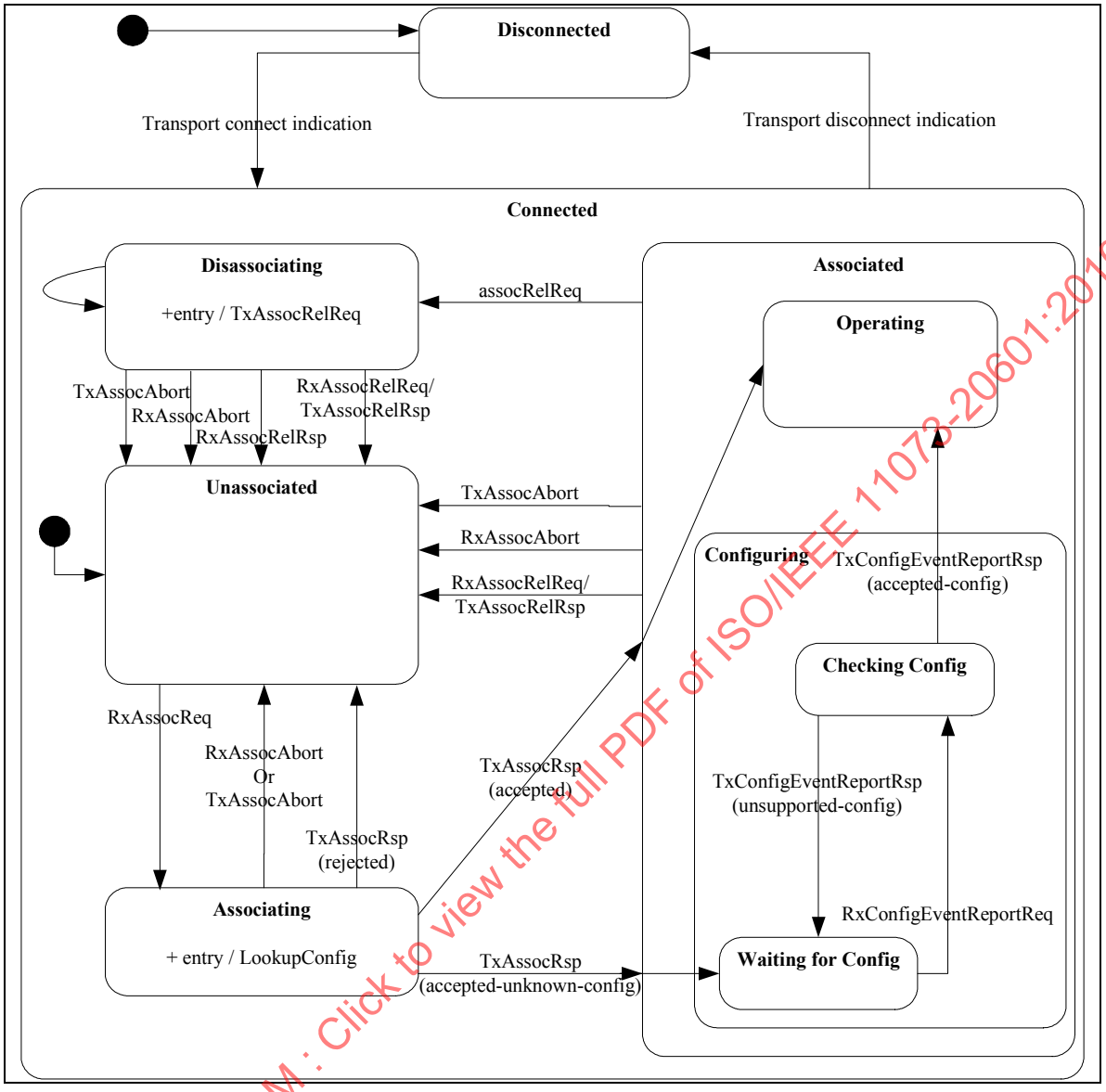


Figure 11 —Manager state machine diagram

8.4.3 Timeout variables

There are a few places in the personal health device protocol where timeouts are used. There are both retry timeout periods and retry counts. To ease long-term document management and facilitate doing electronic "searches" on the timeout values, the specific numerical values have been factored out of the body of this standard and replaced with specific timeout variables. The mapping of the timeouts to numerical values is in Table 21.

Table 21 —Timeout variables

	Communications service	Timeout		Sub-clause
		Vari-able	Value	
Associating procedure				
	Association	TO_{assoc}	10 s (and RC_{assoc} = 3)	8.7.5
	Configuration	TO_{config}	10 s	8.8.5
	Association Release	$TO_{release}$	3 s	8.10.5
Operating procedure				
MDS object	**Confirm action**	TO_{ca}	3 s	8.9.5.2
	Confirm event report	$TO_{cer-mds}$	MDS.Confirm-Timeout	8.9.5.3
	Get	TO_{get}	3 s	8.9.5.4
	Confirm set	TO_{cs}	3 s	8.9.5.5
	<inter-service timeout>	TO_{sp-mds}	3 s	8.9.5.6
PM-store object	**Confirm action**	TO_{ca}	3 s	8.9.5.2
	Confirm event report	$TO_{cer-pms}$	Segm.Confirm-Timeout	8.9.5.3
	Get	TO_{get}	3 s	8.9.5.4
	Confirm set	TO_{cs}	3 s	8.9.5.5
	<end of Segm timeout>	TO_{sp-pms}	Segm.Transfer-Timeout	8.9.5.6
	Confirm action – SegmClear	$TO_{clr-pms}$	PMS.Clear-Timeout	8.9.5.6
Scanner object	**Confirm set**	TO_{cs}	3 s	8.9.5.5
	Confirm event report	$TO_{cer-scan}$	Scan.Confirm-Timeout	8.9.5.3

8.5 Connected procedure

8.5.1 General

Subclauses 8.5.2 through 8.5.5 describe the entry conditions, the normal procedures, the exit conditions, and any error conditions that can occur for the Connected state in the state diagrams.

8.5.2 Entry conditions

The agent and manager enter the Connected state whenever the transport layer indicates that a connection has been established between the agent and manager. Both the agent and manager receive the connection indication from their own transport layers (i.e., no application layer communication occurs by this time). Upon initial entry into the Connected state, both the agent and manager start in the Unassociated state, a substate of the Connected state.

8.5.3 Normal procedures

As the Connected state has a number of substates, the actual operating conditions are described as part of those substates.

8.5.4 Exit conditions

The agent and manager should exit the Associated state by moving to the Disassociating state, sending an Association Release Request, and waiting for an Association Release Response. The agent and manager shall then have closed the active association and returned to the Unassociated state. This is normal behavior before an agent or manager leaves the Connected state. The transport layer is then responsible for closing the connection.

8.5.5 Error conditions

The transport may disconnect unexpectedly (e.g., a wireless transport may be moved out of range or a cabled interface may be removed prematurely). In these cases, the transport should alert the application layer of the disconnection. The agent and manager shall then be responsible to reset to the Disconnected state. This requirement applies to the Connected state and all substates.

8.6 Unassociated procedure

8.6.1 General

Subclauses 8.6.2 through 8.6.5 describe the entry conditions, the normal procedures, the exit conditions, and any error conditions that can occur for the Unassociated state in the state diagrams.

8.6.2 Entry conditions

The Unassociated state is the default state that is entered whenever an agent or manager is first notified about establishment of a connection. This state is also reentered whenever the agent or manager releases or aborts an association with the peer.

8.6.3 Normal procedures

Normally, the agent does nothing during this state.

The manager waits in this state until it receives an Association Request message.

8.6.4 Exit conditions

Whenever the agent determines that it wishes to attempt associating with the manager, it transitions to the Associating state. The manager transitions when it receives an Association Request message.

8.6.5 Error conditions

A number of error conditions may occur while in the unassociated state. The response to such conditions is either to ignore the condition or to generate an Association Abort message. See Table E.1, state 2 (the Unassociated state), for more information.

8.7 Associating procedure

8.7.1 General

The associating procedure allows the agent and manager to agree on a common data protocol and a common set of operating parameters.

8.7.2 Entry conditions

Both the agent and manager shall remain in the Unassociated state until the agent determines that an association is desirable. At that point, the agent shall enter the Associating state and send an Association Request. The manager shall enter the Associating state when it receives an Association Request from the agent.

8.7.3 Normal procedures

Figure 12 and Figure 13 show sequence diagrams of the associating procedure between an agent and manager. Figure 12 shows the situation where the manager already knows about the agent's configuration due either to a prior connection with the agent or to the fact that the agent has a standard configuration (i.e., a predefined configuration that is specified in a specialization standard). Figure 13 shows the case where the manager does not know the agent's configuration and informs it that the association request is accepted, but that the configuration is unknown.

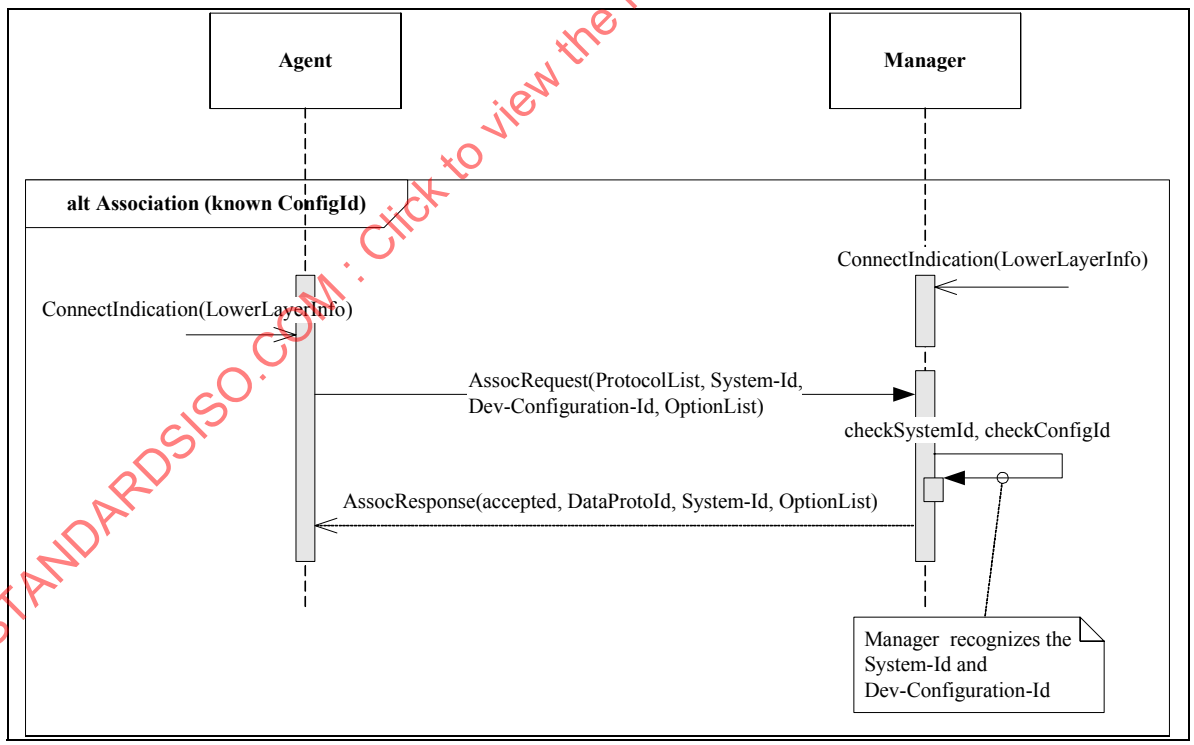


Figure 12 —Association procedure (known configuration)

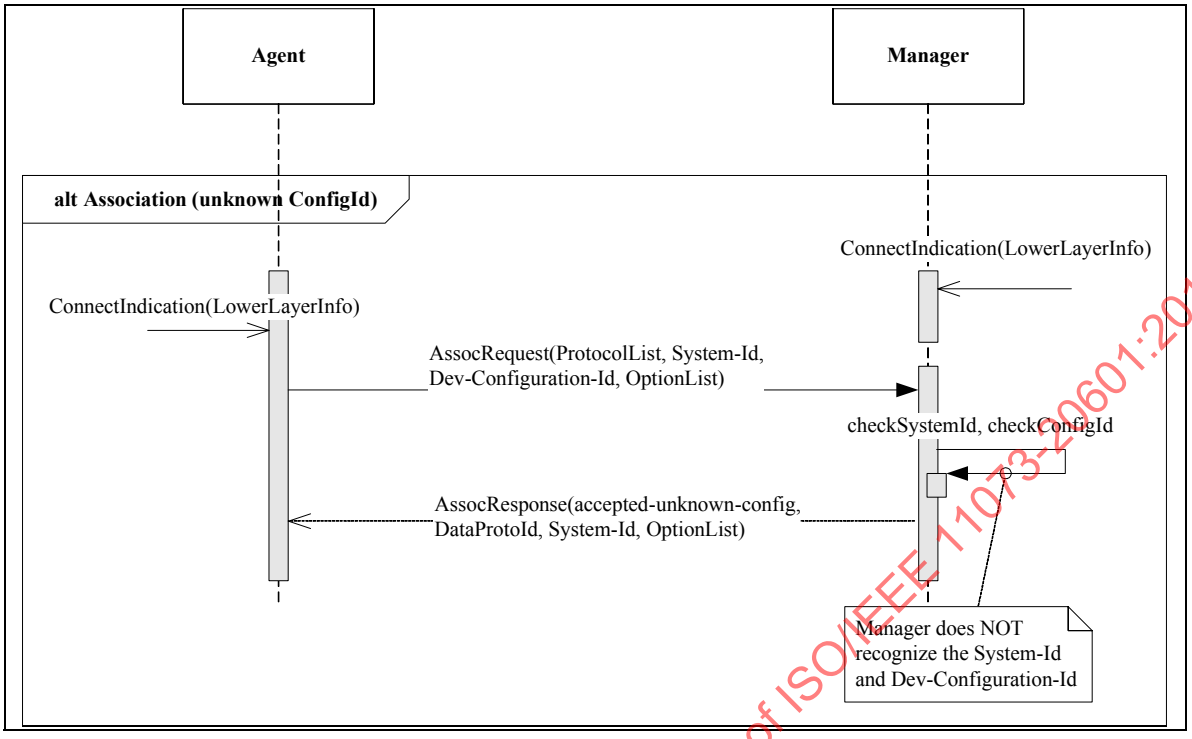


Figure 13 —Association procedure (unknown configuration)

Subclauses 8.7.3.1 and 8.7.3.2 describe the operating conditions for the two different device roles: agent and manager.

8.7.3.1 Agent procedure

8.7.3.1.1 General

When the agent wants to create an association, it shall begin by transitioning to the Associating state and sending an Association Request message to the manager. The AarqApdu definition (see A.8) describes the format of the Association Request message. An example of an Association Request is found in H.2.1.1.

The Association Request message contains the items listed below:

— The version of the association protocol used (assoc-version). This field allows the agent and manager to ensure that they are using the same version of the protocol exchange.

— A list of data protocols that the agent supports (data-proto-list). The agent is allowed to support one or more data protocols for exchanging information. The agent shall order the list of data protocols with the most preferred protocol listed first descending to the least preferred protocol last.

The manager selects the desired protocol and communicates that to the agent.

To allow selection of a data protocol during association, the data-proto-list contains an ID that denotes either that the data protocol is defined by one of the ISO/IEEE 11073 family of standards or that it is manufacturer defined. These options are described in the next two subclauses. Additional codes are available, but reserved, for future extensions.

8.7.3.1.2 Data exchange protocol –defined by this standard

If an agent sets the data-proto-id in A.8 to data-proto-id-20601, then it shall adhere to the abstract syntax definitions found in this standard for data types and message exchange. Further, the data-proto-info field shall be filled in with a PhdAssociationInformation structure, which defines the following information:

— The version of the data exchange protocol.

— The specific DataApdu encoding rule(s) supported by the agent. The agent shall set one or more of the encoding-rules bits.

  — The agent shall always support MDER, i.e., the mder bit of the encoding-rules field shall be set by the agent.

  — The agent may offer other encoding rules, besides MDER, to the manager by setting other bits in the encoding-rules field.

— The version of the nomenclature used.

— A field indicating all functional units and optional features supported by the agent.

— The system type (agent in this case).

— A unique System-Id (see Table 2) of the agent. The EUI-64 format is used to identify the agent. A manager may use this field to determine the identity of the agent with which it is communicating and optionally to implement a simple access restriction policy.

— A dev-config-id, which identifies the current configuration of the agent as described in 7.4.3. For standard configurations, the dev-config-id value shall lie between standard-config-start and standard-config-end, inclusive. For extended configurations, the dev-config-id value shall lie between extended-config-start and extended-config-end, inclusive.

— A data-req-mode-capab, which defines the data request modes supported by the agent (see 8.9.3.3.3).

— An option-list that contains a list of additional attributes the agent wishes to communicate.

8.7.3.1.3 Data exchange protocol – manufacturer defined

Other specifications may use the initial association request to negotiate the use of manufacturer-defined protocols. In this case, the agent sets the data-proto-id in A.8 to data-proto-id-external. To distinguish between many possible manufacturer-defined protocols, the agent uses the ManufSpecAssociation-Information structure to provide a UUID that denotes the specific protocol. The actual behavior of the protocol, beyond the initial association, is outside the scope of the ISO/IEEE 11073 family of standards. The UUID shall be generated according to ITU-T Rec. X.667 (Sept. 2004).

8.7.3.2 Association response

After the agent has sent the Association Request message, the agent shall wait either for an Association Response message from the manager or for a timeout (see 8.7.5 for timeout conditions).

The AareApdu definition (see A.8) describes the format of the Association Response message. An example of an Association Response is found in H.2.1.2. The Association Response message contains the following:

— A result field representing the outcome of the association procedure.

— The version of the common data protocol chosen by the manager if the result field is equal to accepted or accepted-unknown-config.

— The one, and only one, DataApdu encoding rule chosen by the manager if the result field is equal to accepted or accepted-unknown-config.

  — The manager shall always support MDER to enable interoperability.

  — Alternatively, the manager may select one of the other encoding rules, besides MDER, that are offered by the agent.

    NOTE—MDER is always supported by both the agent and the manager. However, if the agent offers additional encoding rules to the manager, it can be concluded that the agent had a valid reason to do so (i.e., the development of additional encoding rule support is not done without a compelling product reason). Thus, if an agent offers additional encoding rules beyond MDER, it is suggested that the manager honor one of the additional encoding rules offered if possible. For example, if an agent offers MDER and packed encoding rules (PER), it is suggested that the manager honor the PER encoding, if possible. If an agent offers MDER and XML encoding rules (XER), it is suggested that the manager honor the XER encoding rules, if possible. If an agent offers MDER, PER, and XER, this standard offers no suggestion as to the preferred encoding rule selection.

— The version of the nomenclature chosen by the manager if the result field is equal to accepted or accepted-unknown-config.

— The system type (manager in this case since the message originated from the manager).

— The unique system ID of the manager. EUI-64 is used to uniquely identify the manager. An agent may use this field to determine whether it is communicating with the intended manager.

— The dev-config-id field shall be manager-config-response in the response.

— The data-req-mode-capab shall be zero in the response.

— A field indicating the common functional units and optional features chosen by the manager if the result field is equal to accepted or accepted-unknown-config.

The result field in the Association Response message indicates the outcome of the request. Possible outcomes are (see AssociateResult in A.8) as follows:

— accepted means the association is accepted and the configuration is known. The agent shall transition to the Operating state (see 8.9 for more detail on the operating procedures).

— accepted-unknown-config means the association is accepted but the agent is required to send its configuration to the manager. When an agent receives a response that the configuration is unknown, it shall transition to the Configuring state and follow the procedures in 8.7.6 to transfer its configuration.

— rejected-unsupported-assoc-version means that the agent and manager do not share a common association version.

— rejected-no-common-protocol means the manager rejects the Association Request because there is no common data protocol found in the DataProtoList shared between the manager and the agent.

— rejected-no-common-parameter means the manager rejects the Association Request because the manager and the agent do not have a common set of operating parameters in the protocol-specific association information (PhdAssociationInformation).

— rejected-unauthorized is used when the manager determines that the agent is not authorized to connect. The method of making the determination is vendor specified.

— rejected-transient is used when the manager cannot accept the association due to transient conditions such as resource limitations.

— rejected-permanent means that the manager is unable to associate with the agent, but no further detail on the reason is available.

— rejected-unknown should be used sparingly and only when the above return codes do not apply.

In all rejected-* conditions, the agent shall transition to the Unassociated state.

8.7.3.3 Manager procedure

When a manager receives an Association Request, it shall compare the protocol and operating parameters with its own and determine whether the agent is compatible with the manager. If the connection is bidirectional, the manager shall report the outcome of this assessment in the result field of an Association Response.

The possible rejection reasons are enumerated in 8.7.3.2. If the manager rejects the association, it shall transition to the Unassociated state.

If the request is not rejected by the manager, the result field in the Association Response message from the manager indicates whether the manager understands the configuration. If the manager recognizes the dev-config-id as a known standard device specialization or as a previous association, the manager shall send an Association Response message with a result field of accepted and transition to the Operating state.

If the manager does not recognize the dev-config-id, the manager shall send an Association Response message with the result field set to accepted-unknown-config and transition to the Configuring state.

When the manager accepts a common protocol, it shall return the preferred common data protocol and common set of operating parameters selected from the list provided in the Association Request in the Association Response.

8.7.4 Exit conditions

The manager exits when it has sent the Association Response. The agent exits the Associating state whenever it receives the Association Response.

8.7.5 Error conditions

The agent shall wait for an Association Response message for a TO_{assoc} (timeout: association procedure) period. If the TO_{assoc} period expires, the agent shall retransmit the Association Request message up to RC_{assoc} (retry count: association procedure) times after the first timeout, with a TO_{assoc} period in between each successive message. If, after this retry sequence, the agent does not successfully receive any Association Response messages, then the agent shall send an Association Abort message to the manager and transition back to the Unassociated state.

If the agent or manager receives an Association Abort message while in the Associating state, it shall transition to the Unassociated state.

8.7.6 Test association

A test association is an association entered into by an agent and manager that frames data exchanges that are intended for test purposes. This standard does not define what these exchanges look like, nor the semantics associated with them, but only the process by which devices enter and exit a test association. Individual device specializations may define standardized test resources, configuration IDs, and processes

that can be used during a test association. The test association may be used for manufacturer-specific testing needs.

Since this standard does not define the semantics of the test association, it also does not define specific mechanisms to ensure that test data are managed properly. However, it is critical that devices provide protection to ensure that test data are not processed by other entities as actual measurement data. In general, only elements that understand the concept of a test association should see measurement data generated by a test association. Implementers should take the following steps:

— Set the test-data bit or the demo-data bit of the MeasurementStatus attribute when generating simulated measurement data. If the MeasurementStatus attribute is not supported, alternative means to flag such data should be used.

— Ensure that local displays and stores of measurement data ignore test or demo data unless they can properly flag such data to the user and can detect entry and exit from a test association. A local component on an agent that does not participate in the IEEE 11073-20601 protocol may not be a good candidate to receive test measurement data.

— Ensure that measurement data that are placed in a PM-store, or other persistent store structure, are never seen outside of a test association. Tagging and/or clearing of persistent stores may be used for this purpose.

— Ensure that devices that display or store test or demo data properly update when events, such as a disconnection, cause the test association to be terminated.

In order for a test association to be formed, both the manager and the agent need to support test associations, and both need to be willing to enter into a test association at a given point in time. A three-step protocol is used to unambiguously enter a test association.

In the first step, the agent passes the manager two bits of information in the fun-units field of the PhdAssociationInformation structure. The fun-unit-havetestcap bit indicates that the agent has testing capabilities that can be used within a test association. The fun-unit-createtestassociation bit is used by the agent to request that the manager establish a test association. The agent shall not set the fun-unit-createtestassociation bit unless it also sets the fun-unit-havetestcap bit. If an agent fills in the PhdAssociationInformation structure with the fun-units-havetestcap bit set, it should not terminate the association due to the receipt of a response with the fun-unit-createtestassociation bit set. This implies that if an agent sets the fun-unit-havetestcap bit and offers more than one configuration in which standardized test capabilities are defined, then the agent should be willing to enter into a test association using any of those configurations.

In the second step of the protocol, the manager signals back to the agent its intent to establish a test association. The manager communicates this information to the agent through the fun-units-createtestassociation bit. The bit is set by the manager to indicate that it has entered into a test association. The manager shall set this bit if, and only if, the fun-units-havetestcap is set in the request from the agent. The manager is not obligated by this standard to enter into a test association even when requested by the agent. The agent shall ignore the fun-units-havetestcap bit in the association reply.

The final step of the test association protocol involves a decision by the agent to either continue with the test association or terminate it. The agent shall not enter into the test association state unless the manager has set the fun-units-createtestassociation bit. The test association shall end whenever the association state machine enters the Unassociated state.

8.8 Configuring procedure

8.8.1 General

The Configuring state occurs when the agent needs to pass configuration information to the manager.

8.8.2 Entry conditions

An Association Response message with an accepted-unknown-config result field shall trigger the agent to enter the Configuring state and send its configuration to the manager. The manager enters the Configuring state immediately after it sends the Association Response with the accepted-unknown-config result.

Note that part of the configuration is also the assignment of values of the Handle attribute to object instances. If the manager knows the agent configuration, it also knows the assigned values of the Handle attribute. This implies that a standard configuration, such as a configuration defined in an ISO/IEEE 11073-104zz device specialization, defines fixed values for the Handle attributes.

8.8.3 Normal procedures

Figure 14 shows the sequence diagram for the configuration procedure. During the configuration procedure, the agent shall transfer the configuration information of all objects that it supports, except the MDS object, as well as all static attributes within the objects. Agents typically have a very static configuration so communicating all static portions during a one-time configuration phase reduces overall communication traffic. New measurement types are not added dynamically, many attributes do not change, and the set of reported object attributes is often the same. A reconfiguration is required only if the agent changes (e.g., as part of an initial setup procedure where specific measurement capabilities may be configured).

The agent performs the configuration procedure using the Confirmed Event Request message with an MDC_NOTI_CONFIG event to send its configuration to the manager. The configuration notification message identifies

— All the objects supported by the agent except the MDS object and

— The set of static attributes for each object.

The attributes include object class nomenclature identification (see 6.3.4.2, 6.3.5.2, and 6.3.6.2), physiological ID (nomenclature code), unit/dimension ID (nomenclature code), optionally strings for labeling, and any other static attributes that might be useful. This information is considered a flat (nonhierarchical) and static containment tree of the agent. The MDS object is excluded from the configuration since the majority of information is dynamic or manufacturer specific. A separate Get MDS Object command provides a mechanism to retrieve this information (see 6.3.2.6.1).

For objects that report on the same attributes each time, the fixed format event report (see 7.4.5) is recommended, and the agent shall send an Attribute-Value-Map describing the message layout. In the case of scanner objects that use the grouped format event reports, the agent shall send the Handle-Attr-Val-Map describing the layout.

If the set of reported object attributes is not fixed, the variable format event report is recommended. In this format, it is possible to communicate the configuration attributes as part of the value updates. In this case, the Attribute-Value-Map is not provided in the configuration event report or is an empty list.

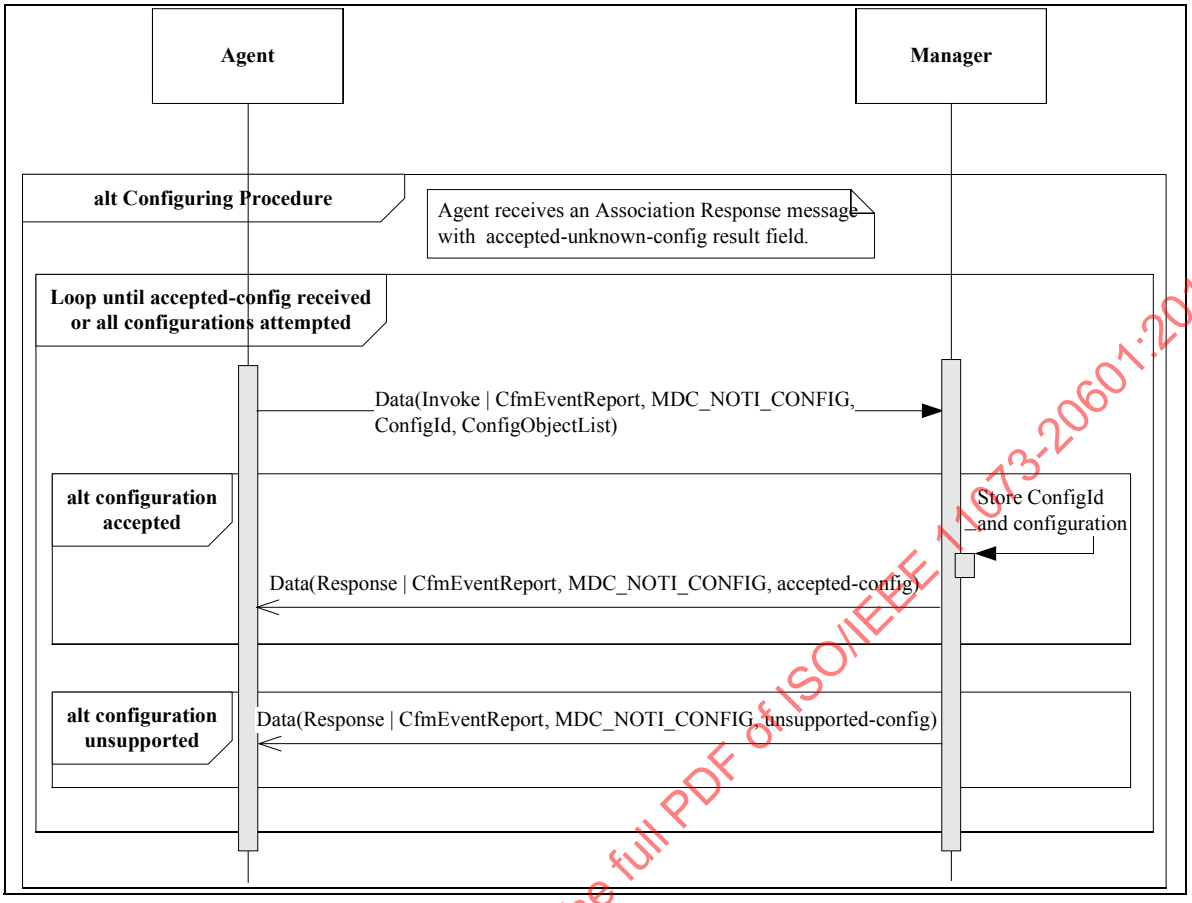


Figure 14 —Configuration procedure

The agent shall use a "Remote Operation Invoke | Confirmed Event Report" data message (see A.10.3 for the initial definition of EventReportArgumentSimple) with an event-type of MDC_NOTI_CONFIG when transferring its configuration (see the ConfigReport in A.11.5 for the remainder of the structure). The manager shall respond with a "Remote Operation Response | Confirmed Event Report" message (see A.10.3 for the definition of EventReportResultSimple) with an event-type of MDC_NOTI_CONFIG filling in the ConfigReportRsp structure. See H.2.2 for an example configuration event request sent by the agent followed by an example response from a manager.

Agents may support more than one configuration. In this case, an agent shall send each of its available configurations beginning with the preferred configuration. If the manager accepts the configuration, it responds with an accepted-config message, and both manager and agent move to the Operating state. If the manager does not accept the configuration, it shall return an unsupported-config response. On receipt of unsupported-config, the agent shall send a further configuration. This process is repeated until the agent has attempted all configurations. Then it shall send an Association Release message with a reason code of no-more-configurations to indicate that it is unable to operate with the manager.

An agent that conforms to one or more device specializations that define standard configurations (i.e., ISO/IEEE 11073-104zz specializations) shall support one or more of the standard configurations and may support one or more extended configurations. For interoperability, this agent shall send the supported standard configurations as a fall back if the extended configurations are unsupported.

If the agent conforms to a standard configuration, it shall use a dev-config-id as defined in the specific ISO/IEEE 11073-104zz device specialization. These standard configuration dev-config-id values are assigned in the range between standard-config-start and standard-config-end, inclusive. When an agent submits a dev-config-id corresponding to a standard configuration, the configuration message need not contain the configuration information and may send an event-type of MDC_NOTI_CONFIG with a standard configuration ID and an empty ConfigObjectList. If the manager does not recognize the standard configuration (e.g., the manager was released prior to the device specialization being released), it shall send a response of standard-config-unknown. The agent may retry the configuration for the standard device by sending the full configuration information.

An agent having a nonstandard configuration shall assign a unique ID to its configuration by generating a value for dev-config-id in the range between extended-config-start and extended-config-end, inclusive.

An agent may use the same value for dev-config-id in future Association Requests with the manager to denote the same configuration of the device. The selected value of dev-config-id shall be reported in the Dev-Configuration-Id attribute of the MDS object.

If the agent changes its configuration so that it can no longer support the old configuration or determines that a new configuration should be used in preference, it shall close any existing association by sending an Association Release message with a reason of configuration-changed. If the new configuration is a new extended configuration, the agent shall assign a new configuration ID. The next time the agent associates, it negotiates with the manager by stepping through each configuration in order of priority as described previously.

8.8.4 Exit conditions

When the manager accepts the preferred configuration, it shall send the accepted-config response to the agent and shall transition to the Operating state. If the manager receives an Association Release Request with a reason of no-more-configurations to indicate that the agent has no further configurations, the manager shall transition to the Unassociated state.

When the agent receives the accepted-config response from the manager, it shall transition to the Operating state. If the agent receives the unsupported-config response from the manager, it shall send the next configuration to the manager until no further configurations are available. Then it shall send an Association Release Request message with a reason of no-more-configurations and enter the Unassociated state.

8.8.5 Error conditions

The agent shall wait for the "Remote Operation Response | Confirmed Event Report | MDC_NOTI_CONFIG" message for a TO_{config} (timeout: configuration procedure) period. If the TO_{config} period expires, the agent shall send an Association Abort message to the manager and transition back to the Unassociated state.

The manager shall wait at least TO_{config} seconds in the Waiting for Configuration state for the configuration information prior to sending an Association Abort message and returning to the Unassociated state.

If the agent or manager receives or sends an Association Abort message at any time, it shall transition to the Unassociated state.

8.9 Operating procedure

8.9.1 General

The communication of health data and status information about the agent occurs during the Operating state.

8.9.2 Entry conditions

The agent and manager enter the Operating state either when the agent's configuration is already known by the manager or after the agent has communicated an acceptable configuration to the manager.

8.9.3 Normal procedures

8.9.3.1 General

Subclauses 8.9.3.2 through 8.9.3.4 describe procedures that can occur when in the Operating state.

8.9.3.2 MDS object attributes

At any time in the Operating or Associated state, the manager may request the MDS object attributes of an agent by sending a data message with the "Remote Operation Invoke | Get" command and a reserved handle value of 0. The agent shall report its implemented MDS object attributes to the manager using a data message with the "Remote Operation Response | Get" response. See H.2.3 for example usages of this set of messages. Agents shall support a Get command that requests all attributes (i.e., the attribute-id-list is empty). Agents may support retrieval of a specific list of attribute IDs.

Figure 15 shows the sequence diagram of the manager requesting the MDS object attributes from an agent.

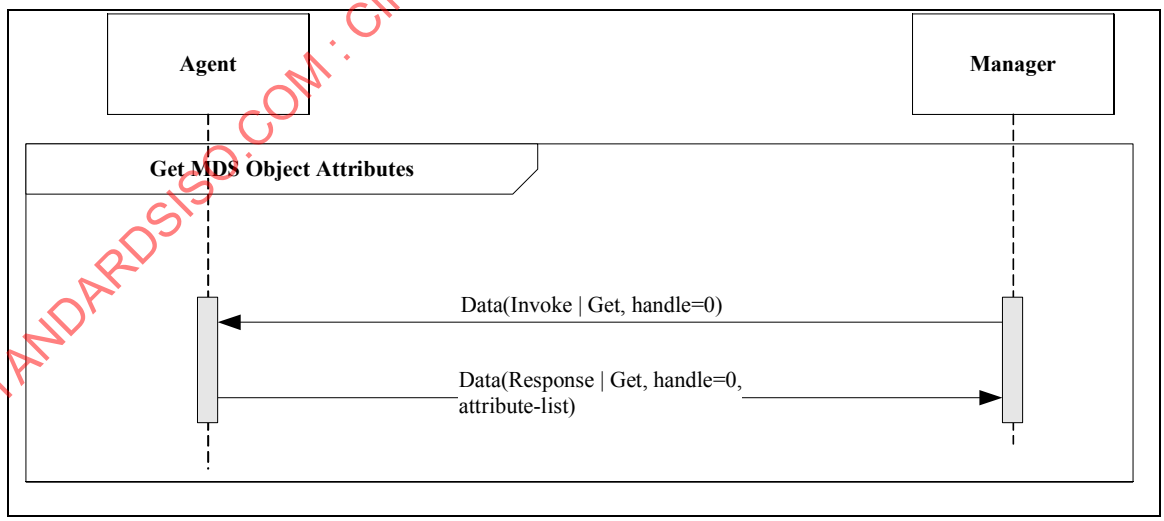


Figure 15 —Get MDS object attributes sequence diagram

8.9.3.3 Measurement data transfer

8.9.3.3.1 General

Measurement data transfer may be initiated by either agent or manager, as introduced in 7.4.4. Agent-initiated transfers would normally be expected from agents that transfer small amounts of infrequent episodic information or require minimal bandwidth. Agents with large amounts of data, frequent data transfer, or streaming data should use manager-initiated transfer. Manager-initiated transfer is preferred in all cases as this approach provides mechanisms to control the data flow. Note that receipt of a request to transfer measurement data is not intended to be a command that an agent perform a measurement, but rather that it shall transfer any measurement data that are available.

In each case, except the single response mode, measurement data transfer is performed using the event report that is confirmed or unconfirmed as selected by the agent.

All variants of the two styles are described in detail in 8.9.3.3.2 through 8.9.3.3.8.

8.9.3.3.2 Agent-initiated measurement data transmission

When an agent supports agent-initiated transmission, it shall indicate that support via the DataReqModeCapab structure or have one or more instances of a scanner object in the agent's configuration.

The agent shall use the EVENT REPORT service (see 7.3) to send a spontaneous measurement to the manager without being requested by the manager first. A DataApdu message in a "Remote Operation Invoke | Event Report" command and one of the MDC_NOTI_SCAN_REPORT_* event-types shall be used for this purpose. If the Confirmed Event Report is used, the manager shall respond with a DataApdu message with the "Remote Operation Response | Confirmed Event Report" response (see Figure 16). If the Unconfirmed Event Report is used, the manager shall not respond.

Scanner objects shall begin with the Operational-State attribute set to disabled on agents with bidirectional communication until the manager enables it. The manager shall set the state of scanner objects to enabled when it wants to receive the data.

For agent-initiated measurement data transmission, the data-req-id field in the Scan Report (MDC_NOTI_SCAN_REPORT_*) shall be set to data-req-id-agent-initiated.

The manager may stop an agent-initiated measurement data transmission from the agent by sending an Association Release Request or Association Abort message to the agent to terminate the association, If the agent uses a scanner object, the manager can disable the scanner by using the SET service on the Operational-State attribute.

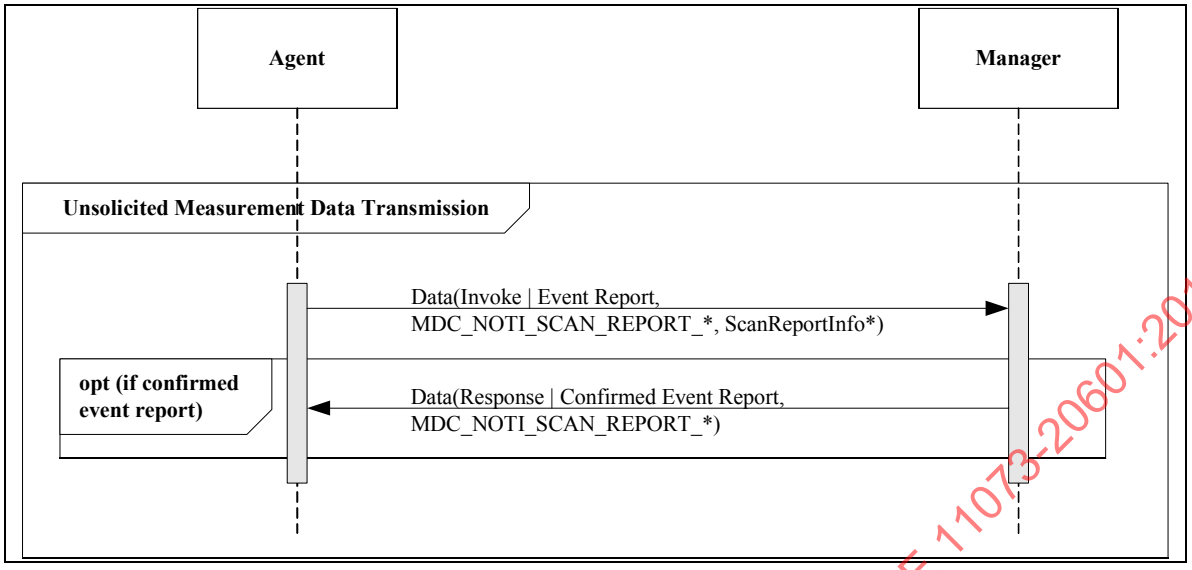


Figure 16 —Agent-initiated measurement data transmission

8.9.3.3.3 Manager-initiated measurement data transmission overview

When an agent supports manager-initiated transmission, it shall indicate what features it supports using the DataReqModeCapab structure. If the agent does not provide any DataReqModeCapab, the manager shall assume it does not support any of the features.

In manager-initiated measurement data transmission, the manager uses the ACTION service (see 7.3) provided by the agent to request measurement data transmission from the agent. When the manager wishes to do this, it shall send a DataApdu, ActionArgumentSimple confirmed request with an MDC_ACT_DATA_REQUEST action type followed by the DataRequest information. This data request may be a start request or a stop request as indicated by the data-req-start-stop bit of the data-req-mode (see A.11.5) or a continuation request as indicated by the data-req-continuation bit.

For a start request, three modes may be used: single response (data-req-mode-single-rsp), time period (data-req-mode-time-period), and no time limit (data-req-mode-time-no-limit). Depending on the mode of the start request, the agent may send one or multiple event reports to the manager. When the manager starts a data mode, it provides a data-req-id that shall be used by the agent in all event reports. If, while a data mode is running, a new start request is received with the same data-req-id, this request shall be treated as taking precedence and the new mode initiated. The agent treats the new start request as if it was a stop followed by a start. Both single response and time period modes have well-defined end points after which the resources supporting the request may be released. The no time limit request does not have a well-defined end point. A manager should issue a stop request when it is no longer interested in a measurement flow, especially for no time limit requests, in order to free up resources on the agent.

For each of these modes one of three different options for the object scope to which the data request refers may be chosen: all data available at the agent (data-req-scope-all), data available at the agent according to a particular object class (data-req-scope-class), and data available at the agent according to specific objects identified by their handles (data-req-scope-handle).

When data-req-scope-all is used, the agent shall consider all objects, except the MDS object, when determining the content of each event report.

When data-req-scope-class, the manager shall use data-req-class to define the class of objects to report. The agent shall consider only the objects described by the given class when generating event reports. Legal

class IDs include MDC_MOC_VMO_METRIC_NU, MDC_MOC_VMO_METRIC_SA_RT, and MDC_MOC_VMO_METRIC_ENUM.

When data-req-scope-handle is sent, the manager shall provide a list of handles in data-req-obj-handle-list. The agent shall consider only the objects described by valid handles in the handle list when generating event reports. Here, the term *valid* refers to all handles associated to a metric- derived object (e.g., numeric, RT-SA, or enumeration) supported by the agent.

A stop request may be used by the manager to stop a time period or no time period measurement data transmission that was started earlier.

When using timed mode, if the manager wants to extend the time that an agent is allowed to send data, the manager shall set the data-req-continuation bit in the mode and set the data-req-time to the amount of time allotted to the agent for continued transmission.

The data-req-id field in the data request is used to differentiate responses from multiple data requests to the same agent (if the agent allows for multiple simultaneous data requests). The manager shall set the value of the data-req-id field to a value in the range from data-req-id-manager-initiated-min to data-req-id-manager-initiated-max, inclusive. The agent shall use the same value of data-req-id in all associated event reports.

Note that the manager may set the value of the data-req-id field to any value within the acceptable range. Then agent shall not rely on the data-req-id field to deduce, for example, the order in which different data requests were generated by the manager.

Streaming agents should use manager-initiated data transmission (or scanner objects) in order to allow the manager to control how it receives the data. Managers should enable streaming agents as soon as possible so agent information is readily available.

The three modes of manager-initiated measurement data transmission are described in 8.9.3.3.4 through 8.9.3.3.6.

8.9.3.3.4 Manager-initiated single response mode

The single response mode allows the manager to request data from the agent and receive it in the response message (see Figure 17). There is no requirement that the agent collect any data (e.g., inflate a blood pressure cuff) to fulfill the response. If the agent does not have data available, it shall return an empty list of data. If the agent has data and the result status is data-req-result-no-error, it shall send a DataResponse message that contains the result status of the request (DataReqResult) as well as the measurement data (ScanReportInfo*). This response message shall complete the measurement data access.

Single response mode does not allow the agent to confirm that the manager receives the measurement data. Where such confirmation is important, the timed command with a timeout value of 0 is used (see 8.9.3.3.5).

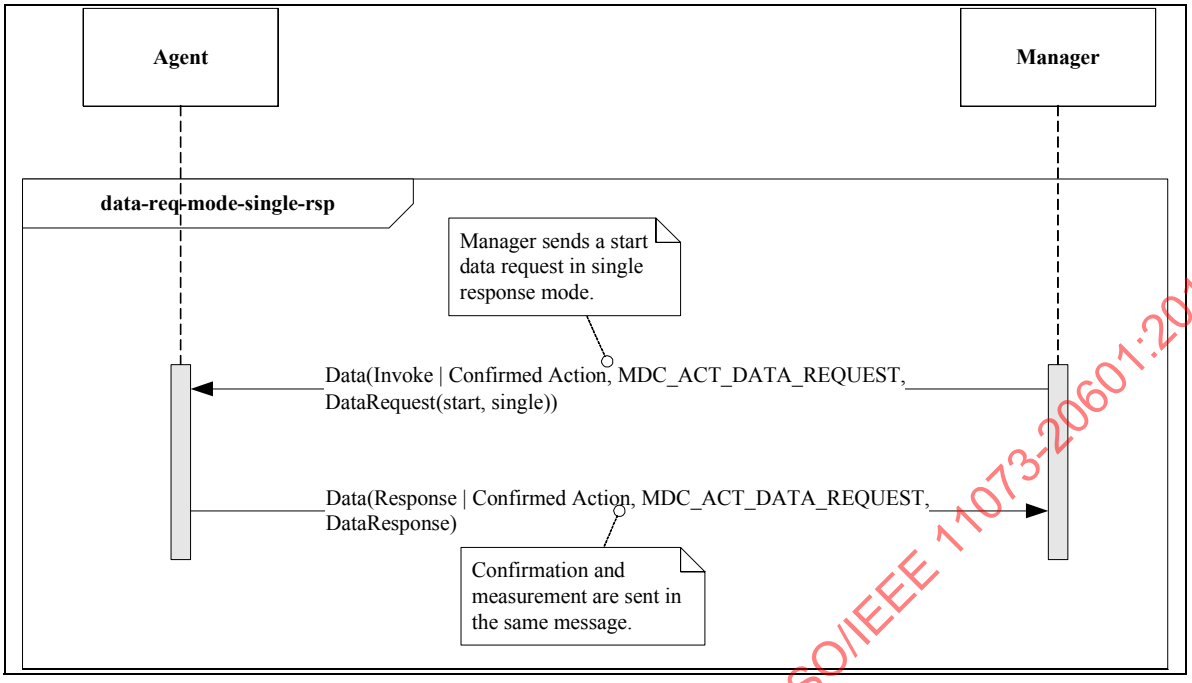


Figure 17 —Manager-initiated measurement data transmission (data-req-mode-single-rsp)

8.9.3.3.5 Manager-initiated time period mode

The time period mode is used by the manager to enable an agent to send any data it collects for the duration of the requested time period (refer to Figure 18). When an agent receives the start DataRequest message from the manager, the agent shall send a DataResponse message acknowledging the result status of the request (DataReqResult) without transferring any measurement data in the response message. If DataReqResult is data-req-result-no-error, anytime data becomes available, the agent shall use the EVENT REPORT service to send event report(s) containing the measurement data to the manager until the time period as specified in the data request has expired, it receives a stop request from the manager, or the association between the agent and manager is terminated. The agent determines whether to use a Confirmed or Unconfirmed Event Report message to transfer the data.

If the manager wants to extend the amount of time, it shall pass in the data-req-id, set the data-req-continuation bit in the mode, and set the data-req-time to the amount of time that the agent may continue transferring data. All other parameters in the DataRequest shall be ignored, and the settings from the original start command shall be used. The agent shall apply each new time period measured from the time the command is received. If a continuation command is received for a data-req-id that is not functioning in a timed mode, the agent shall return a result of data-req-result-continuation-not-supported. If a continuation command is received for a nonexistent data-req-id, the agent shall return data-req-result-invalid-req-id. For example, if the timer expires prior to receiving the continuation command, the data-req-id stops and is removed.

In timed mode, if the data-req-time is set to 0, the agent shall acknowledge the request, if confirmed, transfer immediately any data currently available in event reports, and then stop. In contrast to single response mode (see 8.9.3.3.4), timed mode allows the agent to use either Confirmed or Unconfirmed Event Report messages. For example, an agent may use the Confirmed Event Report to ensure the data have been received by the manager prior to removal from a local cache.

On receipt of a stop data request for an enabled data-req-id, the agent shall stop sending the event reports for that data-req-id immediately.

The data-req-id field in these event reports is used by the manager to couple these measurement data to the appropriate data request.

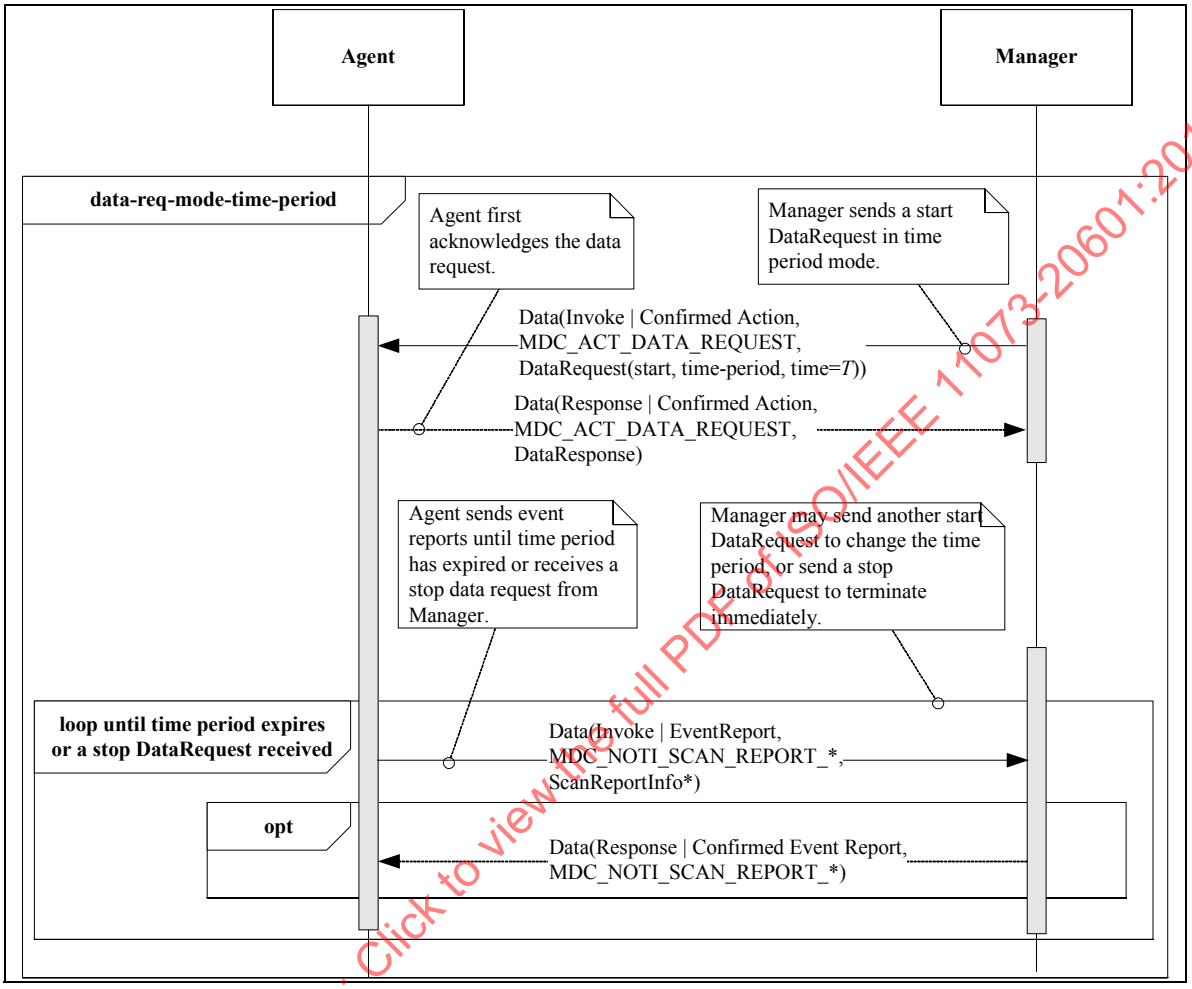


Figure 18—Manager-initiated measurement data transmission (data-req-mode-time-period)

8.9.3.3.6 Manager-initiated no time limit mode

The no time limit mode shall be used to command an agent to send event reports continually until a stop request command is received or the association between the agent and the manager is terminated (see Figure 19).

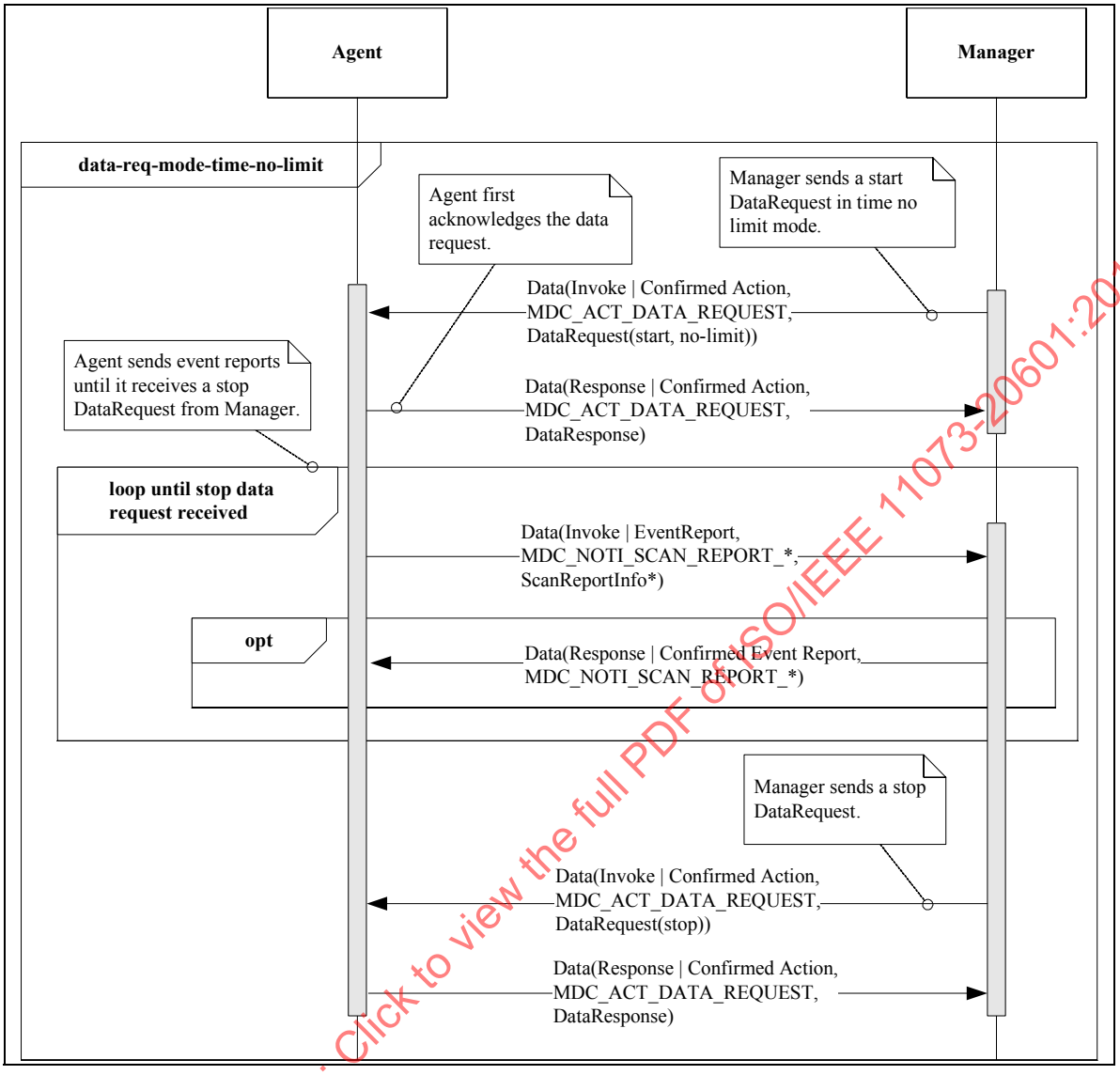


Figure 19—Manager-initiated measurement data transmission (data-req-mode-time-no-limit)

8.9.3.3.7 Scan report number management

A data request, in which the data-req-start-stop bit is set, establishes a new flow of one or more measurement observations from the agent to the manager in the context of the MDS. When a MDS flow is established, the agent creates a new instance of a scan-report-no counter for that flow. There shall be one instance of a scan-report-no counter for each flow, as differentiated by data-req-id. This counter shall start at 0 and increment by 1 for each event report sent on the flow, rolling over to 0. If an agent receives a data-request that has the data-req-start-stop bit set and a value of data-req-id that is already being used in a MDS flow, the agent shall reset the scan-report-no counter of the identified flow to 0. If an agent receives a data-request that has the data-req-continuation bit set, the scan-report-no shall continue counting without a reset.

The manager-initiated single response mode (data-req-mode-single-rsp) form of measurement transmission shall result in a response that has a 0 in the scan-report-no field of the ScanReportInfo* structure. This is because a new flow is created with each data-req-mode-single-rsp request and terminated when the response is sent.

An agent-initiated transfer from the MDS or scanner objects, by way of contrast, establishes a flow that terminates only when the association is broken. Thus for the agent-initiated transfer, the scan-report-no starts at 0, but cannot be reset by the manager within the context of the association. Setting the scanner's Operational-State attribute to disabled halts transmission of event reports, i.e., internal observation of metric objects is halted and continues again after setting the Operational-State attribute to enabled again. The scan-report-no in this case will continue counting from where it was halted.

8.9.3.3.8 Multiple MDS flows referencing a single measurement object

An agent may initiate, as well as receive, requests for flows that associate data-req-ids with metric objects through the context of the MDS. When a metric object that is associated with multiple flows generates measurement data, observations of the data shall be reported on each of the flows.

The agent shall report the maximum number of concurrent manager-initiated flows that it supports in data-req-init-manager-count during the association process. A manager shall limit the number of concurrent manager-initiated flows it requests so that the value reported by the agent is not exceeded. If an agent is unable to establish a new manager-initiated flow due to resource exhaustion, it shall set data-req-result to the value data-req-result-init-manager-overflow in the message response.

8.9.3.4 Persistently stored metric data transfer

8.9.3.4.1 General

When an agent implements one or more PM-store objects, the agent reports about the existence of the PM-store object during the configuration phase. The manager uses this information to query the PM-store object(s) of the agent. The interactions between the manager and agent when retrieving the information in the PM-store(s) is described in 8.9.3.4.2.

8.9.3.4.2 Persistently stored metric data transmission

a) **Retrieving the PM-store attributes.** When the agent and manager are in the Operating state, the manager can inspect the configuration negotiated with the agent to determine the number of PM-store objects in the agent. The manager may query each PM-store to determine the number of PM-segments that exist within the PM-store. Figure 20 shows the sequence diagram of this operation. The manager sends a Get command to the agent requesting attribute information from a particular PM-store. The manager uses the handle number to reference the desired PM-store. The manager shall leave the attribute-id-list empty to request all attributes be returned. The agent responds with the values of the requested attributes. The manager can inspect the attributes to learn about how the store is configured. For instance, the PM-Store-Capab describes the capabilities of the store, and Number-Of-Segments defines how many segments are present in the store. See Table 9 for the full list of attributes and their definitions.

   If the agent supports multiple PM-store object instances, a Get request is required for each PM-store.

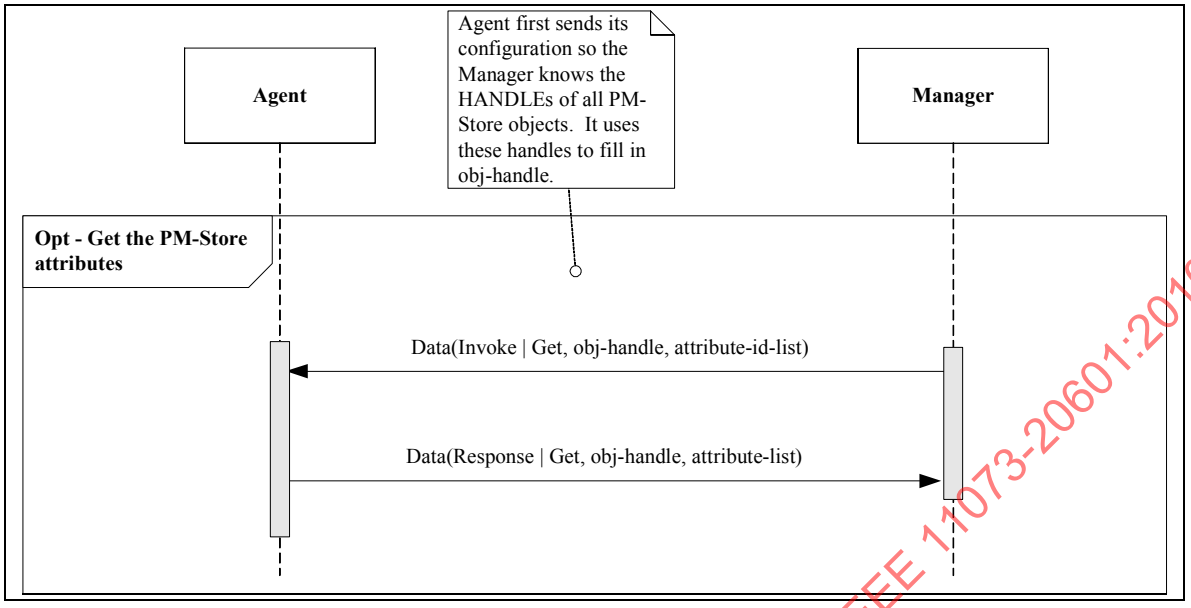


Figure 20 —Retrieving a PM-store's attributes

b) **Retrieving the PM-segment information.** The manager retrieves information on the segments in a PM-store by sending an ACTION.Get-Segment-Info command to the specific PM-store (see Figure 21) with a request to return information from all segments, a particular list of segments, or any segments within a given time range. The agent shall support the first two selection criteria and may provide support for the time range selection criteria. The manager is able to determine whether the agent provides support by inspecting pmsc-abs-time-select in the PM-Store-Capab attribute of the PM-store information retrieved earlier.

The agent responds to the ACTION.Get-Segment-Info command with a list of segment numbers followed by the full attribute list for each of the segments.

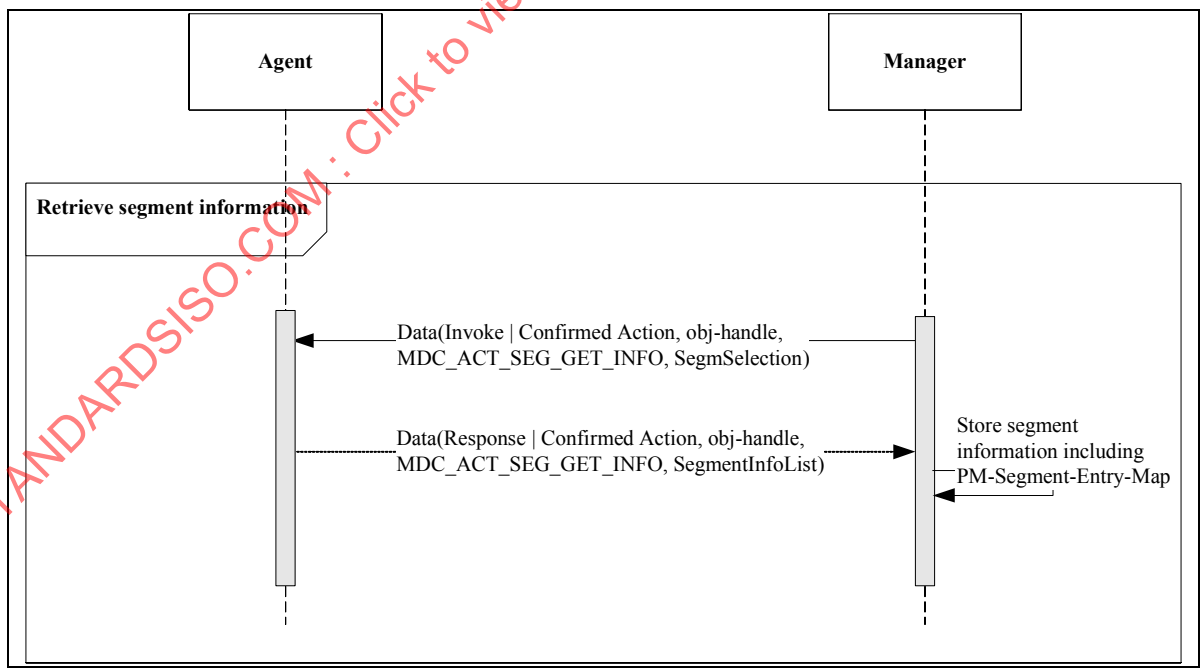


Figure 21 —Retrieve PM-segment information

c) **Transfer PM-segment content.** The manager retrieves specific PM-segments by using the Trig-Segm-Data-Xfer ACTION method to initiate the data transfer (see Figure 22). The manager shall pass on information about the PM-store handle to access and the segment instance number to transmit.

The agent shall decide whether the request can be honored. It checks for a valid segment number, available segment data (i.e., they could be in the update process), or any other error conditions. If there is an error, the agent shall return an appropriate error code in the response and ignore the transmit request. Otherwise, the agent shall send a tsxr-successful response code to indicate that it has received the request and it can be honored.

The manager may send the Trig-Segm-Data-Xfer ACTION invoke message at any time. However, if the manager does send a Trig-Segm-Data-Xfer ACTION invoke message while a Clear-Segments ACTION invoke message is outstanding, the agent may generate a Trig-Segm-Data-Xfer ACTION response message with a return code of trig-segm-xfer-rsp = tsxr-fail-clear-in-process. An example of when this return code might be sent is if the storage medium for the PM-store is a single Flash device. When a Flash device is being erased, it might cause the entire Flash device to be inaccessible.

The agent shall send confirmed Segment-Data-Event event reports until all entries in the PM-segment are sent to the manager or the transfer is aborted by either the sevtsta-agent-abort or sevtsta-manager-abort bits described later. The agent fills in the SegmentDataEvent structure with information about the segment being sent. The agent informs the manager of the PM-store handle and uses the SegmDataEventDescr to describe the segment number being transferred, the entry number of the first entry in the segm-data-event-entries field, the number of entries in the message, and current status information. The agent shall always set any sevtsta-manager-* bits to 0. If the message contains the first entry and/or the last entry of the data entries, then the agent shall set the sevtsta-first-entry and/or sevtsta-last-entry bits, respectively. If the agent wishes to abort the transfer, it shall set the sevtsta-agent-abort bit to 1.

When transferring a segment, the agent uses the segm-data-event-entries field to send all the entries. The agent shall start with the first entry collected, followed by the next entry, and so on. The agent should pack as many entries as possible into the event structure to optimize the transmissions. Each entry shall be formatted according to the structure defined in the PM-segment PM-Segment-Entry-Map.

When the manager receives an event report, it shall reply with a SegmentDataResult response that shall contain the same store-handle, segm-instance number, segm-evt-entry-index, and segm-evt-entry-count. In the segm-evt-status, the manager shall set the sevtsta-manager-confirm bit.

If the agent sets the sevtsta-agent-abort bit, then the manager shall confirm the agent abort by setting the same bit. If the manager wishes to abort the exchange, it shall set the sevtsta-manager-abort bit.

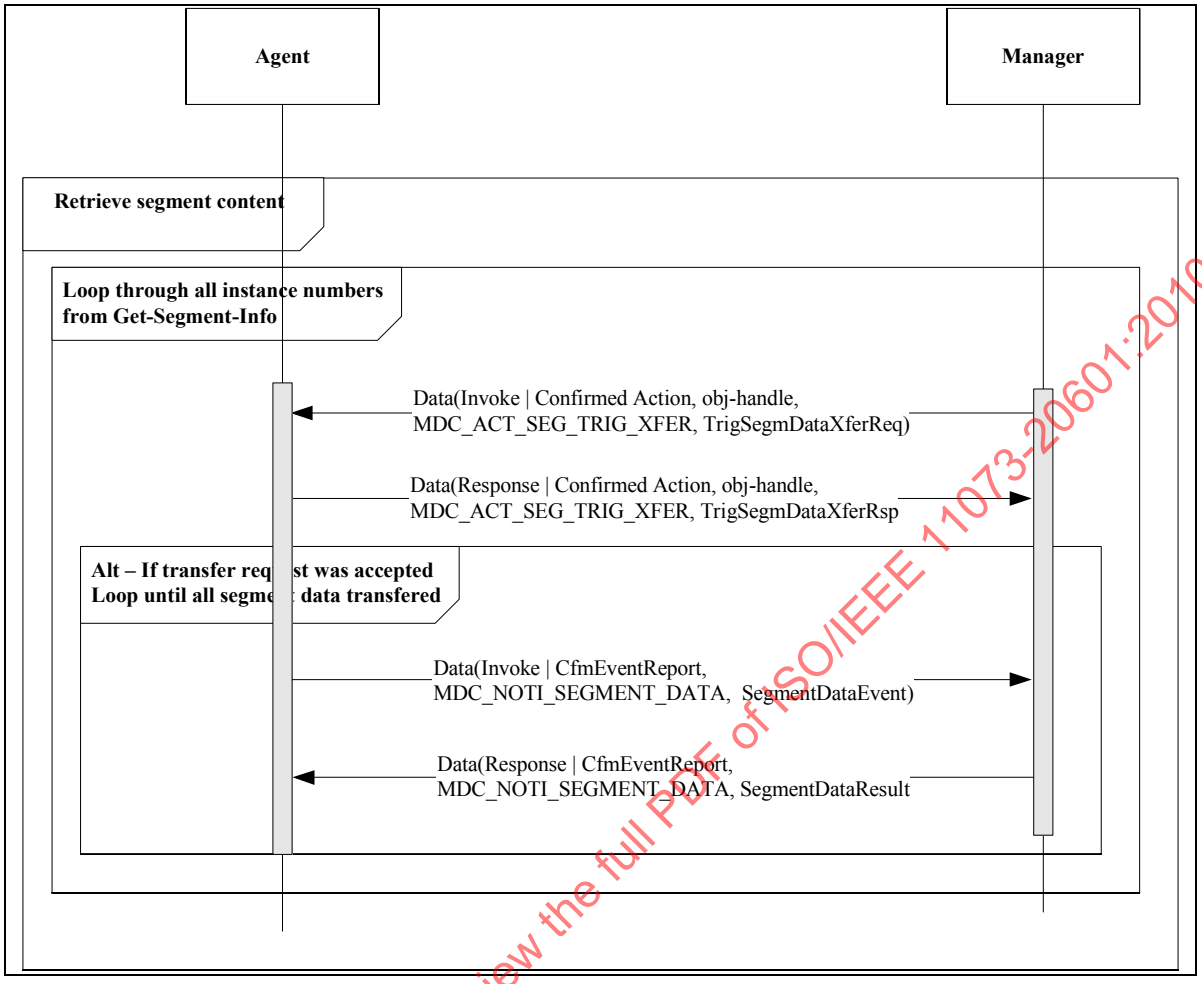


Figure 22 —Retrieve PM-segment content

d) **Clear a PM-segment.** The manager may clear a PM-segment at any time and uses the sequence shown in Figure 23. A typical time for clearing a segment is directly after the entire segment was transferred to the manager. The manager recognizes this condition when it receives a SegmEvtStatus with the sevtsta-last-entry bit set.

Whenever the manager decides to clear segment(s), it sends an ACTION command to the agent with the Clear-Segments method and segment selection criteria of all segments, a particular list of segments, or any segments within a given time range. The agent shall support clearing all segments, should support clearing a particular list of segments (pmsc-clear-segm-by-list-sup), and may support the time range selection criteria (pmsc-clear-segm-by-time-sup). The manager determines which capabilities are supported by inspecting the PM-Store-Capab attribute bits.

When the agent receives a Clear-Segment command, it may delete all present entries and leave the segment, or it can remove the segment. The manager determines which capabilities are supported by inspecting the pmsc-clear-segm-remove bit of the PM-Store-Capab attribute.

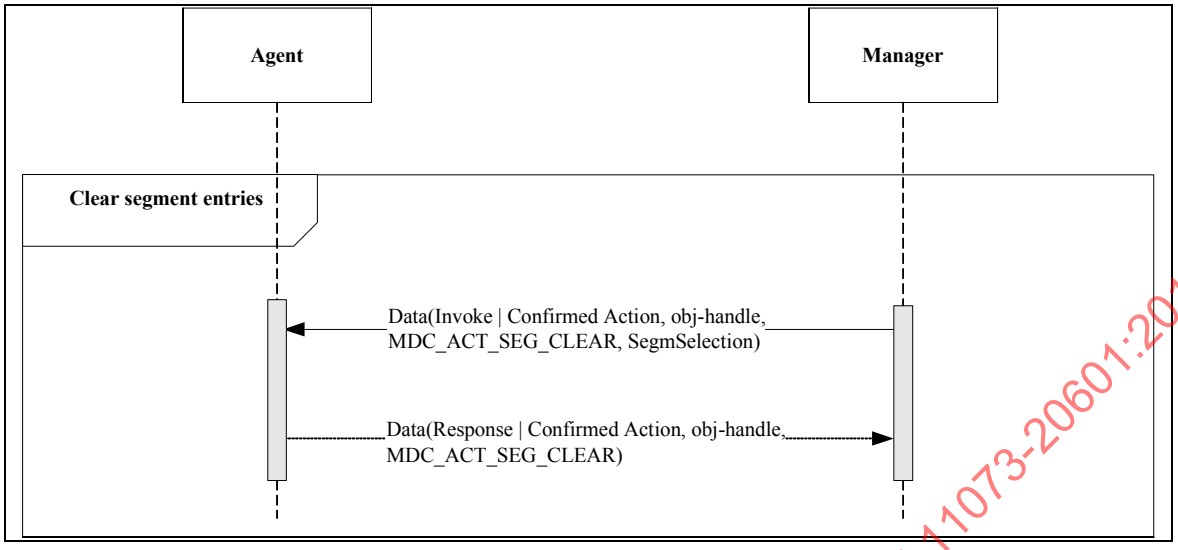


Figure 23 —Clear segment entries

8.9.4 Exit conditions

A normal exit from the Operating state occurs when the agent or manager decides to terminate the association. In this case, the agent or manager shall enter the Disassociating state and follow the disassociating procedure (see 8.10).

When an agent or manager receives an Association Release Request, it shall send an Association Release Response and transition to the Unassociated state.

An agent exiting the Operating state, either normally or abnormally, shall stop all data transfer mechanisms including agent- or manager-initiated measurement data transmission, PM-segment transmission, and scanner transmission.

8.9.5 Error conditions

8.9.5.1 General

As in 8.9.4, an agent exiting the Operating state, either normally or abnormally, shall stop all data transfer mechanisms including agent- or manager-initiated measurement data transmission, PM-segment transmission, and scanner transmission.

If, at any time, there is a transport layer timeout from the reliable transport layer, then the agent or manager shall do the following:

— For timeout/connection-**dependent** reliable transports (TCP, for example), transition back to the Disconnected state, due to the fact that transport timeouts are reported as a “transport disconnect indication” to the upper layers.

— For timeout/connection-**independent** reliable transports (USB, for example), attempt to recover the transport link, attempt to send an Association Abort message to its peer, and then transition back to the Unassociated state.

8.9.5.2 Confirmed Action

After sending a Confirmed Action invoke message, the manager shall wait for a Confirmed Action response message for a TO_{ca} (timeout: confirmed action service) period. If the TO_{ca} expires, the manager shall send an Association Abort message to the agent and transition back to the Unassociated state.

8.9.5.3 Confirmed Event Report

After sending a Confirmed Event Report invoke message, the agent shall wait for a Confirmed Event Report response message for a TO_{cer} (timeout: confirmed event report service) period. If the TO_{cer} expires, the agent shall send an Association Abort message to the manager and transition back to the Unassociated state.

The TO_{cer} is defined on a per-object basis. Each of the objects in this standard that generate event reports has a separate timeout value that is reported by an appropriate attribute in each object:

— $TO_{cer\text{-}mds}$ (TO_{cer} for the MDS object) MDS.Confirmed-Timeout

— $TO_{cer\text{-}pms}$ (TO_{cer} for the PM-store object) Segm.Confirmed-Timeout

— $TO_{cer\text{-}scan}$ (TO_{cer} for the scanner object) Scan.Confirmed-Timeout

8.9.5.4 Get

After sending a Get invoke message, the manager shall wait for a Get Response message for a TO_{get} (timeout: get service) period. If the TO_{get} expires, the manager shall send an Association Abort message to its peer and transition back to the Unassociated state.

8.9.5.5 Confirmed Set

After sending a Confirmed Set invoke message, the manager shall wait for a Confirmed Set Response message for a TO_{cs} (timeout: confirmed set service) period. If the TO_{cs} expires, the manager shall send an Association Abort message to the agent and transition back to the Unassociated state.

8.9.5.6 Special timeouts

In addition to the typical communication service timeouts described previously, there are three special-case timeouts that are also used in the personal health device protocol:

— $TO_{clr\text{-}pms}$: special TO related to the clearing of the PM-store object PMS.Clear-Timeout

— $TO_{sp\text{-}mds}$: special TO inter-service timeout for the MDS object 3 s

— $TO_{sp\text{-}pms}$: special TO for PM-store segment transfer Segm.Transfer-Timeout

For $TO_{clr\text{-}pms}$, after sending a Confirmed Action (MDC_ACT_SEG_CLR) invoke message, the manager shall wait for a Confirmed Action response message for a $TO_{clr\text{-}pms}$ (timeout: confirmed action service to clear the PM-store object) period. If the $TO_{clr\text{-}pms}$ expires, the manager shall send an Association Abort message to the agent and transition back to the Unassociated state.

For $TO_{sp\text{-}mds}$, after sending a Confirmed Action (MDC_ACT_DATA_REQUEST, start, time-period, time=0) invoke message, the manager shall wait for a Confirmed Event Report invoke message for a $TO_{sp\text{-}mds}$ (timeout: special interservice timeout for the MDS object) period. If the $TO_{sp\text{-}mds}$ expires, the manager shall send an Association Abort message to the agent and transition back to the Unassociated state.

For $TO_{sp\text{-}pms}$, after sending a Confirmed Action (MDC_ACT_SEG_TRIG_XFER) invoke message, the manager shall wait for a Confirmed Event Report (segm-evt-status=sevtsta-last-entry, semg-data-event-entries) invoke message for a $TO_{sp\text{-}pms}$ (timeout: special segment transfer timeout of the PM-store object) period. If the $TO_{sp\text{-}pms}$ expires, the manager shall send an Association Abort message to the agent and transition back to the Unassociated state.

8.10 Disassociating procedure

8.10.1 General

The disassociating procedure provides a mechanism for either the agent or manager to release the association gracefully.

8.10.2 Entry conditions

When an agent or manager decides to close the association, it shall transition to the Disassociating state and initiate the disassociating procedure.

8.10.3 Normal procedures

In the Disassociating state, the agent or manager sends an Association Release Request to its peer and waits for the response. The Association Release Request contains a ReleaseRequestReason to indicate the reason for releasing the association:

— The no-more-configurations reason is used by the agent during the Configuring state to indicate that all possible configurations were attempted and the manager rejected every one.

— The configuration-changed reason is used by the agent during the Operating state to indicate that the agent's configuration changed and it is not possible to continue sending data with the previously agreed-upon configuration. Typically, the agent follows this message by sending a new Association Request with a new dev-config-id; however, this step is not required.

— The normal reason is used by either the agent or manager to leave the Operating state without indicating a special condition.

Should an agent or manager receive an Association Release Request when it has an outstanding invoke-id, it shall respond with an Association Release Response and assume that it shall receive no response to its request.

8.10.4 Exit conditions

When the agent or manager receives the response to its release request, it shall transition to the Unassociated state.

If an agent or manager receives an Association Release Request while in the Disassociating state, then it shall send an Association Release Response and shall remain in the Disassociating state waiting for the response to its own Association Release Request.

8.10.5 Error conditions

After sending an Association Release message, the agent or manager shall wait for an Association Release Response message for a $TO_{release}$ (timeout: association release procedure) period. If the $TO_{release}$ expires, the agent or manager shall send an Association Abort message to its peer and transition back to the Unassociated state.

The agent or manager can either send or receive an Association Abort message for other failure conditions and shall transition immediately to the Unassociated state.

8.11 Message encoding

The ASN.1 used in this standard allows conversion into many possible transmission formats. Both manager and agent shall support the MDER as defined in ISO/IEEE 11073-20101:2004 [B14]. The MDER encoding rules are reproduced in Annex F along with additional optimizations specific to this standard. Further, for binary transmissions, network byte order (big-endian encoding) shall be used. This standard also allows manager and agent to negotiate alternate encodings of PER [B17] and XER [B18].

Annex G shows one example of how the ASN.1 data structures could be encoded into C syntax.

Annex H contains supporting examples of binary encodings resulting from messages defined in this standard.

All of the nomenclature codes used in this standard are defined using the MDC_... representation, but the nomenclature codes shall be used during transmission. Annex I contains a listing of the defined values for all the codes utilized in this standard.

8.12 Time coordination

8.12.1 General

There are three types of clocks that an agent may implement: absolute time, relative time, and high-resolution relative time. In all cases, information about the clock capabilities of the agent and whether one or more of the clocks are synchronized with an external time source can be found via the Mds-Time-Info attribute in Table 2. All bit references in the subclauses are part of this attribute. All agents with any type of clock shall support this attribute.

8.12.2 Absolute time

8.12.2.1 General

Agents with an internal real-time clock (RTC) shall indicate this capability by setting the mds-time-capab-real-time-clock bit (see A.11.1). Agents that support the Set-Time action (see 6.3.2.4 and A.4) shall set the mds-time-capab-set-clock bit.

Agents may support an independent method to synchronize the internal RTC to external clock sources. The synchronization method used is not in the scope of this standard. However, the agent shall indicate whether it synchronizes absolute time using the mds-time-capab-sync-abs-time bit. If synchronization is supported, the protocol used to synchronize the internal RTC (e.g., NTP and SNTP) is reported in the time-sync-protocol field using IDs such as MDC_EXT_PROTO_TIME_NTP. The mds-time-state-abs-time-synced bit shall be set only when the agent believes its Date-and-Time attribute is synchronized with the external clock source.

Agents may wish to indicate to the manager whether it should set the time with the Set-Time action. If the mds-time-mgr-set-time bit is set, the manager shall use the Set-Time action command to set the absolute time on the agent. If not set, the agent does not want the manager to set the clock. This situation can occur when the agent is synchronizing the clock via an external clock source or when the user has set the clock locally. In this case, the manager shall not attempt to set the clock.

The Date-and-Time and Absolute-Time-Stamp attributes report the agent's date and time. For some usages, it is important that the agent report the date and time as it was displayed to the person using the device (e.g., a glucose meter). For other usages, reporting a time that is coordinated to a universal time system such as universal time coordinated (UTC) is important. For example, this situation can occur when the date and time are set in the factory to UTC and the device does not expose a method for altering the date and time. On the manager side, the ability to associate the measurement times with the manager's notion of time is critical for some usages.

8.12.2.2 Comparable time

This standard utilizes a concept of "comparable time" to support all three usages. The key concepts of comparable time are as follows:

— When an agent reports time information, it shall ensure that all measurements reported as a set are from the same, unbroken timeline. For temporary measurements, a set consists of all the measurements in a single event report. For PM-store, a set is equivalent to a PM-segment.

— If a set of measurements was collected when the current clock was set differently, then the agent shall either discard the data or communicate the data along with the number of 1/100 of a second to add to each of the measurement times to place them on the same timeline as the agent's current Date-and-Time attribute.

— The above two concepts apply only if the time base used to time stamp the values has changed by an amount significant for the type of measurement. In other words, small clock drifts or minor adjustments to a clock to keep it synchronized with an external time source do not need to be reported

Absolute time shall be interpreted as comparable time for the agent as follows:

— If an agent is associated with a manager when the Date-and-Time attribute is adjusted, it shall send an event report that contains the new Date-and-Time attribute value. The one exception is the case where the manager uses the Set-Time command to change the agent's time. In this case, the agent may decide not to send the event report since the manager already knows the time was changed.

— If an agent collects temporary measurements and the Date-and-Time attribute is adjusted, the agent shall ensure that all measurements included in an event report come from the same, unbroken timeline, that is, no time adjustments occurred within the span of the timestamps contained in that event report. Further, all event reports that contain measurements prior to the time when the current time of the agent was adjusted shall have the MDS attribute Date-and-Time-Adjustment as the first reported data in the event report. This attribute shall specify the number of 1/100s of a second to add to each timestamp in the event report to align with the current clock (e.g., if the clock was advanced by 60 min, this would report 360 000).

— If an agent collects PM-store measurements and the Date-and-Time attribute is adjusted, the agent shall ensure that each PM-segment includes only measurements from the same, unbroken timeline. Further, the PM-segment attribute Date-and-Time-Adjustment shall be present in any PM-segment that contains measurements collected with respect to a different clock setting.

— Note that in the cases where measurements are collected off line, if the clock is changed multiple times before uploading data, the Date-and-Time-Adjustment value is cumulative. In other words, measurements are collected, then the clock is set backwards 30 min, more measurements are collected, and the clock is set back another 30 min Then the first set of data needs to report an offset of –60 min, and the second set reports an offset of –30 min.

8.12.3 Relative time

Agents may implement a relative timer with time resolution down to 125 µs [least significant bit (LSB)]. This resolution is sufficient for sampling rates up to 8 kHz, permits high-resolution relative time periods to be measured, and spans time periods up to 6.2 days. If relative time is used with either temporarily stored measurements or a PM-store, agents shall ensure that the length of storage time never exceeds the resolution of the timer (i.e., 6.2 days). This assurance from an agent allows the manager to query the agent's current relative time and compute how long ago the measurement was taken. If longer storage times are required, either absolute time or high-resolution relative time attributes are used. Agents shall indicate support for relative time by setting the mds-time-capab-relative-time bit in the Mds-Time-Info attribute. This timer shall be initialized prior to association. With the exception of counter rollover, it shall monotonically increase its count and shall not have its value changed once initialized. The actual time resolution (i.e., internal update period) is defined by the agent, but should be appropriate for the purpose of that device.

Agents may support a method to synchronize their internal timer to an external clock source (e.g., Bluetooth piconet). The synchronization method used is not in the scope of this standard. However, the agent shall indicate whether it synchronizes relative time using the mds-time-capab-sync-rel-time bit. If synchronization is supported, the mds-time-state-rel-time-synced bit shall be set only when the agent believes its relative clock is synchronized with the external source. All agents connected to the same manager and indicating their internal timers are synchronized should supply the same relative time for events synchronized in time.

If the agent provides a relative time stamp for a numerical measurement, the time stamp shall be accurate within the limits of the stated time synchronization accuracy and the sample time of the numerical value. The relative time stamp, when used as event time and current time, may provide accurate event-to-event interval time. The relative time stamp can provide accurate absolute time measurements when the manager gets the relative time and absolute time attributes from the MDS object of the agent and determines time relative to its own internal clock.

If the agent provides a relative time stamp for an RT-SA, the time stamp shall relate to the first sample in that array, and the time stamp is accurate within the limits of the stated accuracy of the relative time attribute and the sample time of the RT-SA.

Event reports may contain relative time stamps indicating when the event was generated. If metric-derived objects held in the event report do not have an overriding time stamp, then the event time shall be used as the measurement time as well. If the agent provides a relative time stamp for an event time, the time stamp is accurate within the limits of the stated accuracy of the relative time attribute, the time of the event, and any sample time attributes associated with the event.

8.12.4 High-resolution relative time

Agents may implement an internal high-resolution timer with time resolution down to 1 µs (LSB). This high-resolution timer is sufficient for sampling rates up to 1 MHz, permits very high resolution relative time periods for measurement, and spans time periods up to 584 000 years. Agents shall indicate support for this feature by setting the mds-time-capab-high-res-relative-time bit in the Mds-Time-Info attribute. This timer shall be initialized prior to association. With the exception of counter rollover, it shall monotonically increase its count and shall not have its value changed once initialized. The actual time resolution (i.e., internal update period) is defined by the agent, but should be appropriate for the purpose of that device. High-resolution relative time should retain frequency synchronization with relative time.

Agents may support a method to synchronize their high-resolution internal timer to an external clock source (e.g., Bluetooth piconet). The synchronization method used is not in the scope of this standard. However, the agent shall indicate whether it synchronizes high-resolution relative time using the mds-time-capab-sync-hi-res-relative-time bit. If synchronization is supported, the mds-time-state-hi-res-relative-time-synced bit shall be set only when the agent believes its relative clock is synchronized with the external source. When the agent disconnects from the clock synchronization source, it shall set the synced bit to false as soon as it exceeds the accuracy of the clock synchronization parameters.

If the agent provides a high-resolution relative time stamp for a numerical measurement, the time stamp shall be accurate within the limits of the stated accuracy of the relative time attribute and the sample time of the numerical value.

If the agent provides a high-resolution relative time stamp for an RT-SA, the time stamp shall relate to the first sample in that array. The time stamp is accurate within the limits of the stated accuracy of the relative time attribute and the sample time of the RT-SA.

9. Conformance model

9.1 Applicability

It is expected that this standard will be referenced by other standards in the ISO/IEEE 11073 family of standards to define applications (e.g., for the exchange of personal health measurement data) or to define functional communication profiles (e.g., personal health device interoperability profiles).

In particular, the series of ISO/IEEE 11073-104zz device specializations are necessary to enable an interoperable system. Thus, interoperability requires checking for conformance against this standard and the set of device specializations that are implemented. The device specializations define an appropriate conformance model that includes conformance requirements from this standard for personal health device representation. Device specializations utilize device specific information to define additional conformance criteria that are out of the scope of this standard.

Conformance to definitions of this standard is specified primarily at the appropriate application interface or system interface. Further, behaviors specified by this standard (such as adherence to the specified state machines) are also part of the specification. The behavior at this level is considered critical for conformance to assure the proper and accurate operation of the protocol as a whole. Implementation details such as programming language, layering of software, internal interfaces, and so on are not subject to conformance specifications.

9.2 Conformance specification

This standard on personal health representation offers a high degree of flexibility in how the model is applied for a particular personal health device, particularly in the following areas:

— Information model of a specific device

— Use of attributes, value ranges, and access

— Use of extended communication services (i.e., scanners), scan periods, and scanner configurability

To support interoperability of applications and systems, an implementation based on this standard shall provide specific details about the way that the definitions of this standard are applied, in conjunction with the conformance requirements of any derived device specializations.

These specifications take the form of a set of implementation conformance statements (ICSs). An ICS is a form of data sheet that discloses details of a specific implementation and specifies the features provided. Specific applications or functional communication profiles that are based on this standard shall define more specific conformance requirements, in addition to the ICS defined here.

NOTE—The ICSs defined in the following subclauses provide understanding of the details of an implementation. However, they are not sufficient to provide interoperability of devices or applications. For such interoperability, additional specifications (e.g., timing, latencies, and system loading assumptions) shall be taken into account. These specifications are not within the scope of this standard.

9.3 Implementation conformance statements (ICSs)

The general format of the ICSs is in the form of tables. Templates for these ICS tables are given in Table 22, Table 23, Table 24, Table 25, Table 26, Table 27, Table 28, and Table 29. The tables are to be filled out and provided as an overall conformance statement document.

Generally the column headers of an ICS table contain the following information:

— Index, which is an ID (e.g., a number) of a specific feature.

— Feature, which briefly describes the characteristic for the conformance statement to make.

— Reference, which is a reference to the definition of the feature (may be empty).

— Status, which specifies the conformance requirement (i.e., the requirements for a conforming implementation regarding the feature). In some cases, this standard does not specify conformance requirements, but still wants a definition of the status for a particular feature.

— Support, which is filled out by the implementer and specifies the characteristics of the feature in the implementation.

— Comment, which contains additional information provided by the implementer.

9.4 General conformance

Table 22, Table 23, and Table 24 are intended for use in describing the general base conformance to this standard. Articulated in 9.4.1 through 9.4.3 are the fundamental aspects of the support that a device shall have to claim conformance.

9.4.1 General ICS

In a top-level general ICS, the implementer specifies the versions/revisions that are supported by the implementation as well as some high-level system behavior definitions.

Table 22 shows the general ICS.

Table 22—General ICS

Index	Feature	Reference	Status	Support	Comment
GEN-1	Implementation Description	—	Identification of the device/application. Description of functionality.		
GEN-2	Standard Document Revision	(Standard documents)	Identification of the supported revisions to IEEE Std 11073-20601.	(Set of supported IEEE 11073-20601 revisions)	
GEN-3	Conformance Adherence - Level 1 -	—	Base conformance declaration that device meets the following IEEE 11073-20601 conformance requirements: A) All minimum mandatory (shall) requirements (See Table 23 in 9.4.2 for some of the more critical aspects.) B) All conditional elements were implemented according to the stated conditions. C) All optional elements that are implemented are defined as part of the conformance statement (e.g., in the Attribute ICS tables (see Table 27)).	Yes/No (No implies NON-conformant)	
GEN-4	Conformance Adherence - Level 2 -	—	In addition to GEN-3, device conforms to one or more device specializations based on IEEE Std 11073-20601.	(list the set of IEEE 11073-20601 device specializations that were followed and prepare the information specified in 9.5)	
GEN-5	Communication Profile and Hardware	—	Description of communication infrastructure and hardware requirements for interfacing.		

For each implementation, one general ICS shall be provided.

9.4.2 Minimum requirements ICS

Table 23 shows the minimum requirements for conformance to this standard.

Table 23—IEEE 11073-20601 minimum requirements

Index	Feature	Reference	Status	Support	Comment
REQ-1	State Machine	—	-Mandatory- Does the implementation have strict adherence to the IEEE 11073-20601 personal health device articulated state machine behavior?	Yes/No (No implies NON-conformant)	
REQ-2	Protocol Messages	—	-Mandatory- Does the implementation adhere to the IEEE 11073-20601 personal health device protocol messages?	Yes/No (No implies NON-conformant)	
REQ-3	Objects	—	-Recommended- Do all objects adhere to IEEE Std 11073-20601 or device specializations based on IEEE Std 11073-20601? Adherence to this set of objects, fields, values, and behavior is strongly recommended.	Yes/No. If no, list the extensions as described in 9.5.2.	
REQ-4	Encoding	—	-Mandatory- Is MDER supported? The protocol messages are encoded from the ASN.1 description to/from transmission format using encoding rules. Support for MDER is required. These encoding rules are defined in Annex F of IEEE Std 11073-20601. Negotiation of an alternate encoding rule is allowed. List all supported encoding rules.	Yes/No (No implies NON-conformant) (list of alternate encoding rules that are supported)	
REQ-5	Nomenclature	—	-Mandatory- IEEE Std 11073-20601 is based on ISO/IEEE 11073-10101 [B12] Nomenclature for base nomenclature. IEEE Std 11073-20601 and associated device specializations augment those with additions. Are all nomenclature codes in compliance with one of these sources?	Yes/No (No implies NON-conformant)	
REQ-6	Transport	—	-Mandatory- List all transport classes (i.e., reliable and/or best-effort) supported by the implementation. Are all transport requirements as documented in IEEE Std 11073-20601 met for these transports?	(List of transport classes) Yes/No (No implies NON-conformant)	

9.4.3 Service support ICS

The service support ICS defines services defined in the service model that are implemented. This ICS is supplied only for communicating devices.

Table 24 shows the service support ICS.

Table 24—Service support ICS

Index	Feature	Reference	Status	Support	Comment
SRV-1	GET Service	7.3	Does the implementation support GET? conditional	Sends command, and/or accepts command, or not supported.	
SRV-2	SET Service	7.3	Does the implementation support SET? conditional	Sends command, and/or accepts command, or not supported.	
SRV-3	Confirmed SET Service	7.3	Does the implementation support confirmed SET? optional	Sends command, and/or accepts command, or not supported.	
SRV-4	EVENT REPORT Service	7.3	Does the implementation support EVENT REPORT? conditional	Sends command, and/or accepts command, or not supported.	
SRV-5	Confirmed EVENT REPORT Service	7.3	Does the implementation support confirmed EVENT REPORT? conditional	Sends command, and/or accepts command, or not supported.	
SRV-6	ACTION Service	7.3	Does the implementation support ACTION? conditional	Sends command, and/or accepts command, or not supported.	
SRV-7	Confirmed ACTION Service	7.3	Does the implementation support confirmed ACTION? optional	Sends command, and/or accepts command, or not supported.	

The Support column of the completed table should define if the implementation invokes the service (e.g., sends a GET PDU), provides the service (e.g., processes a received GET PDU), or does not implement the service at all.

In addition, specific restrictions are listed (e.g., if a specific service is restricted to only one object class).

9.5 Device additions/extensions ICS

Table 25, Table 26, Table 27, Table 28, and Table 29 are intended for use in describing the ICS for any additions or extensions the device uses beyond this standard and its specializations. It is expected that all conditional or optional behaviors are articulated as part of the corresponding conformance statement for the respective device specializations.

9.5.1 General additions/extensions ICS

The general additions/extensions ICS defines the basic background on the scope of the supported additions/extensions.

Table 25—General additions/extensions ICS

Index	Feature	Reference	Status	Support	Comment
ADD-1	Use of Private Objects	—	Does the implementation use objects that are not defined in IEEE Std 11073-20601 or any of the listed device specializations?	Yes/No [If yes: POC ICS (see 9.5.2) shall be utilized to explain implementation details]	
ADD-2	Use of Non-20601 Nomenclature Codes from ISO/IEEE 11073-10101 [B12].	—	Does the implementation use nomenclature codes from the ISO/IEEE 11073-10101 [B12] that are not part of IEEE Std 11073-20601 or any of the listed device specializations?	Yes/No (If yes: explain in the appropriate ICS, see 9.5.6)	
ADD-3	Use of Private Nomenclature Extensions	—	Does the implementation use private extensions to the nomenclature? Private nomenclature extensions are allowed only if the standard nomenclature does not include the specific terms required by the application.	Yes/No (If yes: explain in the appropriate ICS, see 9.5.6)	
ADD-4	Payload Format	—	Were any additional payload formats introduced beyond those defined in IEEE Std 11073-20601 or any of the listed device specializations?	Yes/No (If yes then explain fully with purpose and layout. Should be described in ASN.1)	

9.5.2 Personal health device DIM object and class (POC) ICS

The POC ICS defines which managed objects from this standard are instantiated by the implementation and references the class of each object. Table 26 is a template only. For each object supported by the implementation, one row shall be filled out.

Table 26—Template for POC ICS

Index	Feature	Reference	Status	Support	Comment
POC-*n*	Object Description	The class of the object (i.e., numeric, etc.)	Implemented	Specify restrictions (e.g., maximum number of supported instances)	

The *n* in the Index column should be the object handle for implementations that have pre-defined objects. Otherwise, the Index column shall simply be a unique number (1..*m*).

All private objects shall be specified and include a reference to the definition for the object. Where no publicly available reference is available, the definition of the object should be appended to the conformance statement.

The Support column should indicate any restrictions for the object implementation.

An object containment diagram (class instance diagram) should be provided as part of the POC ICS.

9.5.3 POC attribute ICS

For each supported object defined in the POC ICS, a POC attribute ICS is provided to define the conditional, optional, or extended attributes used/supported by the implementation, including any inherited attributes. Mandatory attributes do not need to be listed since they are required to be implemented to be conformant.

Table 27 is a template only.

Table 27—Template for POC attribute ICS

Index	Feature	Reference	Status	Support	Comment
ATTR-*n*–*x*	Attribute Name. Extended attributes shall include the Attribute ID also.	Fill in the reference to the ASN.1 structure if the attribute is not defined in this standard or one of the listed device specializations.	Implemented	Describe: Access Value ranges Additional restrictions value	

The *n* in the Index column is the ID of the managed object for which the table is supplied (i.e., the index of the managed object as specified in the POC ICS in 9.5.2). There is one separate table for each supported managed object.

The *x* in the Index column is just a serial number (1..*m*).

All attributes beyond those defined in this standard or any of the listed device specializations shall be specified and include reference to the definition for the attribute. Where no publicly available reference is available, the definition of the attribute should be appended to the conformance statement.

The attribute access specification fields in the Support column are specified if the implementation provides access services for attributes.

The Support column should also contain attribute value ranges (if applicable), hints about specific restrictions for attribute access or attribute availability and information, and an indication if the attribute value is static or dynamic in the implementation.

NOTE—The attribute definition tables in this standard define a minimum mandatory set of attributes for each object.

9.5.4 POC behavior ICS

The POC behavior ICS specifies all implemented object methods that can be invoked by the ACTION service. Table 28 is a template only. One table is provided for each object that supports special methods.

Table 28—Template for POC behavior ICS

Index	Feature	Reference	Status	Support	Comment
ACT- *n–x*	Method Name. Methods not defined in the standards shall include the Method ID also.	Fill in the reference to the ASN.1 structure if method is not defined in this standard or one of the listed device specializations.		Specific restrictions	

The *n* in the Index column is the ID of the managed object for which the table is supplied (i.e., the index of the managed object as specified in the POC ICS). There is one separate table for each managed object that supports specific object methods (i.e., actions).

The *x* in the Index column is just a serial number (1..*m*).

All methods beyond those defined in this standard or any of the listed device specializations should be specified and include reference to the definition for the method. Where no publicly available reference is available, the definition of the method should be appended to the conformance statement.

The Support column should specify any restrictions for the method.

9.5.5 POC notification ICS

The POC notification ICS specifies all implemented notifications (typically in form of the EVENT REPORT service) that are emitted by supported objects. Table 29 is a template only. One table is provided for each object that supports special object notifications.

Table 29—Template for medical object class (MOC) notification ICS

Index	Feature	Reference	Status	Support	Comment
NOTI-*n–x*	Notification Name and Notification ID	Reference to the subclause in this standard where the event is defined.		Specific restrictions, ID, and description of each object involved	

The *n* in the Index column is the ID of the managed object for which the table is supplied (i.e., the index of the managed object as specified in the POC ICS). There is one separate table for each managed object that supports specific object notifications (i.e., events).

The *x* in the Index column is just a serial number (1..*m*).

All private notifications shall be specified and include reference to the definition for the notification. Where no publicly available reference is available, the definition of the notification should be appended to the conformance statement.

The Support column should specify any restrictions for the notification.

9.5.6 POC nomenclature ICS

The POC nomenclature ICS specifies all implemented nomenclatures that are utilized by the agent. Table 30 is a template only. One row of the table is to be used for each nomenclature element.

Table 30—Template for MOC nomenclature ICS

Index	Feature	Reference	Status	Support	Comment
NOME-*n*	Nomenclature Name and Nomenclature Value	Reference to the subclause in the standard or other location where the nomenclature is defined or used		Describe how the nomenclature is used. Describe any specific restrictions	

The *n* in the Index column is just a sequential number for uniqueness (1..m).

Annex A

(normative)

ASN.1 definitions

A.1 General

This annex provides ASN.1 definitions relevant for the personal health device protocol. Some are imported from other parts of the ISO/IEEE 11073 family of standards and others are created specifically for the personal health device domain. If there is interest in understanding which structures are imported and which are new, see Annex J. This annex ensures that all data structures required to implement this standard are readily available.

The naming convention followed in this annex is to use hyphen (-) to separate words in attributes and to use mixed case when describing data types; however, constructs that were imported from other specifications follow the existing use of capitalization and hyphenation.

A.2 Common data types

This subclause defines a set of ASN.1 data types that are used in the object definitions.

A.2.1 Integer and bit string data types

For representing integer numbers, the object definitions use fixed-size data types only. The bit string data type represents a bit field where each single bit has a defined meaning (i.e., flag fields). The following integer data types and bit string data types are used:

```
--
-- 8-bit unsigned integer
--
INT-U8 ::= INTEGER (0..255)
--
-- 8-bit signed integer
--
INT-I8 ::= INTEGER (-128..127)
--
-- 16-bit unsigned integer
--
INT-U16 ::= INTEGER (0..65535)
--
-- 16-bit signed integer
--
INT-I16 ::= INTEGER (-32768..32767)
--
-- 32-bit unsigned integer
--
INT-U32 ::= INTEGER (0..4294967295)
--
-- 32-bit signed integer
```

```
--
INT-I32 ::= INTEGER (-2147483648..2147483647)
--
-- 16-bit bit string
--
BITS-16 ::= BIT STRING (SIZE(16))
--
-- 32-bit bit string
--
BITS-32 ::= BIT STRING (SIZE(32))
```

Note that in object definitions, integer and bit string data types with named constants or named bits use the above defined basic data types for simplicity. This approach provides an abbreviated notation, but it is illegal ASN.1 syntax. It can be easily transformed to the correct syntax. For example, the definition

```
NamedConstant ::= INT-U16 {
                  const1(1),
                  const2(2)
}
```

becomes correct ASN.1 syntax defined as:

```
NamedConstant ::= INTEGER {
                  const1(1),
                  const2(2)
} (0..65535)
```

A.2.2 Identification data type

All elements (e.g., classes, objects, and measurement types) that need unique identification are assigned an object identifier (OID). The set of valid OIDs for this standard is defined in ISO/IEEE 11073-10101 [B12]. The nomenclature consists of a set of partitions, where each partition covers a specific concept and has its own 16-bit codes. In other words, a specific code is identified by both its partition number and an OID within that partition or its use is context dependent. In the case of context-dependent codes, the specific partition the code utilized is called out within this standard.

The 16-bit identification data type is defined as follows:

```
--
-- OID type as defined in nomenclature
-- (do not confuse with ASN.1 OID)
--
OID-Type ::= INT-U16 -- 16-bit integer type
```

A private partition is available for codes and IDs that are yet to be standardized or for manufacturer-specific codes.

```
--
-- Private OID
--
PrivateOid ::= INT-U16
```

A.2.3 Handle data type

The handle data type is used for efficient and locally unique identification of all managed object instances. (Locally unique means unique within one MDS context.) This data type is defined as follows:

```
--
-- handle
--
HANDLE ::= INT-U16
```

A.2.4 Instance number data type

The instance number is used to distinguish class or object instances of the same type or object instances that are not directly manageable (used, e.g., as the identification attribute for PM-segment objects).

```
--
-- Instance Number
--
InstNumber ::= INT-U16
```

A.2.5 Type ID data type

The type ID data type is used to identify the type of all elements (e.g., classes, objects, and measurement types). It is similar to the OID type (B2.2), but includes both the nomenclature partition and code to provide unique identification of an element. It shall be used when the context is not implicit. This data type is defined as follows:

```
--
-- Type ID
--
TYPE ::= SEQUENCE {
                partition       NomPartition,
                code            OID-Type
}
--
-- The following nomenclature partitions exist:
--
NomPartition ::= INT-U16 {
                nom-part-unspec(0),
                nom-part-obj(1),                -- object-oriented partition
                nom-part-metric(2),             -- metric [supervisory control and data acquisition
                                                -- (SCADA)] partition
                nom-part-alert(3),              -- alerts/events partition
                nom-part-dim(4),                -- dimensions partition
                nom-part-vattr(5),              -- virtual attribute partition for operation objects
                nom-part-pgrp(6),               -- parameter group ID partition
                nom-part-sites(7),              -- measurement and body site locations
                nom-part-infrastruct(8),        -- infrastructure elements partition
                nom-part-fef(9),                -- file exchange format partition
                nom-part-ecg-extn(10),          -- electrocardiogram extensions partition
                nom-part-phd-dm(128),           -- disease management
                nom-part-phd-hf(129),           -- health and fitness
                nom-part-phd-ai(130),           -- aging independently
                nom-part-ret-code(255),         -- return codes partition
                nom-part-ext-nom(256),          -- IDs of other nomenclatures and dictionaries
                nom-part-priv(1024)             -- private partition
}
```

A.2.6 Attribute value assertion (AVA) data type

The AVA data type fully specifies the attribute of an object by its attribute ID and its value. As the structure of the value is attribute dependent, the type is specified by ANY DEFINED BY. This data type supports a number of services used to access object attributes (e.g., GET and SET). The attribute ID values are defined for each object type in the Attribute ID column of the object definition tables (i.e., Table 2, Table 5, Table 6, Table 7, Table 8, Table 9, Table 12, Table 13, Table 14, Table 15, and Table 17). The structure used for the attribute-value is defined by the Attribute Type column of the same tables. The AVA data type is defined as follows:

```
--
AVA-Type ::= SEQUENCE {
                attribute-id        OID-Type,        -- This shall come from the nom-part-obj partition
                attribute-value     ANY DEFINED BY attribute-id
}
```

A.2.7 Attribute list data type

Frequently, a list of attribute ID–attribute value pairs is needed. The attribute list data type is a special data type that is provided for this situation and is defined as follows:

```
--
AttributeList ::= SEQUENCE OF AVA-Type
```

A.2.8 Attribute ID list data type

Frequently, a list of attribute IDs is used. The attribute ID list data type is a special type that is provided for convenience and is defined as follows:

```
--
AttributeIdList ::= SEQUENCE OF OID-Type
```

A.2.9 Floating point type (FLOAT-Type) data type

The FLOAT-Type data type is defined to represent numeric values that are not integer in type. The FLOAT-Type is defined as a 32-bit value with 24-bit mantissa and 8-bit exponent. See F.7 for full definition of this data type. This data type is defined as follows:

```
--
-- 32-bit float type; the integer type is a placeholder only
--
FLOAT-Type ::= INT-U32
```

The 32 bits contain an 8-bit signed exponent to base 10, followed by a 24-bit signed integer (mantissa).

Special values are assigned to express the following:

— NaN (not a number) [exponent 0, mantissa +(2**23 –1) → 0x007FFFFF]

— NRes (not at this resolution) [exponent 0, mantissa –(2**23) → 0x00800000]

— + INFINITY [exponent 0, mantissa +(2**23 –2) → 0x007FFFFE]

— – INFINITY [exponent 0, mantissa –(2**23 –2) → 0x00800002]

— Reserved for future use [exponent 0, mantissa –(2**23–1) → 0x00800001]

A.2.10 Short floating point type (SFLOAT-Type) data type

The short floating point type SFLOAT-Type data type is defined to represent numeric values that are not integer in type and have limited resolution. The SFLOAT-Type is defined as a 16-bit value with 12-bit mantissa and 4-bit exponent. See Annex F.7 for full definition of this data type. This data type is defined as follows:

```
--
-- 16-bit float type; the integer type is a placeholder only
--
SFLOAT-Type ::= INT-U16
```

The 16–bit value contains a 4-bit exponent to base 10, followed by a 12-bit mantissa. Each is in twos-complement form.

Special values are assigned to express the following:

— NaN [exponent 0, mantissa +(2**11 –1) → 0x07FF]

— NRes [exponent 0, mantissa –(2**11) → 0x0800]

— + INFINITY [exponent 0, mantissa +(2**11 –2) → 0x07FE]

— – INFINITY [exponent 0, mantissa –(2**11 –2) → 0x0802]

— Reserved for future use [exponent 0, mantissa –(2**11 –1) → 0x0801]

A.2.11 Relative time data type

The relative time data type is a time counter that is used to determine the relative time between events. This data type is used to position events relative to each other. It is defined as follows:

```
--
-- Relative time has a resolution of 125 µs (LSB), which is sufficient for sampling
-- rates up to 8 kHz and spans time periods up to 6.2 days.
-- The value of 0xFFFFFFFF shall be used when an agent is required to send a relative time in an ASN.1
-- structure but does not support a relative time clock.
--
RelativeTime ::= INT-U32
```

Note that the actual time resolution is defined by the agent.

A.2.12 High-resolution relative time data type

The high-resolution relative time data type is a high-resolution time counter that is used to determine the relative time between events. This data type is used to position events relative to each other. It is defined as follows:

```
--
-- High-resolution time has a resolution of 1 µs and can represent time
-- spans of over 584 000 years. Theoretically, this could be modeled as an INT-U64;
-- however, due to limitations in the ASN.1 compilers, embedded devices support
-- for 64-bit integers, and the MDER specifications, an OCTET STRING was
```

```
-- used instead.
--
HighResRelativeTime ::= OCTET STRING (SIZE(8))
```

Note that the agent defines the actual time resolution used.

```
--
-- Absolute time adjust has a resolution of 1/100 of a second and can represent time
-- adjustments of plus or minus 44 505 years. Theoretically, this could be modeled as an INT-I48;
-- however, due to potential limitations in ASN.1 compilers, embedded devices support
-- for 48-bit integers, and the MDER specifications, an OCTET STRING was
-- used instead.
--
AbsoluteTimeAdjust ::= OCTET STRING (SIZE(6))
```

A.2.13 Absolute time data type

The absolute time data type specifies the time of day with a resolution of 1/100 of a second. The hour field shall be reported in 24-hr time notion (i.e., from 0 to 23). The values in the structure shall be encoded using binary coded decimal (i.e., 4-bit nibbles). For example, the year 1996 shall be represented by the hexadecimal value 0x19 in the century field and the hexadecimal value 0x96 in the year field. This format is easily converted to character- or integer-based representations. The absolute time data type is defined as follows:

```
--
AbsoluteTime ::= SEQUENCE {
                century         INT-U8,
                year            INT-U8,
                month           INT-U8,
                day             INT-U8,
                hour            INT-U8,
                minute          INT-U8,
                second          INT-U8,
                sec-fractions   INT-U8          -- 1/100 of a second if available
}
```

Note that the agent defines the actual time resolution used (i.e., if the clock resolution is 1 s, then sec-fractions is always zero). Agents should have a resolution of 1 s or better.

A.2.14 Operational state data type

The operational state data type defines if a certain object or other property is enabled or disabled.

```
--
OperationalState ::= INT-U16 {
                disabled(0),
                enabled(1),
                notAvailable(2)
}
```

A.3 Attribute data types

A.3.1 MDS attributes

```
--
-- SystemModel contains manufacturer name and manufacturer specific model information.
-- While model-number field name suggests a number, there is no requirement that the field
-- contains numeric values. The format of the manufacturer name and model number strings
-- are decided upon by the agent vendor, but shall be printable ASCII.
--
SystemModel ::= SEQUENCE {
                    manufacturer    OCTET STRING,               -- string size shall be even
                    model-number    OCTET STRING                -- string size shall be even
}

--
-- ProductionSpec deals with serial numbers, part numbers, revisions, etc.
-- Note that an agent may have multiple components; therefore, the prod-spec should be an
-- ASCII printable string of the format “spec-type: vendor-specified-str” where spec-type is
-- replaced by the string representation of spec-type. The format of the vendor-specified-str
-- is determined by the vendor.
--
ProductionSpec ::= SEQUENCE OF ProdSpecEntry

ProdSpecEntry ::= SEQUENCE {
                    spec-type       INT-U16 {
                                    unspecified(0),
                                    serial-number(1),
                                    part-number(2),
                                    hw-revision(3),
                                    sw-revision(4),
                                    fw-revision(5),
                                    protocol-revision(6),
                                    prod-spec-gmdn(7)           -- see note on GMDN below
                                    },
                    component-id    PrivateOid,
                    prod-spec       OCTET STRING                -- string size shall be even
}

-- Note: The Global Medical Device Nomenclature (GMDN) is based on ISO 15225 [B15]
-- and was developed under the auspices of CEN TC257 SC1.[7]

--
-- PowerStatus defines whether device is on battery or on mains. Upper bits define the charging
-- state.
--
PowerStatus ::= BITS-16 {
                    onMains(0),
                    onBattery(1),
                    chargingFull(8),
                    chargingTrickle(9),
                    chargingOff(10)
}
```

[7] More information can be found about this technical committee at http://www.nkkn.net/gmdn/gmdnproject.htm.

```
--
-- All measures about the battery are values with their dimensions. See the description
-- of Remaining-Battery-Time in Table 2 for a description of legal units.
--
BatMeasure ::= SEQUENCE {
                value           FLOAT-Type,
                unit            OID-Type        -- from nom-part-dim partition
}
```

A.3.2 Metric attributes

This group contains imported attribute definitions that apply to the numeric, enumeration, and the RT-SA objects.

```
--
-- Status of the measurement
-- The bit values 14 and 15 are used in other ISO/IEEE 11073 standards and shall not be used for a different
-- purpose.
--
MeasurementStatus ::= BITS-16 {
                invalid(0),
                questionable(1),
                not-available(2),
                calibration-ongoing(3),
                test-data(4),
                demo-data(5),
                validated-data(8),              -- relevant, e.g., in an archive
                early-indication(9),            -- early estimate of value
                msmt-ongoing(10)                -- indicates a new measurement is just being taken
                                                -- (episodic)
}
```

A.3.3 Numeric attributes

```
--
-- NuObsValue (Numeric Observed Value) always includes identification, state, and dimension.
--
NuObsValue ::= SEQUENCE {
                metric-id       OID-Type,       -- This code comes from the partition identified in
                                                -- Metric::Type attribute of the numeric object.
                state           MeasurementStatus,
                unit-code       OID-Type,       -- from nom-part-dim dimensions nomenclature
                                                -- partition
                value           FLOAT-Type
}

--
-- Observed value for compound numerics
--
NuObsValueCmp ::= SEQUENCE OF NuObsValue
```

A.3.4 RT-SA attributes

```
--
-- SaSpec describes the sample array.
```

```
--
SaSpec ::= SEQUENCE {
                array-size          INT-U16,        -- number of samples per metric update period
                sample-type         SampleType,
                flags               SaFlags
}

--
-- SampleType describes one sample in the observed value array.
--
SampleType ::= SEQUENCE {
                sample-size         INT-U8,         -- e.g., 8 for 8-bit samples, 16 for 16-bit samples,
                                                    -- shall be divisible by 8
                significant-bits    INT-U8          -- defines significant bits in one sample
                    { signed-samples(255)}          -- if value is 255, the samples
                                                    -- in Simple-Sa-Observed-Value and
                                                    -- lower-scaled-value and upper-scaled-value in
                                                    -- ScaleRangeSpec shall be interpreted as signed
                                                    -- integers in twos-complement form.
}

--
-- SaFlags defines additional wave form properties.
--
SaFlags ::= BITS-16 {
                smooth-curve(0),                    -- for optimum display, use a smoothing algorithm
                delayed-curve(1),                   -- curve is delayed (not real time)
                static-scale(2),                    -- ScaleRangeSpec does not change
                sa-ext-val-range(3)                 -- The nonsignificant bits in a sample are not 0, e.g.,
                                                    -- when they are used for annotations or markers.
                                                    -- The receiver shall apply a bit mask to extract the
                                                    -- significant bits from the sample. If the samples are
                                                    -- signed, the sa-ext-val-range bit shall not be set
                                                    -- (because, by definition, there cannot be
                                                    -- nonsignificant bits in the field).
}

--
-- The scale and range definition attribute describes a mapping between scaled values
-- and absolute values and defines the expected range of absolute values and scaled values.
-- Dependent on the range of the scaled values, multiple attribute types exist.
-- The mapping between sample values and converted absolute values is defined by
-- the Scale-and-Range-Specification formula in 6.3.5.3.
--
ScaleRangeSpec8 ::= SEQUENCE {
                lower-absolute-value        FLOAT-Type,
                upper-absolute-value        FLOAT-Type,
                lower-scaled-value          INT-U8,         -- n.b. interpret as INT-I8
                upper-scaled-value          INT-U8          -- if Sa-Specification attribute
                                                            -- indicates signed samples
}

ScaleRangeSpec16 ::= SEQUENCE {
                lower-absolute-value        FLOAT-Type,
                upper-absolute-value        FLOAT-Type,
                lower-scaled-value          INT-U16,        -- n.b. interpret as INT-I16
```

```
                upper-scaled-value          INT-U16         -- if Sa-Specification attribute
                                                            -- indicates signed samples
}

ScaleRangeSpec32 ::= SEQUENCE {
                lower-absolute-value        FLOAT-Type,
                upper-absolute-value        FLOAT-Type,
                lower-scaled-value          INT-U32,        -- n.b. interpret as INT-I32
                upper-scaled-value          INT-U32         -- if Sa-Specification attribute
                                                            -- indicates signed samples
}
```

A.3.5 Enumeration attributes

```
--
-- EnumObsValue describes the enumeration observed value.
--
EnumObsValue ::= SEQUENCE {
                metric-id       OID-Type,       -- This code comes from the partition defined in the
                                                -- Metric-Id-Partition attribute, if valued. Otherwise,
                                                -- it comes from the same partition as the Type
                                                -- attribute.
                state           MeasurementStatus,
                value           EnumVal         -- supports different value data types
}

-- EnumVal is used to denote different specific observation data types as follows
-- (Note that the type of measurement is coded in the top-level structure EnumObsVal::metric-id):
--
--              enum-obj-id:    used to communicate a metric OID, e.g., as an annotation or
--                              other event defined in the Metric::Type partition
--              enum-text-string: used to communicate a free text string (e.g., a status message)
--              enum-bit-str:   for coding bit string values; the bit string data type shall be
--                              defined separately, e.g., in the nomenclature or in a
--                              device-specific standard
--
-- Other data types defined in ISO/IEEE 11073-10201:2004 [B13] are not included here as they are not
-- relevant for personal health devices.
--
EnumVal ::= CHOICE {
                enum-obj-id     [1] OID-Type,   -- This code comes from the partition defined in the
                                                -- Enum-Observed-Value-Partition attribute, if
                                                -- valued. Otherwise, it comes from the same
                                                -- partition as the Type attribute.

                enum-text-string [2] OCTET STRING,      -- printable ASCII text, size even
                enum-bit-str    [16] BITS-32            -- bit string
}
```

A.3.6 Scanner attributes

None

A.3.7 Configurable scanner attributes

```
--
-- ConfirmMode defines if confirmed event reports or unconfirmed event reports are used.
--
ConfirmMode ::= INT-U16 {
                    unconfirmed(0),
                    confirmed(1)
}
```

A.3.8 Episodic configurable scanner attributes

None

A.3.9 Periodic configurable scanner attributes

None

A.3.10 PM-store and PM-segment attributes

```
--
-- StoSampleAlg describes how samples are derived and averaged.
--
StoSampleAlg ::= INT-U16 {
                    st-alg-nos(0),                      -- not otherwise specified
                    st-alg-moving-average(1),
                    st-alg-recursive(2),
                    st-alg-min-pick(3),
                    st-alg-max-pick(4),
                    st-alg-median(5),
                    st-alg-trended(512),                -- trend values are used
                    st-alg-no-downsampling(1024),       -- means no averaging, this is a real measured sample
                    st-alg-manuf-specific-start(61440), -- start of the reserved manufacturer-specific range
                    st-alg-manuf-specific-end(65535)    -- end of the reserved manufacturer-specific range
}
```

A.4 ACTION-method-related data types

```
--
-- SetTimeInvoke selects the date and time to be set.
--
SetTimeInvoke ::= SEQUENCE {
                    date-time       AbsoluteTime,
                    accuracy        FLOAT-Type      -- accounts for set time (e.g., 2 min error);
                                                    -- value is defined in seconds. This parameter is
                                                    -- inherited from ISO/IEEE 11073-10201:2004
                                                    -- [B13], but not used. Thus, it shall be zero (0).
}

--
-- SegmSelection selects the PM-segments that are subject to the method.
--
SegmSelection ::= CHOICE {
```

```
                all-segments    [1] INT-U16,    -- if this type is chosen to select all segments
                                                -- the actual contents of the field is “do not care”
                                                -- and shall be zero
                segm-id-list    [2] SegmIdList, -- using this list requires that the manager already
                                                -- knows the Instance-Number attributes of the
                                                -- PM-segments, e.g., from a previous
                                                -- Get-Segment-Info method call.
                abs-time-range  [3] AbsTimeRange
                                                -- support of abs-time-range is optional, indicated in
                                                -- the PM-Store-Capab attribute
}

--
-- SegmIdList selects PM-segments by ID.
--
SegmIdList ::= SEQUENCE OF InstNumber

--
-- AbsTimeRange allows selection of PM-segments by time period.
--
AbsTimeRange ::= SEQUENCE {
                from-time       AbsoluteTime,
                to-time         AbsoluteTime
}

--
-- SegmentInfoList returns the object attributes (except the Fixed-Segment-Data) of all
-- selected PM-segment object instances in response to the Get-Segment-Info PM-store method.
-- This is required by the manager to retrieve the dynamic information about the segments.
--
SegmentInfoList ::= SEQUENCE OF SegmentInfo

SegmentInfo ::= SEQUENCE {
                seg-inst-no     InstNumber,
                seg-info        AttributeList
}
```

A.5 Message-related data types

```
ObservationScan ::= SEQUENCE {
                obj-handle      HANDLE,
                attributes      AttributeList
}
```

A.6 Other

```
--
-- TimeProtocolId indicates the time protocols that are supported/used by the device.
--
TimeProtocolId ::= OID-Type -- from the nom-part-infrastruct nomenclature partition
```

A.7 Personal health device protocol frame

The following data type represents the top-level message frame of the personal health device protocol. The data Apdu (encapsulated by the PrstApdu) is interpreted according to this standard as a result of the negotiation contained within the association procedure as described in 8.7.3.1.

MDER encoding rules shall always apply to the structure in A.7.

```
ApduType ::= CHOICE {
        aarq        [57856] AarqApdu,       -- Association Request [0xE200]
        aare        [58112] AareApdu,       -- Association Response [0xE300]
        rlrq        [58368] RlrqApdu,       -- Association Release Request
                                            -- [0xE400]
        rlre        [58624] RlreApdu,       -- Association Release Response
                                            -- [0xE500]
        abrt        [58880] AbrtApdu,       -- Association Abort [0xE600]
        prst        [59136] PrstApdu        -- Presentation APDU [0xE700]
}
```

A.8 Association protocol definitions

MDER encoding rules shall always apply to the structures in A.8.

```
AarqApdu ::= SEQUENCE {
        -- The assoc-version defines the version of the association procedure
        -- used by the agent. The agent shall set exactly one
        -- version bit. If the manager does not understand that version, it shall
        -- reject the association request with rejected-unsupported-assoc-version.
        assoc-version           AssociationVersion,
        data-proto-list         DataProtoList
}

DataProtoList ::= SEQUENCE OF DataProto

-- If the data-proto-id is set to data-proto-id-20601, the data-proto-info shall
-- be filled with a PhdAssociationInformation structure.
-- If the data-proto-id is set to data-proto-id-external, the data-proto-info shall
-- be filled with a ManufSpecAssociationInformation structure.
-- If the data-proto-id is set to data-proto-id-empty, the data-proto-info shall
-- be empty (only used when the AareApdu is a reject).
DataProto ::= SEQUENCE {
        data-proto-id           DataProtoId,
        data-proto-info         ANY DEFINED BY data-proto-id
}

-- All other DataProtoId values are reserved and shall not be used.
DataProtoId ::= INT-U16 {
        data-proto-id-empty(0),         -- shall be used in AareApdu only when result is
                                        -- a rejection
        data-proto-id-20601(20601),     -- indicates exchange protocol follows this standard
        data-proto-id-external(65535)   -- indicates manufacturer specific
                                        -- data protocol UUID is part of
                                        -- the ManufSpecAssociationInformation
}
```

```
-- Association response
AareApdu ::= SEQUENCE {
        result                          AssociateResult,
        selected-data-proto             DataProto
}

-- Release request
RlrqApdu ::= SEQUENCE {
        reason                          ReleaseRequestReason
}

-- Release response
RlreApdu ::= SEQUENCE {
        reason                          ReleaseResponseReason
}

-- Abort
AbrtApdu ::= SEQUENCE {
        reason                          Abort-reason
}

-- Reason for the Abort
-- All unassigned " Abort-reason " values are reserved for future expansion and shall not be used.
Abort-reason ::= INT-U16 {
        undefined(0),
        buffer-overflow(1),
        response-timeout(2),
        configuration-timeout(3)        -- Configuration message not received in timely
                                        -- fashion
}

-- See 8.7.3.2 for a usage description.
-- All unassigned " AssociateResult " values are reserved for future expansion and shall not be used.
AssociateResult ::= INT-U16 {
        accepted(0),
        rejected-permanent(1),
        rejected-transient(2),
        accepted-unknown-config(3),
        rejected-no-common-protocol(4),
        rejected-no-common-parameter(5),
        rejected-unknown(6),
        rejected-unauthorized(7),
        rejected-unsupported-assoc-version(8)
}

-- All unassigned " ReleaseRequestReason " values are reserved for future expansion and shall not be used.
ReleaseRequestReason ::= INT-U16 {
        normal(0),                      -- used when the agent or manager decides to
                                        -- release the association under normal conditions
        no-more-configurations(1),      -- used by the agent when all possible configurations
                                        -- were attempted and the manager
                                        -- rejected them all.
        configuration-changed(2)        -- used by the agent when its configuration changes
                                        -- requiring the agent to release the association. This
                                        -- may be followed by an Association Request with
                                        -- new configuration information.
```

```
}

-- All unassigned " ReleaseResponseReason " values are reserved for future expansion and
-- shall not be used.
ReleaseResponseReason ::= INT-U16 {
                normal(0)
}

-- Association Request DataProto values are mapped to the PhdAssociationInformation.
-- This information is used to announce and negotiate the protocol version, profile, etc.

PhdAssociationInformation ::= SEQUENCE {
                -- The protocolVersion information is used to communicate acceptable versions. When
                -- the agent sends the protocolVersion, it shall set the bit(s) for each version
                -- that it supports. When the manager responds, it shall set a single bit
                -- to indicate the protocol version to be used by both. If there is not
                -- a common protocol version, the manager shall reject the association request
                -- and set the protocolVersion to all zeros.
                protocol-version            ProtocolVersion,
                encoding-rules              EncodingRules,
                nomenclature-version        NomenclatureVersion,
                functional-units            FunctionalUnits,
                system-type                 SystemType,
                system-id                   OCTET STRING,
                dev-config-id               ConfigId,
                data-req-mode-capab         DataReqModeCapab,
                option-list                 AttributeList
}

--
-- Manufacturer-specific association information for a proprietary data protocol
--
ManufSpecAssociationInformation ::= SEQUENCE {
                data-proto-id-ext           UuidIdent,
                data-proto-info-ext         ANY DEFINED BY data-proto-id-ext
}

-- All unassigned " AssociationVersion " bit values are reserved for future expansion and
-- shall be set to zero.
AssociationVersion ::= BITS-32 {
                assoc-version1(0)           -- This bit shall be set if version 1 of the association
                                            -- protocol is supported
}

-- All unassigned " ProtocolVersion " bit values are reserved for future expansion and shall be set to zero.
ProtocolVersion ::= BITS-32 {
                protocol-version1(0)        -- This bit shall be set if version 1 of the data
                                            -- exchange protocol is supported
}

--
--The agent and manager shall always support MDER.
--The agent and manager may negotiate other encoding rules besides MDER.
-- All unassigned " EncodingRules " bit values are reserved for future expansion and shall be set to zero.
--
```

```
EncodingRules ::= BITS-16 {
                    mder(0),                    -- This bit shall be set if MDER supported/selected
                    xer(1),                     -- This bit shall be set if XER supported/selected
                    per(2)                      -- This bit shall be set if PER supported/selected
}

-- All unassigned " NomenclatureVersion " bit values are reserved for future expansion and
-- shall be set to zero.
NomenclatureVersion ::= BITS-32 {               -- values reference a specific nomenclature standard
                    nom-version1(0)             -- This bit shall be set if version 1 is supported
}

-- All unassigned " FunctionalUnits " bit values are reserved for future expansion and shall be set to zero.
FunctionalUnits ::= BITS-32 {
                    fun-units-unidirectional(0),    -- Reserved for future use. This bit shall be set if
                                                    -- the agent is unidirectional
                    fun-units-havetestcap(1),       -- This bit indicates if the device can enter a
                                                    -- test association
                    fun-units-createtestassoc(2)    -- This bit is used to indicate an intention to
                                                    -- form a test association
}

-- All unassigned " SystemType " bit values are reserved for future expansion and shall be set to zero.
SystemType ::= BITS-32 {
                    sys-type-manager(0),
                    sys-type-agent(8)
}

ConfigId ::= INT-U16 {
                    manager-config-response(0),
                    standard-config-start(1),
                    standard-config-end(16383),
                    extended-config-start(16384),
                    extended-config-end(32767),
                    reserved-start(32768),
                    reserved-end(65535)
}
```

A.9 Presentation protocol definitions

MDER encoding rules shall always apply to the structures in A.9.

```
--
-- The OCTET STRING contains the data APDU encoded according to the abstract and transfer syntaxes
-- negotiated at association time.  When the data-proto-id is negotiated to be data-proto-id-20601, the
-- OCTET STRING shall be an encoded version of DataApdu.
--
PrstApdu ::=        OCTET STRING
```

A.10 Data protocol definitions

A.10.1 General

The DataApdu and the related structures in A.10 shall use encoding rules as negotiated during the association procedure as described in 8.7.3.1. The agent and manager shall always support the MDER. The agent and manager may negotiate other encoding rules besides MDER.

A.10.2 Data protocol frame

```
--
-- Combined Remote Operation Primitive Type and Operation Type
-- In the remote operation invoke messages (roiv-*), invoke-id is an opaque handle
-- that allows the sender of the message to identify the associated response message (if any).
-- The sender of roiv-* message shall select a value of invoke-id that enables it to differentiate this message
-- from any other roiv-* messages that have not been retired. Messages are retired either by the
-- reception of a response (rors-*, roer, or rorj) or by exceeding the confirmation timeout value.
-- When a response message (rors-*, roer, or rorj) is returned, the invoke-id from the invocation
-- message shall be copied into the invoke-id of the response. This allows the initiator to match
-- responses to outstanding requests. Since the handle is opaque, the receiver can make no other
-- assumptions about invoke-id. In particular, it can not assume that it will be unique over any sequence of
-- numbers or period of time.
--
DataApdu ::= SEQUENCE {
                invoke-id                       InvokeIDType,
                message                         CHOICE {
                roiv-cmip-event-report          [256] EventReportArgumentSimple, -- [0x0100]
                roiv-cmip-confirmed-event-report [257] EventReportArgumentSimple, -- [0x0101]
                roiv-cmip-get                   [259] GetArgumentSimple, -- [0x0103]
                roiv-cmip-set                   [260] SetArgumentSimple, -- [0x0104]
                roiv-cmip-confirmed-set         [261] SetArgumentSimple, -- [0x0105]
                roiv-cmip-action                [262] ActionArgumentSimple, -- [0x0106]
                roiv-cmip-confirmed-action      [263] ActionArgumentSimple, -- [0x0107]
                rors-cmip-confirmed-event-report [513] EventReportResultSimple, -- [0x0201]
                rors-cmip-get                   [515] GetResultSimple, -- [0x0203]
                rors-cmip-confirmed-set         [517] SetResultSimple, -- [0x0205]
                rors-cmip-confirmed-action      [519] ActionResultSimple, -- [0x0207]
                roer                            [768] ErrorResult, -- [0x0300]
                rorj                            [1024] RejectResult -- [0x0400]
                }
}

-- The sender should limit the number of messages outstanding simultaneously.
-- In fact, the receiver might not be able to handle more than one message at a time.
InvokeIDType ::= INT-U16

-- At any point, if a DataApdu invoked action (roiv-*) results in an error, the receiver sends
-- back an ErrorResult. The invokeID is used to determine which invocation resulted in an
-- error condition. The error-value shall be filled in with an error value from the RoerErrorValue list
-- below. The parameter is filled in with further information if warranted by the error-value. The use of
-- the parameter value is defined in the comments found in RoerErrorValue.
ErrorResult ::= SEQUENCE {
                error-value         RoerErrorValue,
```

```
                parameter           ANY DEFINED BY error-value
}

-- All unassigned " RoerErrorValue " values are reserved for future expansion and shall not be used.
-- Note that ISO/IEEE 11073-20101:2004 [B14] defines a number of RoerErrorValue values that are not
-- defined in this standard. For consistency, numbering of the RoerErrorValue skips any value already
-- defined in ISO/IEEE 11073-20101:2004.
RoerErrorValue ::= INT-U16 {
                -- no-such-object-instance is returned when referencing an illegal handle or when there
                -- is an attempt to access any object other than the MDS before the configuration
                -- is agreed, i.e., agent and manager are not in the operating state.
                no-such-object-instance(1),
                -- no-such-action is returned when the action command is illegal
                no-such-action(9),
                -- invalid-object-instance is returned when object exists but the command
                -- is illegal for that object type (e.g., Get on any object except MDS or PM-store)
                invalid-object-instance(17),
                -- protocol-violation is returned when there has been a protocol violation (e.g., APDU
                -- exceeds maximum size)
                protocol-violation(23) ,
                -- not-allowed-by-object is returned when an action is attempted on an object
                -- but the object did not allow the action
                -- The higher layer may report the reason for aborting the action as an OID-Type
                -- in the parameter field using a return code taken from the return code partition
                not-allowed-by-object(24),
                -- action-timed-out is returned when an action is aborted before completion or when to
                -- complete the action would exceed the currently defined timeout value.
                -- The higher layer may report the reason for aborting the action as an OID-Type
                -- in the parameter field using a return code taken from the return code partition
                action-timed-out(25),
                -- action-aborted is returned when an action has been aborted due to reasons in the
                -- higher layers (e.g., storage capacity exceeded).
                -- The higher layer may report the reason for aborting the action as an OID-Type
                -- in the parameter field using a return code taken from the return code partition
                action-aborted(26)
}

-- The RejectResult shall be used when a message is rejected.
RejectResult ::= SEQUENCE {
                problem             RorjProblem
}

-- All unassigned " RorjProblem " values are reserved for future expansion and shall not be used.
RorjProblem ::= INT-U16 {
                -- unrecognized-apdu is returned if the DataApdu is unrecognized,
                unrecognized-apdu(0),
                -- badly-structured-apdu is returned when the receiver is unable to
                -- understand the DataApdu due to its structure (or lack thereof)
                -- (e.g., incorrect data lengths)
                badly-structured-apdu(2),
                -- unrecognized-operation is sent when the operation being requested
                -- is not understood by the receiver
                unrecognized-operation(101),
                -- resource-limitation is sent when the receiver cannot handle the
                -- message due to limited resources.
                resource-limitation(103),
```

```
                -- unexpected-error covers error conditions where there is not a
                -- more specific error code defined
                unexpected-error(303)
}
```

A.10.3 EVENT REPORT service

```
-- For event reports defined in this standard, obj-handle shall either be 0 to represent the MDS object
-- or a handle representing a scanner or PM-store object.
-- If the agent does not support RelativeTime (as indicated by the mds-time-capab-relative-time
-- bit in MdsTimeCapState), it shall set the event-time to 0xFFFFFFFF. Managers shall
-- ignore the event-time if the agent reports that it does not support RelativeTime.
-- For the event-types defined in Table 4, Table 11, Table 16, and Table 18, the
-- corresponding event-info structure shall be used. Accordingly, event-info will be one of
-- ConfigReport, ScanReportInfoFixed, ScanReportInfoVar, ScanReportInfoMPFixed,
-- ScanReportInfoMPVar, ScanReportInfoGrouped, ScanReportInfoMPGrouped,
-- or SegmentDataEvent
EventReportArgumentSimple ::= SEQUENCE {
                obj-handle      HANDLE,
                event-time      RelativeTime,
                event-type      OID-Type,       -- From the nom-part-obj partition
                                                -- Subpartition NOTI (MDC_NOTI_*)
                event-info      ANY DEFINED BY event-type
}

-- For event reports defined in this standard, obj-handle shall be either 0 to represent the MDS object
-- or a handle representing a scanner or PM-store object.
-- The event-type of the result shall be a copy of the event-type from the invocation.
-- For the event-types defined in Table 4, Table 11, Table 16, and Table 18, the corresponding
-- event-reply-info shall be used. Accordingly event-reply-info will be empty, ConfigReportRsp,
-- or SegmentDataResult.
EventReportResultSimple ::= SEQUENCE {
                obj-handle      HANDLE,
                currentTime     RelativeTime,
                event-type      OID-Type,       -- From the nom-part-obj partition
                                                -- Subpartition NOTI (MDC_NOTI_*)
                event-reply-info ANY DEFINED BY event-type
}
```

A.10.4 GET service

```
-- For GET requests defined in this standard, obj-handle shall either be 0 to represent the MDS object
-- or a handle representing a PM-store object.
-- The attribute-id-list shall be left empty to query for all attributes of the MDS or PM-store object.
-- Alternatively, specific attributes of an object may be queried by listing the desired
-- Attribute IDs found in Table 2 or Table 9.
GetArgumentSimple ::= SEQUENCE {
                obj-handle      HANDLE,
                attribute-id-list AttributeIdList
}

-- For GET responses defined in this standard, obj-handle shall match the one in the corresponding request.
-- The attribute-list contains all the requested attributes using the variable format.
-- If a requested attribute ID does not exist within the MDS object, it shall not
-- be returned in the attribute-list.
GetResultSimple ::= SEQUENCE {
```

```
                obj-handle          HANDLE,
                attribute-list      AttributeList
}

TypeVerList ::= SEQUENCE OF TypeVer


-- Since the type shall come from ISO/IEEE 11073-10101 [B12], communication
-- nom-part-infrastruct partition, subpartition DEVspec, a simple OID-Type is used rather
-- than a TYPE.
-- The individual IEEE 11073-104zz specializations define which specification is classified
-- as version 1, 2, …, and so on; thus, version 3 may correspond to specification version 1.5.
TypeVer ::= SEQUENCE {
                type                OID-Type,
                version             INT-U16
}
```

A.10.5 SET service

```
-- For SETs defined in this standard, obj-handle shall be the value of a handle representing a scanner object.
SetArgumentSimple ::= SEQUENCE {
                obj-handle          HANDLE,
                modification-list   ModificationList
}

ModificationList ::= SEQUENCE OF AttributeModEntry

AttributeModEntry ::= SEQUENCE {
                modify-operator     ModifyOperator,
                attribute           AVA-Type
}

-- All unassigned " ModifyOperator " values are reserved for future expansion and shall not be used.
ModifyOperator ::= INT-U16 {
                replace(0),
                addValues(1),       -- used for modifying the values contained in list-like data types
                removeValues(2),    -- used for modifying the values contained in list-like data types
                setToDefault(3)
}

--
-- The obj-handle shall be set to the value received in the SetArgumentSimple.
-- The attribute-list shall contain each attribute-id that was modified and return
-- the new value of the attribute. Normally, this is the value from the Set
-- command; however, it is possible that, due to rounding conditions or an
-- error condition, the returned value could differ from the requested value.
SetResultSimple ::= SEQUENCE {
                obj-handle          HANDLE,
                attribute-list      AttributeList
}
```

A.10.6 ACTION service

```
-- For action requests defined in this standard, obj-handle shall either be 0 to represent the MDS object or
-- a handle representing a PM-store object.
-- For the action-types defined in Table 3 and Table 10, the corresponding action-info-args
```

```
-- structures shall be used. Accordingly, action-info-args will be one of DataRequest,
-- SetTimeInvoke, SegmSelection, or TrigSegmDataXferReq.
ActionArgumentSimple ::= SEQUENCE {
                obj-handle          HANDLE,
                action-type         OID-Type,       -- From the nom-part-obj partition
                                                    -- Subpartition ACT (MDC_ACT_*)

                action-info-args    ANY DEFINED BY action-type
}

-- For action responses defined in this standard, obj-handle shall match the one in the
-- corresponding request.
-- The action-type shall be copied from the invocation message action-type.
-- For the action-types defined in Table 3 and Table 10, the resulting action-info-args
-- shall be used. Accordingly, action-info-args will be empty, DataResponse,
-- SegmentInfoList, or TrigSegmDataXferRsp.
ActionResultSimple ::= SEQUENCE {
                obj-handle          HANDLE,
                action-type         OID-Type,       -- From the nom-part-obj partition
                                                    -- Subpartition ACT (MDC_ACT_*)
                action-info-args    ANY DEFINED BY action-type
}
```

A.11 Data types for new object attributes and object services

A.11.1 General data types

```
AttrValMap ::= SEQUENCE OF AttrValMapEntry

AttrValMapEntry ::= SEQUENCE {
                attribute-id        OID-Type,   -- This comes from the nom-part-obj partition
                attribute-len       INT-U16
}
```

A.11.2 MDS-related data types

```
UuidIdent ::= OCTET STRING(SIZE(16))

-- time-sync-accuracy allows an agent to report how closely synchronized its clock is with
-- respect to the clock sync master when time synchronization is used.
MdsTimeInfo ::= SEQUENCE {
                mds-time-cap-state          MdsTimeCapState,
                time-sync-protocol          TimeProtocolId,  -- this is a nomenclature code from
                                                             -- nom-part-infrastruct partition
                time-sync-accuracy          RelativeTime,    -- 0xFFFFFFFF if unknown
                                                             -- 0 if better than 1/8 ms
                time-resolution-abs-time    INT-U16,         -- Resolution of the agent's
                                                             -- absolute time clock.
                                                             -- 0 if unknown; otherwise,
                                                             -- the number of 1/100 s
                                                             -- that elapse with each clock
                                                             -- increment. For example, if an
                                                             -- agent has a clock that clicks at
                                                             -- 1 s intervals, this value
```

```
                                                        -- would be 100.
        time-resolution-rel-time        INT-U16,        -- Resolution of the agent's
                                                        -- relative time clock. 0 if
                                                        -- unknown; otherwise, the number
                                                        -- of 125 µs that elapse
                                                        -- with each clock increment. For
                                                        -- example, if an agent has a clock
                                                        -- that clicks at 1 s intervals,
                                                        -- this value would be 8000.
        time-resolution-high-res-time   INT-U32         -- Resolution of the agent's
                                                        -- high-resolution time clock.
                                                        -- 0 if unknown; otherwise, the
                                                        -- the number of microseconds
                                                        -- that elapse with each clock
                                                        -- increment. For example, if an
                                                        -- agent has a clock that clicks
                                                        -- at 1 s intervals, this value
                                                        -- would be 1 000 000.
}

-- All unassigned " MdsTimeCapState " bit values are reserved for future expansion and shall be set to zero.
MdsTimeCapState ::= BITS-16 {
        mds-time-capab-real-time-clock(0),              -- device supports an internal RTC
        mds-time-capab-set-clock(1),                    -- device supports Set Time Action
        mds-time-capab-relative-time(2),                -- device supports RelativeTime
        mds-time-capab-high-res-relative-time(3),       -- device supports
                                                        -- HighResRelativeTime
        mds-time-capab-sync-abs-time(4),                -- device syncs AbsoluteTime
        mds-time-capab-sync-rel-time(5),                -- device syncs RelativeTime
        mds-time-capab-sync-hi-res-relative-time(6),    -- device syncs HiResRelativeTime
        mds-time-state-abs-time-synced(8),              -- AbsoluteTime is synced
        mds-time-state-rel-time-synced(9),              -- RelativeTime is synced
        mds-time-state-hi-res-relative-time-synced(10), -- HiResRelativeTime is synced
        mds-time-mgr-set-time(11)                       -- manager is encouraged to
                                                        -- set the time
}

-- ************
-- A list of various regulatory and certification compliance items to which the agent claims adherence.
-- ************
RegCertDataList ::= SEQUENCE OF RegCertData

RegCertData ::= SEQUENCE {
        auth-body-and-struc-type        AuthBodyAndStrucType,
        auth-body-data                  ANY DEFINED BY auth-body-and-struc-type
}

AuthBodyAndStrucType ::= SEQUENCE {
        auth-body                       AuthBody,
        auth-body-struc-type            AuthBodyStrucType
}

-- All unassigned " AuthBody " values are reserved for future expansion and shall not be used.
AuthBody ::= INT-U8 {
        auth-body-empty(0),
        auth-body-ieee-11073(1),
```

```
                    auth-body-continua(2),
                    auth-body-experimental(254),
                    auth-body-reserved(255)
}
--
-- Some other possible/expected authoritative bodies
-- auth-body-eu(),
-- auth-body-ieee(),
-- auth-body-iso(),
-- auth-body-us-fda(),
-- specific values will be assigned when a given authoritative body
--      assigns its first AuthBodyStrucType for a specific
--      auth-body-data.


-- AuthBodyStrucType is controlled and assigned by the authoritative body
AuthBodyStrucType ::= INT-U8
```

A.11.3 Metric-related data types

```
--
-- SupplementalTypeList provides an extensible mechanism to list additional information about an object.
-- This can hold information such as the location of the sensor or the responsiveness of the object.
--
SupplementalTypeList ::= SEQUENCE OF TYPE


--
-- The Metric Spec Small attribute is an abbreviated MetricSpec attribute as defined in ISO/IEEE
-- 11073-10201:2004 [B13]. It defines availability, periodicity, and category of the measurement.
-- If the metric corresponds to default values (no bits set), the attribute is not required.
-- All unassigned " MetricSpecSmall " bit values are reserved for future expansion and shall be set to zero.
--
MetricSpecSmall ::= BITS-16 {
                    mss-avail-intermittent(0),      -- value is available only intermittently
                    mss-avail-stored-data(1),       -- Agent may store and send multiple historical
                                                    -- values (e.g., a weighing scale stores up
                                                    -- to 25 values)
                    mss-upd-aperiodic(2),           -- value is sent only aperiodically
                                                    -- (e.g., when changed)
                    mss-msmt-aperiodic(3),          -- the measurement is aperiodic
                    mss-msmt-phys-ev-id(4),         -- the measurement is a physiological trigger only
                                                    -- (e.g., to mark the detection of a heart beat)
                    mss-msmt-btb-metric(5),         -- the measurement is beat-to-beat or breath-to-breath
                    mss-acc-manager-initiated (8),  -- the object value can be accessed by manager-
                                                    -- initiated measurement data transmission
                    mss-acc-agent-initiated(9),     -- the object value is updated using agent-initiated
                                                    -- measurement data transmission
                    -- NOTES regarding the usage of the following mss-cat-* bits
                    -- For automatically acquired measurements, neither the mss-cat-setting nor the
                    -- mss-cat-calculation bits are set. The metric represents a normal, regular measured
                    -- value. This implies that, for automatically acquired measurements provided by an
                    -- agent, none of the mss-cat-* bits are set (default).
                    mss-cat-manual(12),             -- if this bit is set, the metric is acquired manually
                                                    -- (e.g., a person manually entered the value).
                                                    -- If this bit is not set, the metric is acquired
```

```
                                                -- automatically (e.g., the device measures the value)
                mss-cat-setting(13),            -- If this bit is set, the metric represents a device
                                                -- setting. This may be a manually or automatically
                                                -- set value, as reported by the mss-cat-manual bit.
                mss-cat-calculation(14)         -- If this bit is set, the metric represents a calculated
                                                -- value. This may be a manually or automatically
                                                -- calculated value, as reported by the
                                                -- mss-cat-manual bit. Calculated values are
                                                -- derived from automatically acquired measurements
                                                -- and/or manually entered values.
}

-- This attribute is partly inherited from ISO/IEEE 11073-10201:2004 [B13], but enhanced by
-- value ms-struct::fix-comp-no.
--
MetricStructureSmall ::= SEQUENCE {
                ms-struct INT-U8 {
                                ms-struct-simple(0),
                                ms-struct-compound(1),          -- multiple observed values,
                                                                -- same dynamic context
                                ms-struct-reserved(2),          -- for ISO/IEEE 11073-10201:2004
                                ms-struct-compound-fix(3)       -- similar to compound(1) but the
                                                                -- compound observed value array
                                                                -- size shall not be dynamic
                                                                -- during an association
                },
                ms-comp-no INT-U8               -- maximum number of components/elements in
                                                -- compound observed value, 0 if ms-struct is set to
                                                -- ms-struct-simple
}

-- This attribute defines a list of MetricIds.
--
MetricIdList ::= SEQUENCE OF OID-Type

--
-- The EnumPrintableString is the data type to report Enumeration Observed Values in the form of
-- ASCII printable strings.
--
EnumPrintableString ::= OCTET STRING            -- string size shall be even

PersonId ::= INT-U16 {
                unknown-person-id(65535)        -- 0xFFFF
}
```

A.11.4 Scanner-related data types

```
HandleAttrValMap ::= SEQUENCE OF HandleAttrValMapEntry

HandleAttrValMapEntry ::= SEQUENCE {
                obj-handle      HANDLE,
                attr-val-map    AttrValMap
}

HANDLEList ::= SEQUENCE OF HANDLE
```

A.11.5 MDS services

```
-- The following definitions support the above definitions of EventReportArgumentSimple
-- and ActionArgumentSimple.
--
-- The Scan Report Info types are utilized as the result data types for the various
-- MDS-Dynamic-Data-Update* family of events (see 6.3.2.5 for more detail).
--

-- The ScanReport* definitions are used when reporting information about measurements that were
-- sampled. There are two vectors: A) single person or multiple person and B) variable format,
-- fixed format, or grouped format. Combinations of these vectors lead to the six top-level definitions:
-- ScanReportInfoVar, ScanReportInfoFixed, ScanReportInfoGrouped,
-- ScanReportInfoMPVar, ScanReportInfoMPFixed, and ScanReportInfoMPGrouped.
-- The SEQUENCE OF ObservationScan or ObservationScanFixed may contain multiple instances
-- of the same handle as long as there is a time stamp to distinguish between the instances.
-- In all cases, scan-report-no shall be initialized to zero at association time and monotonically
-- increasing by one until roll-over occurs.
-------------------------------------------------------------------------------------------------------------
ScanReportInfoVar ::= SEQUENCE {
                data-req-id         DataReqId,
                scan-report-no      INT-U16,        -- counter for detection of missing scan reports
                obs-scan-var        SEQUENCE OF ObservationScan
}

-------------------------------------------------------------------------------------------------------------
ScanReportInfoFixed ::= SEQUENCE {
                data-req-id         DataReqId,
                scan-report-no      INT-U16,        -- counter for detection of missing scan reports
                obs-scan-fixed      SEQUENCE OF ObservationScanFixed
}

ObservationScanFixed ::= SEQUENCE {
                obj-handle          HANDLE,         -- unique identification of metric object
                obs-val-data        OCTET STRING-- observed value data defined by obj-handle
}
-------------------------------------------------------------------------------------------------------------
-- obs-scan-grouped is a SEQUENCE OF so episodic measurements can combine more than
-- one report into a single scan report. Periodic reports should not need to place more than one
-- report in a single ScanReport.
ScanReportInfoGrouped ::= SEQUENCE {
                data-req-id         INT-U16,
                scan-report-no      INT-U16,        -- counter for detection of missing scan reports
                obs-scan-grouped SEQUENCE OF ObservationScanGrouped
}

ObservationScanGrouped ::= OCTET STRING         -- The format is defined by HandleAttrValMap

-------------------------------------------------------------------------------------------------------------
ScanReportInfoMPVar ::= SEQUENCE {
                data-req-id         DataReqId,
                scan-report-no      INT-U16,        -- counter for detection of missing scan reports
                scan-per-var        SEQUENCE OF ScanReportPerVar
}

DataReqId ::= INT-U16 {
```

```
                    data-req-id-manager-initiated-min(0),           -- 0x0000
                    data-req-id-manager-initiated-max(61439),       -- 0xEFFF
                    -- Values between data-req-id-manager-initiated-min and
                    -- data-req-id-manager-initiated-max, inclusive, shall be used in
                    -- manager-initiated measurement data transmission.
                    --
                    data-req-id-agent-initiated(61440)              -- 0xF000
                    -- data-req-id-agent-initiated shall be used in agent-initiated measurement
                    -- data transmission.
                    --
                    -- Values between 0xF001 and 0xFFFF, inclusive, are reserved.
}

--
-- The value used for person-id is vendor determined (i.e., if an agent has two buttons
-- to distinguish between two people, the agent may use ID 1 and 2 or ID 35 and 97).
-- The process of mapping this ID to a specific person is outside the scope of this
-- standard.
--
ScanReportPerVar ::= SEQUENCE {
                    person-id           PersonId,
                    obs-scan-var        SEQUENCE OF ObservationScan
}

-------------------------------------------------------------------------------------------------------------
ScanReportInfoMPFixed ::= SEQUENCE {
                    data-req-id         DataReqId,
                    scan-report-no      INT-U16,        -- counter for detection of missing scan reports
                    scan-per-fixed      SEQUENCE OF ScanReportPerFixed
}

ScanReportPerFixed ::= SEQUENCE {
                    person-id           PersonId,
                    obs-scan-fixed      SEQUENCE OF ObservationScanFixed
}

-------------------------------------------------------------------------------------------------------------
ScanReportInfoMPGrouped ::= SEQUENCE {
                    data-req-id         INT-U16,
                    scan-report-no      INT-U16,        -- counter for detection of missing scan reports
                    scan-per-grouped SEQUENCE OF ScanReportPerGrouped

}

ScanReportPerGrouped ::= SEQUENCE {
                    person-id           PersonId,
                    obs-scan-grouped ObservationScanGrouped
}

-------------------------------------------------------------------------------------------------------------
-- The ConfigReport definition is used when reporting an agent's configuration to a manager (see
-- Table 4)
ConfigReport ::= SEQUENCE {
                    config-report-id    ConfigId,
                    config-obj-list     ConfigObjectList
}
```

```
ConfigObjectList ::= SEQUENCE OF ConfigObject

ConfigObject ::= SEQUENCE {
        obj-class       OID-Type,       -- From the nom-part-obj partition
                                        -- Subpartition MOC/BASE (MDC_MOC_VMD_*)
        obj-handle      HANDLE,
        attributes      AttributeList
}

ConfigReportRsp ::= SEQUENCE {
        config-report-id  ConfigId,
        config-result     ConfigResult
}

-- All unassigned " ConfigResult " values are reserved for future expansion and shall not be used.
ConfigResult ::= INT-U16 {
        accepted-config(0),
        unsupported-config(1),
        standard-config-unknown(2)
}

DataRequest ::= SEQUENCE {
        data-req-id                     DataReqId,      -- Allows differentiation of
                                                        -- responses for multiple data
                                                        -- requests (if the
                                                        -- device allows for multiple
                                                        -- simultaneous data requests).
                                                        -- Mirrored back in
                                                        -- ScanReportInfo* data-req-id
        data-req-mode                   DataReqMode,    -- Defines the mode by setting one
                                                        -- or more bits.
        data-req-time                   RelativeTime,   -- Tells how long the agent is
                                                        -- allowed to transmit data.
                                                        -- This is used only for
                                                        -- data-req-mode-time-period.
        data-req-person-id              INT-U16,        -- 0xFFFF all persons available
        data-req-class                  OID-Type,       -- From the nom-part-obj partition
                                                        -- Subpartition MOC/BASE
                                                        -- (MDC_MOC_VMD_*)
        data-req-obj-handle-list        HANDLEList
}

-- All unassigned " DataReqMode " bit values are reserved for future expansion and shall be set to zero.
DataReqMode ::= BITS-16 {
        data-req-start-stop(0),         -- start data request: 1 | stop data request: 0
        data-req-continuation(1),       -- continuation of a timed data request.
                                        -- Set to 1 to extend the time allocated to a data
                                        -- transfer. If this is set to 1, all other bits shall
                                        -- be ignored, and the settings from the initial
                                        -- start command shall be used.
        -- exactly one of the following data-req-scope-* bits shall be set
        data-req-scope-all(4),
        data-req-scope-class(5),
        data-req-scope-handle(6),
        -- exactly one of the following data-req-mode-* bits shall be set
```

```
                data-req-mode-single-rsp(8),        -- response is directly embedded in DataResponse
                data-req-mode-time-period(9),       -- time limited data request with
                                                    -- responses as event reports. The time period
                                                    -- is specified in data-req-time in DataRequest.
                data-req-mode-time-no-limit(10),    -- time unlimited data request with
                                                    -- responses as event reports
                data-req-person-id(12)
}

DataReqModeCapab ::= SEQUENCE {
                data-req-mode-flags                 DataReqModeFlags,
                data-req-init-agent-count INT-U8,   -- maximum number of parallel agent initiated
                                                    -- data requests/ flows. Shall currently be
                                                    -- set only to 0 or 1.
                data-req-init-manager-count INT-U8 -- maximum number of parallel manager
                                                    -- initiated data requests
}

-- All unassigned " DataReqModeFlags " bit values are reserved for future expansion and
-- shall be set to zero.
DataReqModeFlags ::= BITS-16 {                      -- this field is used in the association to flag
                                                    -- data request capabilities
                data-req-supp-stop(0),              -- supports stopping a running data request
                data-req-supp-scope-all(4),         -- supports requesting all objects
                data-req-supp-scope-class(5),       -- supports requesting objects based on object class
                data-req-supp-scope-handle(6),      -- supports requesting objects based on object handle
                data-req-supp-mode-single-rsp(8),   -- supports single response
                data-req-supp-mode-time-period(9), -- supports time limited data request
                data-req-supp-mode-time-no-limit(10), -- supports time unlimited data request
                data-req-supp-person-id(11),
                data-req-supp-init-agent(15)        -- agent uses agent-initiated data requests/flows
}

-- DataResponse is returned as a result of an MDS-Data-Request (see Table 3). However, the event-type
-- and event-info fields are filled in using the same parameters as found in MDS object events. See Table 4
-- for the legal event-type values and the corresponding event-info
-- structure; however, for this usage, ConfigReport shall not be used. Thus, event-info is
-- one of ScanReportInfoFixed, ScanReportInfoVar, ScanReportInfoMPFixed, or ScanReportInfoMPVar.
DataResponse ::= SEQUENCE {
                rel-time-stamp  RelativeTime,       -- set to 0xFFFFFFFF if RelativeTime not supported
                data-req-result DataReqResult,
                event-type      OID-Type,           -- event-type and event-info are only
                                                    -- in case of data-req-mode-single-rsp,
                                                    -- otherwise event-type shall be 0 and
                                                    -- event-info.length = 0
                                                    -- From the nom-part-obj partition
                                                    -- Subpartition NOTI (MDC_NOTI_*)
                event-info      ANY DEFINED BY event-type
}

-- The values in DataReqResult are used in a DataResponse data-req-result field. This is returned
-- in response to a DataRequest. The agent shall return data-req-result-no-error if the request
-- was successful. Otherwise, one of the defined errors shall be returned.
-- All unassigned " DataReqResult " values are reserved for future expansion and shall not be used.
DataReqResult ::= INT-U16 {
                data-req-result-no-error(0),
```

```
                data-req-result-unspecific-error(1),
                -- The following error codes are returned when the manager request contains
                -- a DataReqMode that is not supported by the agent.
                data-req-result-no-stop-support(2),
                data-req-result-no-scope-all-support(3),
                data-req-result-no-scope-class-support(4),
                data-req-result-no-scope-handle-support(5),
                data-req-result-no-mode-single-rsp-support(6),
                data-req-result-no-mode-time-period-support(7),
                data-req-result-no-mode-time-no-limit-support(8),
                data-req-result-no-person-id-support(9),
                -- The following error codes are returned when the manager request contains
                -- unknown values in the supporting fields (e.g., data-req-person-id).
                data-req-result-unknown-person-id(11),
                data-req-result-unknown-class(12),
                data-req-result-unknown-handle(13),
                -- The following note a condition where the manager set more than one of the
                -- scope or mode bits.
                data-req-result-unsupp-scope(14),          -- unsupported scope bits set
                data-req-result-unsupp-mode(15),           -- unsupported mode bits set

                data-req-result-init-manager-overflow(16), -- manager has tried to establish more than
                                                           -- data-req-init-manager-count flows
                data-req-result-continuation-not-supported(17), -- manager has attempted to continue
                                                           -- a data transfer that is not running in
                                                           -- timed mode
                data-req-result-invalid-req-id(18)         -- manager has attempted to continue
                                                           -- a data transfer on a nonexistent
                                                           -- data-req-id.
}
```

A.11.6 Scanner services

See A.11.5 for MDS services type definitions that are reused for the scanner services, namely

ScanReportInfoVar
ScanReportInfoFixed
ScanReportInfoGrouped
ScanReportInfoMPVar
ScanReportInfoMPFixed
ScanReportInfoMPGrouped

A.11.7 Numeric related data types

```
-- A simple numeric observed value is represented just by the floating point value.
--
SimpleNuObsValue ::= FLOAT-Type

-- A list type of SimpleNuObsValue
--
SimpleNuObsValueCmp ::= SEQUENCE OF SimpleNuObsValue

-- In many cases, the basic numeric observed value can be expressed with a smaller floating point value.
--
BasicNuObsValue ::= SFLOAT-Type
```

```
-- A list type of BasicNuObsValue
--
BasicNuObsValueCmp ::= SEQUENCE OF BasicNuObsValue
```

A.11.8 PM-store and PM-segment related data types

```
--
-- The PM-Store-Capab attribute defines specific static capabilities and properties of the PM-store object
-- instance. The default value of this attribute is 0 (no bits set).
-- All unassigned " PmStoreCapab " bit values are reserved for future expansion and shall be set to zero.
--
PmStoreCapab ::=BITS-16 {
                pmsc-var-no-of-segm(0),         -- indicates that the number of PM-segments
                                                -- contained in this PM-store is dynamic and may
                                                -- change
                pmsc-epi-seg-entries(4),        -- Some/ all PM-segments contain
                                                -- episodic/aperiodic entries and therefore have
                                                -- to contain explicit time stamp information
                pmsc-peri-seg-entries(5),       -- Some/all PM-segments contain periodically
                                                -- sampled entries and therefore the PM-segment
                                                -- or PM-store shall support the
                                                -- Sample-Period attribute
                pmsc-abs-time-select(6),        -- PM-segments in the SegmSelection data type can
                                                -- be selected by defining an abs-time-range
                pmsc-clear-segm-by-list-sup(7), -- clearing a list of segments is supported
                pmsc-clear-segm-by-time-sup(8), -- clearing segments by time range is supported
                pmsc-clear-segm-remove(9),      -- if this bit is set, the agent will completely remove
                                                -- the specified PM-segment instance as part of the
                                                -- Clear-Segment method. If this bit is not set, it will
                                                -- just remove all entries from the specified
                                                -- PM-segment.
                pmsc-multi-person(12)           -- The PM-store supports PM-segment for more
                                                -- than one person
}

--
-- All entries in the segment shall follow the format defined by this attribute. First, the optional header
-- shall follow the description in segm-entry-header. This allows each entry in the segment to be preceded
-- by an optional header (e.g., for time stamp information) that is applicable to all elements in an entry.
-- Next, the elements shall follow the format and order described in segm-entry-elem-list.
-- An element typically represents a measurement. For each element, the stored data is defined in the form
-- of an attribute-value map, in the same way as metric objects.
--
PmSegmentEntryMap ::= SEQUENCE {
                segm-entry-header       SegmEntryHeader,        -- defines optional elements in front
                                                                -- of each entry
                segm-entry-elem-list    SegmEntryElemList
}

--
-- The following bit string defines optional data items that are in front of each segment entry.
-- Multiple data items are definable. In this case, the data item with the lower bit number shall come
-- in front of items with higher bit numbers. The header allows definition of data items that are common
-- to all elements in the entry. If all bits are zero, the segment entry event report shall begin with data
-- from the first element.
-- All unassigned " SegmEntryHeader " bit values are reserved for future expansion and shall be set to zero.
```

```
-- If any bits are set to one beyond the expected bits (e.g., a new bit was added in a later version),
-- the data shall not be retrieved since the offset to the first data element cannot be calculated.
--
SegmEntryHeader ::= BITS-16 {
                    seg-elem-hdr-absolute-time(0),      -- entry preceded by absolute time
                                                        -- (data type AbsoluteTime)
                    seg-elem-hdr-relative-time(1),      -- entry preceded by relative time
                                                        -- (data type RelativeTime)
                    seg-elem-hdr-hires-relative-time(2) -- entry preceded by high resolution relative time
                                                        -- (data type HighResRelativeTime)
}

SegmEntryElemList ::= SEQUENCE OF SegmEntryElem

--
-- SegmEntryElem shall reference a metric object instance in the agent configuration
-- using its handle value. This referenced object shall exist in the agent
-- configuration, and the metric-type and class-id shall be equal to the corresponding attributes of the
-- referenced metric object.
--
SegmEntryElem ::= SEQUENCE {
                    class-id        OID-Type,       -- contains nomenclature code from OO nom-part-obj
                                                    -- partition defining the object class (e.g., numeric)
                    metric-type     TYPE,           --specific static TYPE of the stored element
                    handle          HANDLE,         -- handle of referenced object
                    attr-val-map    AttrValMap      -- attribute value map describing the stored data
}

--
-- Request to start the transfer of the specified segment
--
TrigSegmDataXferReq ::= SEQUENCE {
                    seg-inst-no     InstNumber
}

TrigSegmDataXferRsp ::= SEQUENCE {
                    seg-inst-no         InstNumber,
                    trig-segm-xfer-rsp  TrigSegmXferRsp
}

-- All unassigned " TrigSegmXferRsp " values are reserved for future expansion and shall not be used.
TrigSegmXferRsp ::= INT-U16 {
                    tsxr-successful(0),                 -- Agent will start transfer of segment
                    tsxr-fail-no-such-segment(1),       -- segment ID not found
                    tsxr-fail-clear-in-process(2),      -- the storage media is currently being cleared. No
                                                        -- access is currently possible.
                    tsxr-fail-segm-empty(3),            -- the segment being requested is empty
                    tsxr-fail-not-otherwise-specified(512)
}


--
-- the SegmentDataEvent
--
-- Notes:
--     - the agent shall transfer all segment entries in order, first entry first (first in first out).
```

```
--

SegmentDataEvent ::= SEQUENCE {
        segm-data-event-descr     SegmDataEventDescr,
        segm-data-event-entries   OCTET STRING        -- contains the specified segment
                                                      -- entries in an opaque data structure.
                                                      -- Only complete entries shall be
                                                      -- included in this field.
}

SegmentDataResult ::= SEQUENCE {
        segm-data-event-descr   SegmDataEventDescr
}

--
-- The Segment Data Event Descriptor defines which entries of the Segment Data are communicated in the
-- Event message.
--
SegmDataEventDescr ::= SEQUENCE {
        segm-instance           InstNumber,     -- instance number of segment being transferred
        segm-evt-entry-index    INT-U32,        -- array index of the first entry in this event
        segm-evt-entry-count    INT-U32,        -- count of entries in this event
        segm-evt-status         SegmEvtStatus
}

-- All unassigned " SegmEvtStatus " bit values are reserved for future expansion and shall be set to zero.
SegmEvtStatus ::= BITS-16 {
        sevtsta-first-entry(0),             -- this event contains the first segment entry
        sevtsta-last-entry(1),              -- this event contains the last segment entry (both first
                                            -- and last bits can be set if all entries fit in one event)
        sevtsta-agent-abort(4),             -- transfer aborted by agent (manager shall reply
                                            -- with the same status)
        sevtsta-manager-confirm(8),         -- set in reply if segment was received correctly (if
                                            -- not set in reply, agent shall repeat the last event)
        sevtsta-manager-abort(12)           -- sent in reply by manager (agent shall stop sending
                                            -- messages)
}

SegmentStatistics ::= SEQUENCE OF SegmentStatisticEntry

SegmentStatisticEntry ::= SEQUENCE {
        segm-stat-type    SegmStatType,
        segm-stat-entry   OCTET STRING-- this attribute contains one segment entry in the
                                      -- format defined by the PmSegmentEntryMap
}

-- All unassigned " SegmStatType " values are reserved for future expansion and shall not be used.
-- Values from 0xF000 to 0xFFFF are reserved for manufacturer-specific extensions.
SegmStatType ::= INT-U16 {
        segm-stat-type-undefined (0),
        segm-stat-type-minimum(1),
        segm-stat-type-maximum(2),
        segm-stat-type-average(3)
}
```

Annex B

(informative)

Scale and range specification example

B.1 General

The algorithm for defining the scale and range for an RT-SA is defined in 6.3.5.3, but is repeated here for reference:

$$Y = M \times X + B$$

where

Y = the converted absolute value
M = (upper-absolute-value – lower-absolute-value) / (upper-scaled-value – lower-scaled-value)
B = upper-absolute-value – (M × upper-scaled-value)
X = the scaled value

Note that the term *absolute-value* does not refer to the mathematical absolute value in which all values are positive, but rather to the actual, measured value.

The formula allows measured values with offset range and limited resolution to be replaced by an integer scalar value that can reduce the amount of data that must be communicated between an agent and a manager. The ScaleRangeSpec8, 16 and 32 structures, defined in A.3.4, convey both the upper and lower absolute values and the upper and lower scaled values and allow the manager to determine the parameters for the formula to convert the scaled values into their respective absolute values and to confirm the received values fall within the expected range.

Within an agent, the scaled value that results from the actual measured value may be found from the following:

$$X = (R - B) / M$$

where

R = actual measured value

A suitable value for M would provide scaled values to convey appropriate resolution for the absolute measured values. In practice, the parameters M and B might be set by A/D resolution and other hardware factors.

B.2 Thermometer example

The following example illustrates the algorithm. Readings from a thermometer capable of producing Centigrade readings from –45 °C to 50 °C with a resolution of 0.5°C are to be transmitted as unsigned samples using the ScaleRangeSpec8.

The following values are used for the ScaleRangeSpec8 structure:

Lower-absolute-value = –45.0
Upper-absolute-value = 50.0
Lower-scaled-value = 0
Upper-scaled-value = 190

Giving

M= (50.0 – (–45.0))/(190 – 0) = 0.5
B = 50.0 – (0.5 × 190) = –45.0

Some representative values are given in Table B.1 and Figure B.1 plots the scaled and converted values.

Table B.1—Conversion map

Scaled (x)	Converted (y)
0	–45.0
50	–20.0
100	5.0
150	30.0
190	50.0

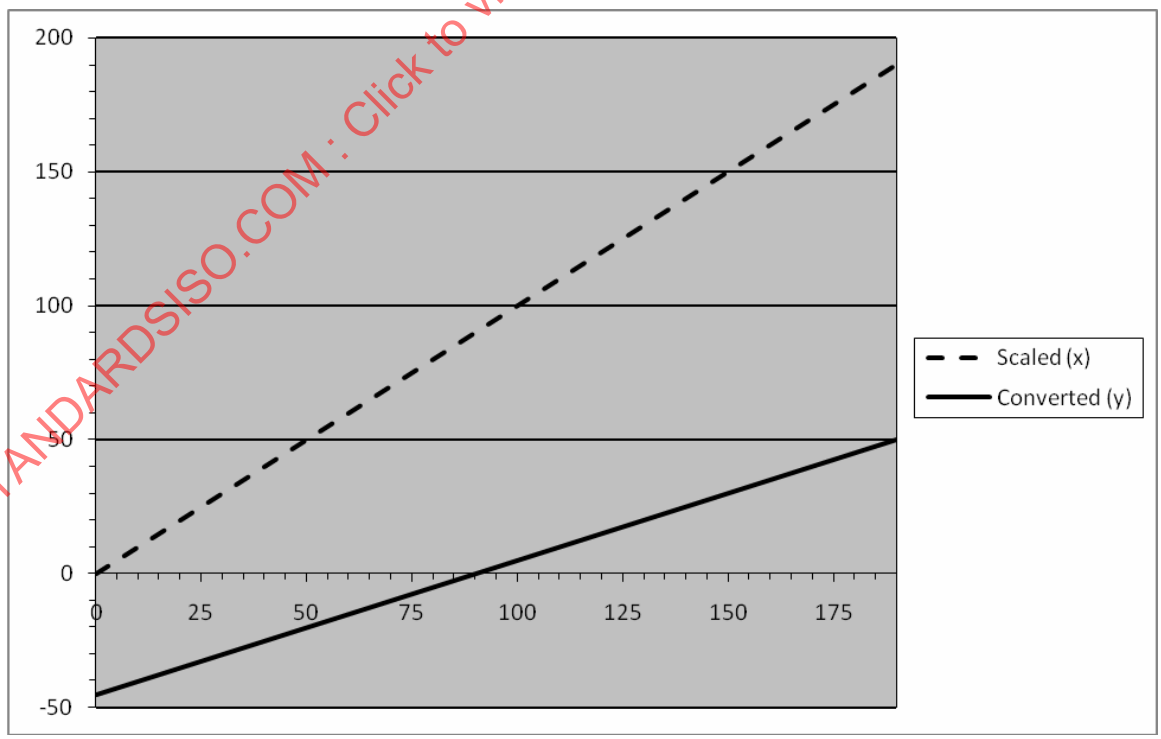


Figure B.1—Graphical view of conversion

Annex C

(informative)

The PM-store concept

C.1 General

The PM-store concept provides a method for representing, accessing, and transferring large amounts of metric data that are stored in the agent. The information is organized in a hierarchical object model with capability to allow data to be stored with a structure appropriate to the nature of the data.

At a top level, the PM-store object is the primary access point for all information about the stored metric data. An agent supporting persistently stored metric data may instantiate one or more PM-store objects. The PM-store object is part of the device configuration and is directly accessed with the object access services defined in this standard.

Each PM-store may contain 0, 1, or more PM-segments that are the actual data container objects. The number of PM-segments may change as a result of the operation of the agent. In other words, the agent may create new PM-segments based on time intervals, size of the stored data, or even manual controls of the user.

The PM-store concept provides an information model with a two-level hierarchy with multiple PM-segment objects within multiple PM-store objects.

Typical-use cases for using multiple PM-stores include the following:

— If the agent stores data with different characteristics (e.g., aperiodic measurements versus periodic measurements), separate PM-store objects are used to define optimized data types for the stored data and thus conserve memory for the stored data.

Typical-use cases for using multiple PM-segments include the following:

— If the agent needs to structure the stored data in a more hierarchical form, it can use multiple instances of PM-store objects with multiple instances of PM-segment objects to model this hierarchy (e.g., use the PM-store to represent a training session, and then use the PM-segment to model individual exercises within this training session).

For the actual data storage, the attribute value map concept as used for the metric attributes is used here. A special mapping attribute allows defining the structure of the binary stored data and avoiding any overhead for identification, length fields, etc., in the actual stored and transmitted binary data. This assumes that stored data are essentially a large array of equally formatted data.

The transfer of the stored data is triggered by the manager after inspecting the information in the PM-store objects. The manager can select the data episodes to transfer. The actual transfer is then done by the agent using acknowledged event report messages. The agent is expected to fill the SegmentDataEvent data structure to the available maximum size.

C.2 Persistent metric store object hierarchy

C.2.1 General

A persistent metric store consists of the following four key parts:

— **PM-store** – This object is at the top level, and it contains attributes about the storage object as well as zero or more PM-segments.

— **PM-segment** – This object contains attributes that describe the segment as well as zero or more entries.

— **Entry** – Each entry holds an optional entry header and one or more elements.

— **Element** – Each element holds data from one or more metric measurements.

Figure C.1 shows an example layout of these four parts, which are further described in the remainder of this annex.

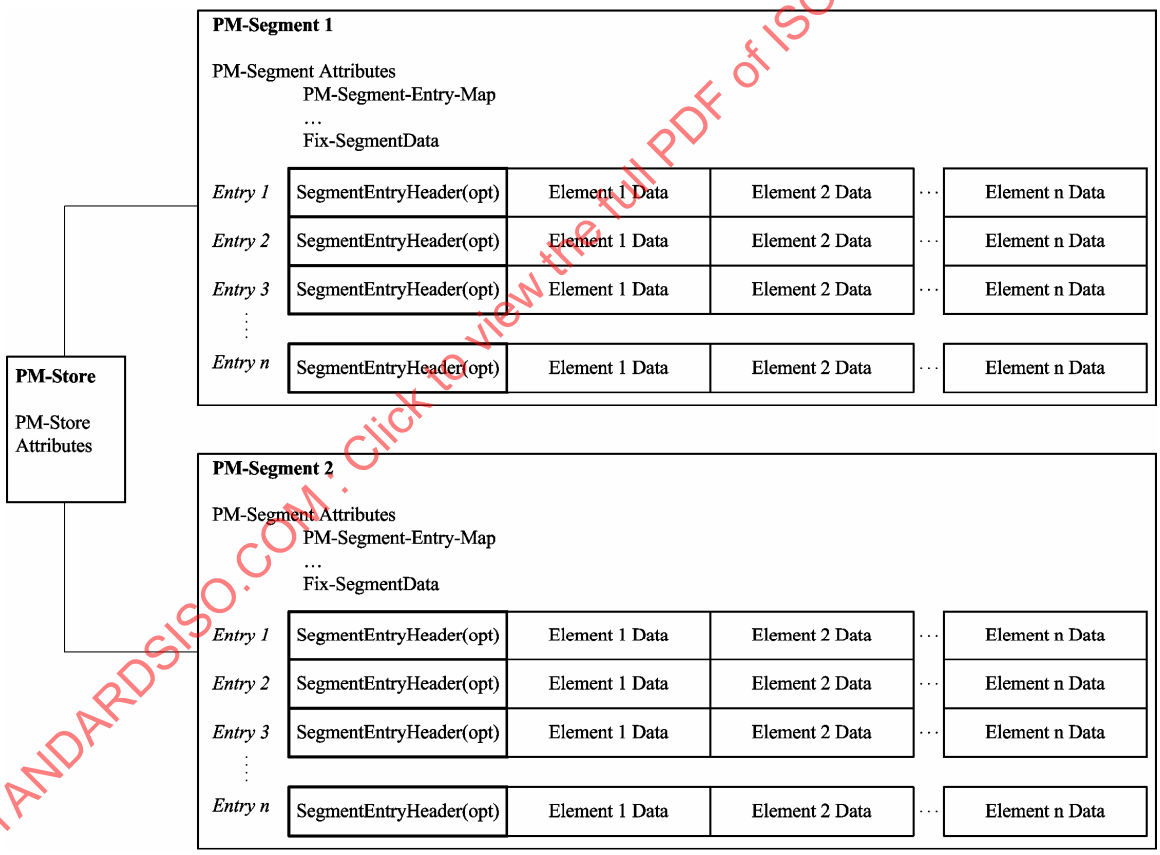


Figure C.1—PM-store with 2 PM-segments, fixed-segment-data within the segments

C.2.2 PM-store object

Support for the PM-store object is optional. Only agents that wish to store persistent metrics need to provide support for the PM-store object, attributes, methods, and associated events. A manager becomes aware of all supported PM-store objects as part of the agent configuration. Attributes of the PM-store describe common characteristics of the stored data (e.g., if values are stored periodically or episodically).

An agent may provide more than one PM-store. Multiple stores are used to represent data with different formats or with different characteristics or to group data into different logical buckets. The PM-store object is also the access point for all methods related to the stored metrics (specifically, the manager retrieving the stored data).

The agent controls the number of PM-segments that exist in a PM-store. A PM-store may have zero elements when there are no data present. When data are present, the PM-store has one or more PM-segment objects. As the number of PM-segments is dynamic, the PM-segment objects are not part of the agent configuration. Instead, the PM-store object contains the information about available PM-segments in the form of PM-store attributes that are queried using the GET service.

C.2.3 PM-segment object

The basic format of the PM-segment segment data is shown in Figure C.2.

Entry 1	SegmentEntryHeader(opt)	Element 1 Data	Element 2 Data	···	Element n Data
Entry 2	SegmentEntryHeader(opt)	Element 1 Data	Element 2 Data	···	Element n Data
Entry 3	SegmentEntryHeader(opt)	Element 1 Data	Element 2 Data	···	Element n Data
⋮					
Entry k	SegmentEntryHeader(opt)	Element 1 Data	Element 2 Data	···	Element n Data

Figure C.2—PM-segment data format

The segment contains *k* entries. The format of an entry is defined by the PM-Segment-Entry-Map attribute of the PM-segment. An entry represents the stored data at one particular point in time. Each entry is preceded by an optional header (e.g., containing a time stamp) common to all elements of the entry. The entry then contains *n* elements; the format of each element is defined by an attribute value map. An entry typically contains a measurement (e.g., numeric and enumeration). The resulting data structure does not contain any attribute ID or length fields and is, therefore, extremely compact.

The PM-segment typically represents one storage episode. This episode has a time context (i.e., data stored in this segment is from 12:00 – 15:00), some related attributes, and a storage array that contains the actual (measured) data for that episode contained in the Fixed-Segment-Data attribute (see Figure C.3).

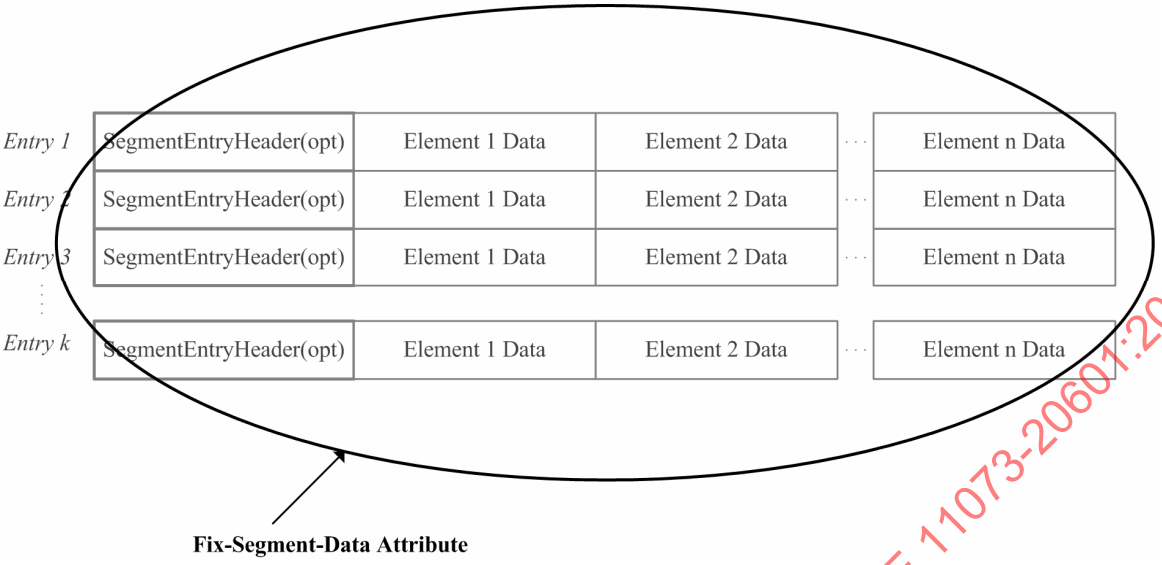


Figure C.3—Fixed-segment-data attribute containing actual stored data

A PM-store may contain zero or more PM-segments (i.e., zero if data are not yet stored; one or more depending on the stored episodes and the capabilities of the agent).

For example, a running watch's PM-segment could contain the stored data about one training cycle (e.g., a 5-mi run starting at 12:00). The device is able to store multiple segments (i.e., multiple such training cycles).

C.2.4 PM-segment entry (within the fixed-segment-data)

The Fixed-Segment-Data attribute contains both entries and elements. The entry items are depicted as rows in Figure C.4. All entries within a segment have the same data structure that is defined by the PM-Segment-Entry-Map. This is very comparable to the Attribute-Value-Map that is defined for the metric objects. However, it allows grouping multiple measurements in an entry item.

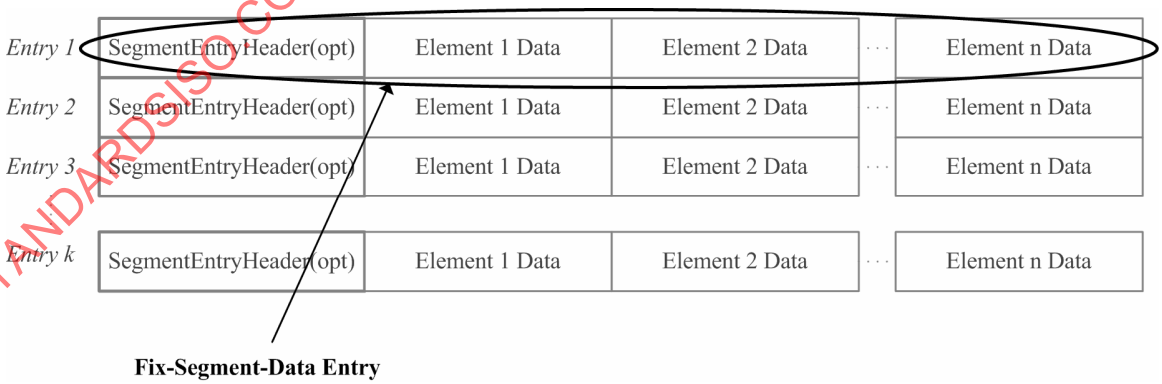


Figure C.4—Entry (array element in fixed-segment-data)

The PM-Segment-Entry-Map attribute defines the list of measurements stored in one entry. For each measurement, the list of attributes that are stored is also defined. Additionally, a common header (e.g., to include a common time stamp) that applies to the complete entry is optionally defined.

Using the running watch example from above, assume the agent stores the heart rate, the current running speed, and a SpO2 value once every second. The only attribute stored from these measurements is the numerical value (which is defined by the PM-Segment-Entry-Map). In this case, no entry header is required because the measurements are periodic and a time stamp is not needed. For periodic measurements, the time of a particular stored measurement is calculated from the start and end time, the sample period, and the index of the entry. Therefore, a separate time stamp for each measurement is not needed in this case, and a header with time stamp information is not required.

Thus, each entry row has the following three elements:

HR	Speed	SpO2
120	10	98

C.2.5 PM-segment entry element

An element contains the binary representation of the defined attributes of one metric object (see Figure C.5). The SegmEntryElem (see A.11.8) within the PM-Segment-Entry-Map defines each entry element.

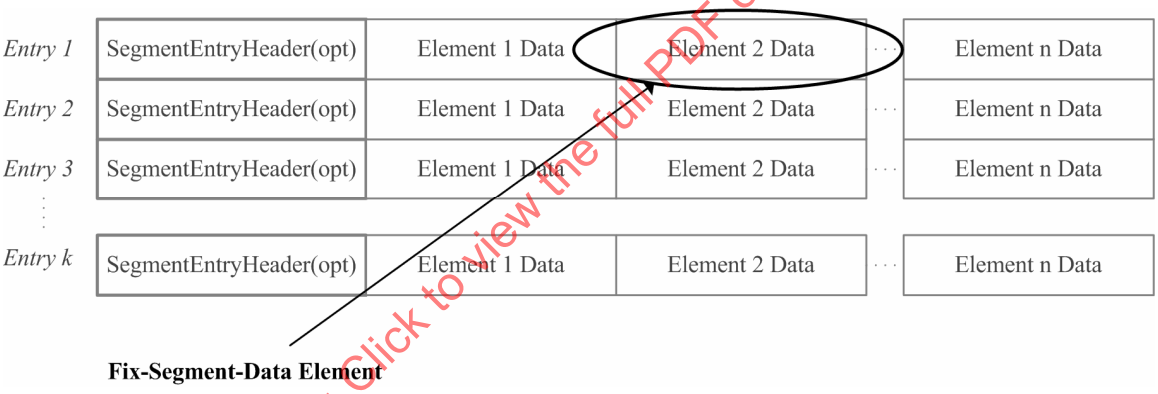


Figure C.5—Element: the set of attributes for one measurement

In the running watch example, there are three metrics modeled in the entry. For each metric, only one single attribute is defined, the numeric observed value. Therefore, the heart rate, the speed, and the SpO2 values are each an element within an entry.

However, the PM-Segment-Entry-Map may contain attributes beyond just the observed value. For instance, it is possible to include attributes such as validity, time stamps, unit codes, and so on.

Annex D

(informative)

Transport profile types

D.1 General

This standard utilizes the concept of a "type" to group and differentiate the services offered by available transport technologies that have been profiled for use by the ISO/IEEE 11073 family of standards. Specifically, the ISO/IEEE 11073 family of standards recognizes the following transport profile types:

— Type 1: Transport profiles that contain **both** reliable and best-effort transport services. There will be one or more virtual channels of reliable transport services and zero or more virtual channels of best-effort transport services.

— Type 2: Transport profiles that contain **only** a unidirectional transport service.

— Type 3: Transport profiles that contain **only** a best-effort transport service. There will be 1 or more virtual channels of best-effort transport services.

The reason the transport profile types are significant is that the different transport services offered by the transport profile types have an effect on the implementation of some upper layer functionality. In particular, they have an effect on the implementation of this standard's confirmed service mechanism.

For the confirmed service mechanism, the definition of *confirmed* is as follows:

— For the data plane (EVENT REPORT services): Allows the agent to know when the manager has "accepted responsibility" for a piece of data so that the agent can delete that datum.

— For the control plane (ACTION, GET, and SET services): Allows the manager to know when the agent has "completed" the requested transaction.

D.2 Type 1

The Type 1 transport profile provides both reliable and best-effort transport services. Considering the definition and goals of the confirmed service mechanism, confirmed messages are certainly sensitive to packet loss. Thus the reliable transport service is the appropriate service to use for all confirmed messages.

Additionally, the agent and manager state machines as defined in this standard (see 8.4) are synchronized state machines. Having synchronized state machines implicitly assumes that there is a reliable transport used between the two state machines that guarantees delivery of a message or indicates failure of delivery. Thus, all association procedure related messages are delivered over a reliable transport service. (For ease of reference, the agent and manager state machines described in 8.4 will be referred to as Type 1 state machines to correlate those state machines to the Type 1 transport profile.)

For unconfirmed messages, the application software is free to use, at its discretion, either a reliable or a best-effort transport service (see Table D.1).

Table D.1—Type 1 transport profile usage

Transport service	IEEE 11073-20601 messages	
	Association procedure & Confirmed	Unconfirmed
Best-effort	Not supported	Supported
Reliable	Supported	Supported

For a transport profile to be considered a Type 1 profile, it must support one or more reliable virtual channels and zero or more best-effort virtual channels.

D.3 Type 2

The Type 2 transport profile provides only a unidirectional transport service. Considering the definition and goals of the confirmed service mechanism, confirmed messages are not able to be supported by a unidirectional transport service. (The manager has no way to send confirmation messages back to the agent.)

As a unidirectional service, this service is inherently a best-effort transport service. The manager's transport layer has no way to request a transport-level retransmission if a transport protocol data unit (PDU) is lost. Thus, a reliable transport service is not possible with a unidirectional transport service.

With the lack of a reliable transport service, the Type 1 state machine will not function correctly over a Type 2 transport profile. Thus, there needs to be a Type 2 state machine specifically for the Type 2 environment (see Table D.2).

Table D.2—Type 2 transport profile usage

Transport service	IEEE 11073-20601 messages	
	Association procedure & Confirmed	Unconfirmed
Best-effort	Not supported	Supported
Reliable	Not supported	Not supported

D.4 Type 3

D.4.1 General

The Type 3 transport profile provides only best-effort transport service. This lack of a reliable transport service presents some difficulties for using the Type 1 state machine and the confirmed service mechanism as currently defined. There are different, and not mutually exclusive, solutions to these difficulties.

D.4.2 Type 3a

One method to handle this situation is to add a reliable transport service companion function to the best-effort transport service. If this step were done, the Type 3 transport profile (best-effort-only) would, in essence, become a Type 1 transport profile (reliable and best-effort). In this case, the Type 1 state machine and the confirmed service mechanism would be applicable.

Thus, Type 3a transport profile is just considered a special case of the Type 1 transport profile.

D.4.3 Type 3b

Another method to handle the best-effort-only transport service would be to migrate the functionality of the reliable transport service up into the personal health device protocol. This approach would result in a Type 3 state machine and a Type 3 confirmation service mechanism (see Table D.3).

Table D.3—Type 3b transport profile usage

Transport service	IEEE 11073-20601 messages	
	Association procedure & Confirmed	Unconfirmed
Best-effort	Supported	Supported
Reliable	**Not supported**	**Not supported**

D.4.4 Type 3c

A third method to handle the best-effort-only transport service would be to add a reliability-lite transport service companion function to the best-effort transport service. This approach is similar to the Type 3a strategy. However, in the Type 3c reliability-lite transport service, some the characteristics of the reliable transport service are relaxed. The expectation is that relaxing some of the reliable transport service characteristics into reliability-lite characteristics would result in a smaller and simpler reliability-lite implementation when compared to a full reliable transport service.

The actual characteristics of the reliable transport service that are relaxed will determine if the currently defined Type 1 state machine and confirmed service mechanism would function correctly in a Type 3c transport profile environment.

D.5 Summary

The descriptions of the transport profile types are summarized in Table D.4.

Table D.4—Transport profile types

Transport profile	Description	"2 x 2" view			Assoc. state machine & confirmed	Data transfer modes
Type 1/3a	Reliable & best-effort		Cfm	Uncfm	Type 1	3
		Best-effort	**NO**	ok		
		Reliable	ok	ok		
Type 2	Unidirectional only		~~Cfm~~	Uncfm	New Type 2	1
		Best-effort	**NO**	ok		
		~~Reliable~~	**NO**	**NO**		
Type 3b	Best-effort-only		Cfm	Uncfm	New Type 3	2
		Best-effort	ok	ok		
		~~Reliable~~	**NO**	**NO**		
Type 3c	Reliable-lite & best-effort		Cfm	Uncfm	To be determined (possibly Type 1)	3
		Best-effort	**NO**	ok		
		Reliable-lite	ok	ok		

Annex E

(normative)

State tables

E.1 General

All the states used by the agent and manager state tables are shown in Table E.1.

Table E.1—States

State number	State	Used by agent	Used by manager
1	Disconnected	Y	Y
2	Connected Unassociated	Y	Y
3	Connected Associating	Y	
4	Connected Associated Configuring Sending Config	Y	
5	Connected Associated Configuring Waiting Approval	Y	
6	Connected Associated Configuring Waiting		Y
7	Connected Associated Configuring Checking Config		Y
8	Connected Associated Operating	Y	Y
9	Connected Disassociating	Y	Y

E.2 Agent state table

The agent state machine shall be implemented as described in Table E.2, which uses the following notations:

REQ – A request from the application software interfacing with the state machine
IND – A condition asserted by a lower layer of software through a well-defined application programming interface
Rx – APDU that has arrived on the input data stream
Tx – APDU that is send on the output data stream

Table E.2—Agent state table

Signal ID	Initial state	Event/input stream	Next state	Semantic behaviors/notes	Tx stream (output)/event output
1.1	Disconnected	IND Transport connection	Connected Unassociated	"Shall" indicate to application layer.	None
2.2	Connected Unassociated	IND Transport disconnect	Disconnected	"Should" indicate to application layer.	None

Table E.2—Agent state table

Signal ID	Initial state	Event/input stream	Next state	Semantic behaviors/notes	Tx stream (output)/event output
2.5	Connected Unassociated	REQ Assoc	Connected Associating	Timeout=reset, retry=reset.	Tx aarq
2.6	Connected Unassociated	REQ AssocRel	Connected Unassociated <no state transition>	None.	None
2.7	Connected Unassociated	REQ AssocAbort	Connected Unassociated <no state transition>	Should not happen.	Tx abrt
2.8	Connected Unassociated	Rx aarq(*)	Connected Unassociated	Agent-agent association.	Tx aare(rejected-permanent)
2.12	Connected Unassociated	Rx aare(*)	Connected Unassociated <no state transition>	Should not happen.	Tx abrt
2.16	Connected Unassociated	Rx rlrq	Connected Unassociated <no state transition>	Should not happen.	Tx abrt
2.17	Connected Unassociated	Rx rlre	Connected Unassociated <no state transition>	Should not happen. Ignore.	None
2.18	Connected Unassociated	Rx abrt	Connected Unassociated <no state transition>	None.	None
2.19	Connected Unassociated	Rx prst(*)	Connected Unassociated <no state transition>	Should not happen.	Tx abrt
3.2	Connected Associating	IND Transport disconnect	Disconnected	None.	None
3.3	Connected Associating	IND Timeout and maximum retry not reached	Connected Associating <no state transition>	Timer=reset, retry++.	Tx aarq
3.4	Connected Associating	IND Timeout and maximum retry reached	Connected Unassociated	None.	Tx abrt
3.6	Connected Associating	REQ AssocRel	Connected Unassociated	None.	Tx abrt
3.7	Connected Associating	REQ AssocAbort	Connected Unassociated	None.	Tx abrt
3.8	Connected Associating	Rx aarq(*)	Connected Unassociated	Agent-agent association.	Tx aare(rejected-permanent)
3.13	Connected Associating	Rx aare(accepted)	Connected Associated Operating	This causes a direct transition to operating state.	None
3.14	Connected Associating	Rx aare(accepted-unknown-config)	Connected Associated Configuring Sending Config	The manager has accepted the association but does not have a configuration.	None

Table E.2—Agent state table

Signal ID	Initial state	Event/input stream	Next state	Semantic behaviors/notes	Tx stream (output)/event output
3.15	Connected Associating	Rx aare(rejected-*)	Connected Unassociated	No further attempts to connect.	None
3.16	Connected Associating	Rx rlrq	Connected Unassociated	Should not happen. The agent has received a request to release the association, but it has not yet established an association.	Tx abrt
3.17	Connected Associating	Rx rlre	Connected Unassociated	Should not happen.	Tx abrt
3.18	Connected Associating	Rx abrt	Connected Unassociated	None.	None
3.19	Connected Associating	Rx prst(*)	Connected Unassociated	Should not happen.	Tx abrt
4.2	Connected Associated Configuring Sending Config	IND Transport Disconnect	Disconnected	None.	None
4.4	Connected Associated Configuring Sending Config	IND Timeout	Connected Unassociated	No reply.	Tx abrt
4.6	Connected Associated Configuring Sending Config	REQ AssocRel (*)	Connected Disassociating	Software requests association release. Timeout=reset.	Tx rlrq(*)
4.7	Connected Associated Configuring Sending Config	REQ AssocAbort	Connected Unassociated	Software abort.	Tx abrt
4.8	Connected Associated Configuring Sending Config	Rx aarq(*)	Connected Unassociated	Should not happen.	Tx abrt
4.12	Connected Associated Configuring Sending Config	Rx aare	Connected Unassociated	Should not happen.	Tx abrt
4.16	Connected Associated Configuring Sending Config	Rx rlrq	Connected Unassociated	None.	Tx rlre(normal)
4.17	Connected Associated Configuring Sending Config	Rx rlre	Connected Unassociated	Should not happen.	Tx abrt
4.18	Connected Associated Configuring Sending Config	Rx abrt	Connected Unassociated	None.	None
4.22	Connected Associated Configuring Sending Config	Rx roiv-cmip-get, handle=0	Connected Associated Configuring Sending Config <no state transition>	Manager allowed to probe MDS. See 6.3.2.6.1.	rors-cmip-get.(MDS Attributes)

Table E.2—Agent state table

Signal ID	Initial state	Event/input stream	Next state	Semantic behaviors/notes	Tx stream (output)/event output
4.23	Connected Associated Configuring Sending Config	Rx roiv-* but not (roiv-cmip-get, handle=0)	Connected Associated Configuring Sending Config <no state transition>	Not allowed until operating state is reached.	Tx roer (no-such-object-instance)
4.26	Connected Associated Configuring Sending Config	Rx (rors-*, roer, or rorj)	Connected unassociated	Should not happen.	Tx abrt
4.32	Connected Associated Configuring Sending Config	REQ Send(ConfigReport)	Connected Associated Configuring Waiting Approval	The agent has a configuration that it has not yet tried with the manager.	Tx EventReport(Config Report)
5.2	Connected Associated Configuring Waiting Approval	IND Transport Disconnect	Disconnected	None.	None
5.4	Connected Associated Configuring Waiting Approval	IND Timeout	Connected Unassociated	No reply.	Tx abrt
5.6	Connected Associated Configuring Waiting Approval	REQ AssocRel(*)	Connected Disassociating	Software request association release. Timeout=reset.	Tx rlrq(*)
5.7	Connected Associated Configuring Waiting Approval	REQ AssocAbort	Connected Unassociated	Software abort.	Tx abrt
5.8	Connected Associated Configuring Waiting Approval	Rx aarq(*)	Connected Unassociated	Should not happen.	Tx abrt
5.12	Connected Associated Configuring Waiting Approval	Rx aare(*)	Connected Unassociated	Should not happen.	Tx abrt
5.16	Connected Associated Configuring Waiting Approval	Rx rlrq	Connected Unassociated	None.	Tx rlre(normal)
5.17	Connected Associated Configuring Waiting Approval	Rx rlre	Connected Unassociated	Should not happen.	Tx abrt
5.18	Connected Associated Configuring Waiting Approval	Rx abrt	Connected Unassociated	None.	None
5.22	Connected Associated Configuring Waiting Approval	Rx roiv-cmip-get, handle=0	Connected Associated Configuring Sending Config	Manager allowed to probe MDS. See 6.3.2.6.1.	rors-cmip-get (MDS Attributes)

Table E.2—Agent state table

Signal ID	Initial state	Event/input stream	Next state	Semantic behaviors/notes	Tx stream (output)/event output
5.23	Connected Associated Configuring Waiting Approval	Rx roiv-* but not (roiv-cmip-get, handle=0)	Connected Associated Configuring Sending Config	Not allowed until operating state is reached.	Tx roer (no-such-object-instance)
5.27	Connected Associated Configuring Waiting Approval	Rx rors-cmip-confirmed-event-report (unsupported-config)	Connected Associated Configuring Sending Config	Manager has rejected the configuration.	None
5.29	Connected Associated Configuring Waiting Approval	Rx rors-cmip-confirmed-event-report (accepted-config)	Connected Associated Operating	Manager has accepted configuration.	None
5.30	Connected Associated Configuring Waiting Approval	Rx (rors-*, roer, or rorj), but not Rx: rors-cmip-confirmed-event-report.	Connected Unassociated	Should not happen.	Tx abrt
8.2	Connected Associated Operating	IND Transport Disconnect	Disconnected	None.	None
8.4	Connected Associated Operating	IND Timeout	Connected Unassociated	No reply.	Tx abrt
8.6	Connected Associated Operating	REQ AssocRel	Connected Disassociating	None. Timeout=reset.	Tx rlrq(normal)[8]
8.7	Connected Associated Operating	REQ AssocAbort	Connected Unassociated	None.	Tx abrt
8.8	Connected Associated Operating	Rx aarq(*)	Connected Unassociated	Should not happen.	Tx abrt
8.12	Connected Associated Operating	Rx aare(*)	Connected Unassociated	Should not happen.	Tx abrt
8.16	Connected Associated Operating	Rx rlrq	Connected Unassociated	If the agent has any outstanding invoke-ids, it shall assume that it shall receive no response to its request.	Tx rlre(normal)
8.17	Connected Associated Operating	Rx rlre	Connected Unassociated	Should not happen.	Tx abrt
8.18	Connected Associated Operating	Rx abrt	Connected Unassociated	None.	None
8.21	Connected Associated Operating	Rx roiv-*	Connected Associated Operating <no state transition>	Normal processing of messages. This is the normal operating state.	Tx (rors-*, or roer, or rorj)

[8] An AssocRel should not be sent until all outstanding invoke-ids are retired.

Table E.2—Agent state table

Signal ID	Initial state	Event/input stream	Next state	Semantic behaviors/notes	Tx stream (output)/event output
8.26	Connected Associated Operating	Rx (rors-*, roer, or rorj)	Connected Associated Operating <no state transition>	Normal processing of messages. This is the normal operating state.	None[9]
9.2	Connected Disassociating	IND Transport Disconnect	Disconnected	None.	None
9.4	Connected Disassociating	IND Timeout	Connected Unassociated	No reply.	Tx abrt
9.6	Connected Disassociating	REQ AssocRel	Connected Disassociating	Already disassociating. Ignore.	None
9.7	Connected Disassociating	REQ AssocAbort	Connected Unassociated	Abort the graceful disassociation process.	Tx abrt
9.8	Connected Disassociating	Rx aarq	Connected Unassociated	Should not happen.	Tx abrt
9.12	Connected Disassociating	Rx aare(*)	Connected Unassociated	Should not happen.	Tx abrt
9.16	Connected Disassociating	Rx rlrq	Connected Disassociating <no state transition>	Both sides releasing connection. Respond and wait for own rlre.	Tx rlre(normal)
9.17	Connected Disassociating	Rx rlre	Connected Unassociated	Release process completed, exit to unassociated.	None
9.18	Connected Disassociating	Rx abrt	Connected Unassociated	None.	None
9.21	Connected Disassociating	Rx roiv-*	Connected Disassociating <no state transition>	The manager sent an invoke message as the agent sent an rlrq. The agent has transitioned out of the Operating state and therefore will not provide a response.	None
9.26	Connected Disassociating	Rx (rors-*, roer, or rorj)	Connected Unassociated	Example 1: Application layer has outstanding invoke-ids but has previously issued a ReleaseRequest anyway. Example 2: Unsolicited rors-*.	Tx abrt

E.3 Manager state table

The manager state machine shall be implemented as described in Table E.3, which uses the following notations:

REQ – A request from the application software interfacing with the state machine
IND – A condition asserted by a lower layer of software through a well-defined API
Rx – APDU that has arrived on the input data stream
Tx – APDU that is send on the output data stream

[9] If an rors-* is received with an unknown invoke-id, then the application layer shall cause an Abort message to be sent to the manager by sending an "REQ abrt" to the state machine.

Table E.3—Manager state table

Signal ID	Initial state	Event/input stream	Next state	Semantic behaviors/notes	Tx stream (output)/event generated
1.1	Disconnected	IND Transport connection	Connected Unassociated	”Shall” indicate to application layer.	None
2.2	Connected Unassociated	IND Transport disconnect	Disconnected	”Should” indicate to application layer.	None
2.6	Connected Unassociated	REQ AssocRel	Connected Unassociated <no state transition>	Should not happen. Ignore.	None
2.7	Connected Unassociated	REQ AssocAbort	Connected Unassociated <no state transition>	Should not happen. Ignore.	None
2.9	Connected Unassociated	Rx aarq (acceptable and known configuration)	Connected associated operating	The device and configuration are known to the manager.	Tx aare(accepted)
2.10	Connected Unassociated	Rx aarq (acceptable with unknown configuration)	Connected associated configuring waiting	The manager determines that the connection is acceptable, but does not have valid configuration information for the agent.	Tx aare(accepted-unknown-config)
2.11	Connected Unassociated	Rx aarq (unacceptable configuration)	Connected unassociated	The manager determines that the connection is unacceptable.	Tx aare(reject-*)
2.12	Connected Unassociated	Rx aare(*)	Connected Unassociated <no state transition>	Should not happen.	Tx abrt
2.16	Connected Unassociated	Rx rlrq	Connected Unassociated <no state transition>	Should not happen.	Tx abrt
2.17	Connected Unassociated	Rx rlre	Connected Unassociated <no state transition>	Should not happen. Ignore.	None
2.18	Connected Unassociated	Rx abrt	Connected Unassociated <no state transition>	Should not happen. Ignore.	None
2.19	Connected Unassociated	Rx prst(*)	Connected Unassociated <no state transition>	Should not happen.	Tx abrt
6.2	Connected Associated Configuring Waiting	IND Transport disconnect	Disconnected	None.	None
6.4	Connected Associated Configuring Waiting	IND Timeout	Connected Unassociated	No reply.	Tx abrt
6.6	Connected Associated Configuring Waiting	REQ AssocRel	Connected Disassociating	None. Timeout=reset.	Tx rlrq(normal)

Table E.3—Manager state table

Signal ID	Initial state	Event/input stream	Next state	Semantic behaviors/notes	Tx stream (output)/event generated
6.7	Connected Associated Configuring Waiting	REQ AssocAbort	Connected Unassociated	None.	Tx abrt
6.8	Connected Associated Configuring Waiting	Rx aarq(*)	Connected Unassociated	Should not happen.	Tx abrt
6.12	Connected Associated Configuring Waiting	Rx aare(*)	Connected Unassociated	Should not happen.	Tx abrt
6.16	Connected Associated Configuring Waiting	Rx rlrq	Connected Unassociated	The manager has received a request to release the association.	Tx rlre(normal)
6.17	Connected Associated Configuring Waiting	Rx rlre	Connected Unassociated	Should not happen.	Tx abrt
6.18	Connected Associated Configuring Waiting	Rx abrt	Connected Unassociated	None.	None
6.24	Connected Associated Configuring Waiting	Rx roiv-confirmed-event-report	Connected Associated Configuring Checking Config	Event report containing configuration from agent.	None
6.25	Connected Associated Configuring Waiting	Rx roiv-* but not (roiv-confirmed-event-report)	Connected Associated Configuring Waiting <no state transition>	Not allowed.	Tx roer (no-such-object-instance)
6.26	Connected Associated Configuring Waiting	Rx(rors-*, roer, or rorj)	Connected Associated Configuring Waiting <no state transition>	Manager may have sent a roiv-cmip-get(handle=0). See 6.3.2.6.1.	None[10]
7.2	Connected Associated Configuring Checking Config	IND Transport Disconnect	Disconnected	None.	None
7.4	Connected Associated Configuring Checking Config	IND Timeout	Connected Unassociated	No reply.	Tx abrt
7.6	Connected Associated Configuring Checking Config	REQ AssocRel	Connected Disassociating	None.	Tx rlrq(normal)

[10] If an rors-* is received with an unknown invoke-id, then the application layer shall cause an Abort message to be sent to the agent by sending an "REQ abrt" to the state machine.

Table E.3—Manager state table

Signal ID	Initial state	Event/input stream	Next state	Semantic behaviors/notes	Tx stream (output)/event generated
7.7	Connected Associated Configuring Checking Config	REQ AssocAbort	Connected Unassociated	None.	Tx abrt
7.8	Connected Associated Configuring Checking Config	Rx aarq(*)	Connected Unassociated	Should not happen.	Tx abrt
7.12	Connected Associated Configuring Checking Config	Rx aare(*)	Connected Unassociated	Should not happen.	Tx abrt
7.16	Connected Associated Configuring Checking Config	Rx rlrq	Connected Unassociated	The manager has received a request to release the association.	Tx rlre(normal)
7.17	Connected Associated Configuring Checking Config	Rx rlre	Connected Unassociated	Should not happen.	Tx abrt [11]
7.18	Connected Associated Configuring Checking Config	Rx abrt	Connected Unassociated	None.	None
7.24	Connected Associated Configuring Checking Config	Rx roiv-confirmed-event-report	Connected Associated Configuring Checking Config <no state transition>	The agent is sending measurements before a configuration is agreed to.	If not a config report, then Tx roer (no-such-object-instance) If config report, then Tx abrt
7.25	Connected Associated Configuring Checking Config	Rx roiv-* but not (roiv-confirmed-event-report)	Connected Unassociated	The agent only sends event report messages. This should never happen.	Tx roer(no-such-action)
7.26	Connected Associated Configuring Checking Config	Rx(rors-*, roer, or rorj)	Connected Associated Configuring Checking Config <no state transition>	Manager might have sent a roiv-cmip-get(handle=0). See 6.3.2.6.1.	None
7.31	Connected Associated Configuring Checking Config	REQ agent supplied unsupported configuration	Connected Associated Configuring Waiting	None.	Tx rors-cmip-configuration-event(unsupported-config)
7.32	Connected Associated Configuring Checking Config	REQ agent supplied supported configuration	Connected Associated Operating	None.	Tx rors-cmip-configuration-event(supported-config)
8.2	Connected Associated Operating	IND Transport Disconnect	Disconnected	None.	None

[11] If an rors-* is received with an unknown invoke-id, then the application layer shall cause an Abort message to be sent to the agent by sending an "REQ abrt" to the state machine.

Table E.3—Manager state table

Signal ID	Initial state	Event/input stream	Next state	Semantic behaviors/notes	Tx stream (output)/event generated
8.4	Connected Associated Operating	IND Timeout	Connected Unassociated	No reply.	Tx abrt
8.6	Connected Associated Operating	REQ AssocRel	Connected Disassociating	None.	Tx rlrq(normal) or Tx rlrq(configuration-changed)[12]
8.7	Connected Associated Operating	REQ AssocAbort	Connected Unassociated	None.	Tx abrt
8.8	Connected Associated Operating	Rx aarq(*)	Connected Unassociated	Should not happen.	Tx abrt
8.12	Connected Associated Operating	Rx aare(*)	Connected Unassociated	Should not happen.	Tx abrt
8.16	Connected Associated Operating	Rx rlrq	Connected Unassociated	None.	Tx rlre(normal)
8.17	Connected Associated Operating	Rx rlre	Connected Unassociated	Should not happen.	Tx abrt
8.18	Connected Associated Operating	Rx abrt	Connected Unassociated	None.	None
8.21	Connected Associated Operating	Rx roiv-*	Connected Associated Operating <no state transition>	Normal processing of messages. This is the normal operating state.	Tx (rors-*, or roer, or rorj)
8.26	Connected Associated Operating	Rx (rors-*, roer, or rorj)	Connected Associated Operating <no state transition>	Normal processing of messages. This is the normal operating state.	None[13]
9.2	Connected Disassociating	IND Transport Disconnect	Disconnected	None.	None
9.4	Connected Disassociating	IND Timeout	Connected Unassociated	No reply.	Tx abrt
9.6	Connected Disassociating	REQ AssocRel	Connected Disassociating <no state transition>	Already disassociating. Ignore	None
9.7	Connected Disassociating	REQ AssocAbort	Connected Unassociated	Abort the graceful disassociation process.	Tx abrt
9.8	Connected Disassociating	Rx aarq(*)	Connected Unassociated	Should not happen.	Tx abrt
9.12	Connected Disassociating	Rx aare(*)	Connected Unassociated	Should not happen.	Tx abrt
9.16	Connected Disassociating	Rx rlrq	Connected Disassociating <no state transition>	Both sides releasing connection. Wait for own rlre.	Tx rlre(normal)
9.17	Connected Disassociating	Rx rlre	Connected Unassociated	Release process completed. Exit to unassociated.	None

[12] An AssocRel should not be sent until all outstanding invoke-ids are retired.
[13] If an rors-* is received with an unknown invoke-id, then the application layer shall cause an Abort message to be sent to the agent by sending an "REQ abrt" to the state machine.

Table E.3—Manager state table

Signal ID	Initial state	Event/input stream	Next state	Semantic behaviors/notes	Tx stream (output)/event generated
9.18	Connected Disassociating	Rx abrt	Connected Unassociated	None.	None
9.21	Connected Disassociating	Rx roiv-*	Connected Disassociating <no state transition>	The agent sent an invoke message as the manager sent an rlrq. The manager has transitioned out of the Operating state and therefore will not provide any response.	None
9.26	Connected Disassociating	Rx (rors-*, roer, or rorj)	Connected Unassociated	Example 1: Application layer has outstanding invoke-ids, but has previously issued a ReleaseRequest anyway. Example 2: Unsolicited rors-*	Tx abrt[14]

[14] If an rors-* is received with an unknown invoke-id, then the application layer shall cause an Abort message to be sent to the agent by sending an "REQ abrt" to the state machine.

Annex F

(normative)

Medical device encoding rules (MDER)

F.1 General

This annex is duplicated from ISO/IEEE 11073-20101:2004 [B14], A.1 through A.4. They are replicated here for implementation convenience.

This annex defines specialized MDER, which concerns presentation of sequential binary strings as they are intended to appear on the network relative to organization in computer memory, to representation in abstract syntax, i.e., programming language or abstract syntax, or in diagrams that are used in specifications. This specification is intended to be consistent with respect to any and all ISO/IEEE 11073 lower layer alternatives; thus, implementations in the upper layers may have to provide for transparency based on a specific lower layer profile.

Significant goals for MDER include the ability to optimize formatting and parsing performance as well as minimizing bandwidth utilization. Formatting optimization focuses on the ability of a data communication processor to define pre-defined transmission templates in which only dynamically changing data need to be included in relatively high-frequency messages, particularly waves.

F.2 Supported ASN.1 syntax

ASN.1 is a standard notation that is used for the definition of data types, values, and constraints on values. This notation is used extensively in OSI standards and is used extensively in the ISO/IEEE 11073 family of standards (e.g., in the DIM where all the data definitions are formalized using ASN.1).

To support the requirement for encoding and decoding performance and support of pre-defined transmission templates, the MDER defines methods to transform ASN.1 syntax into a byte stream suitable for communication.

In contrast to other ISO/OSI standards for ASN.1 encoding rules (e.g., BER and PER), MDER is optimized for a subset of the ASN.1 only. MDER does not support the full set of ASN.1 data types, but only a defined and restricted set of ASN.1 constructs.

The ISO/IEEE 11073 family of standards uses this restricted set of ASN.1 for the definition of data types used within the managed objects only; therefore, MDER is suitable and sufficient for the encoding of data structures within these standards.

The restricted set of ASN.1 used for ISO/IEEE 11073 PDU components is a strict subset of legal ASN.1 data types; therefore, other general standard encoding rules (e.g., XER, PER) may be used as well, as negotiated during association.

Table F.1 defines the specialization of ASN.1 suitable for encoding with MDER. All ASN.1 PDU components destined for encoding with MDER are subject to this specialization.

For each ASN.1 data type, this specialization is indicated by I for included with restriction, R for restrictions on use, or E for excluded.

Table F.1—Supported ASN.1 data types

ASN.1 type	Status	Comments
INTEGER	R	Size constraints shall be used for all INTEGER data types to define the value range of the integer. Short names for the supported constraint types are defined as follows: INT-U8 ::= INTEGER(0..255) INT-I8 ::= INTEGER (-128..127) INT-U16 ::= INTEGER (0..65535) INT-I16 ::= INTEGER (-32768..32767) INT-U32 ::=INTEGER (0..4294967295) INT-I32 ::= INTEGER (-2147483648..2147483647) Only the abbreviated, size-constrained INTEGER data types should be used with data type definitions for encoding in MDER.
BIT STRING	R	Size constraints shall be used for all BIT STRING data types to define the value range of the bit string. Short names for the supported constraint types are defined as follows: BITS-8 ::= BIT STRING (SIZE(8)) BITS-16 ::= BIT STRING (SIZE(16)) BITS-32 ::= BIT STRING (SIZE(32)) Only the abbreviated, size-constrained BIT STRING data types should be used with data type definitions for encoding in MDER.
OCTET STRING	I	—
SEQUENCE	R	May not use OPTIONAL, DEFAULT, or automatic tagging.
SEQUENCE OF	I	—
CHOICE	R	Implicit or explicit tagging may be used.
ANY DEFINED BY	I	An ANY DEFINED BY shall identify a component within the data structure (typically a SEQUENCE) that defines this data structure to a decoder/ parser.

F.3 Byte order

Refer to Figure F.1, which shows how various binary strings are mapped between network and memory. Network byte order (NBO) representation is used in diagrams. The following rules are numbered for reference convenience:

1) Representation in diagrams uses the NBO format shown in Figure F.1.

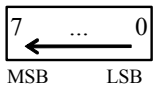
2) No alignment is used in MDER. In other words, additional bytes are not added to byte strings, e.g., to obtain lengths that are divisible by two or four. However, variable-length data items, i.e., strings, should have an even length for performance reasons. For example, because most data elements are 16-bit, they are not misaligned if strings are even length.

3) MDAP communicants are restricted to using the NBO (big-endian) convention.

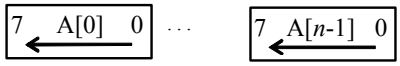
4) The association protocol shall use ISO MDER to provide for universal interoperability during negotiation of MDER conventions. All other PDUs exchanged in the life cycle of device-host communication will be based in MDER, e.g., CMIP* and ROSE* PDUs. The suffixed asterisk (*) indicates that MDER is used as an optimization of the ISO protocol that is based typically in binary encoding rules (BER).

Multibyte structures are mapped between network and computer memory and ordered in computer memory in two basic ways, referred to as *big endian* and *little endian*. Big-endian format is consistent with NBO, but little endian is not. For example, in the last example in Figure F.1, the structure ABCD would be ordered DCBA. In this case, if big endian is the negotiated protocol, then a little-endian machine needs to swap components of these structures both to and from memory, as appropriate. Program language macros and machine-dependent byte-swapping instructions that typically facilitate normalization are implementation issues, but are facilitated by non-normative definitions in this and related standards.

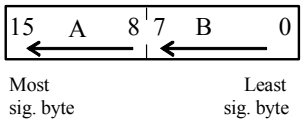
- NBO
  - One byte bit string, i.e., octet
    - Bit sequence: in order of least significant bit (LSB) to most significant bit (MSB), e.g. 0, ... , 7 or 24, ... , 31; bit ordering is representing in diagrams by the following notation, ⟵ , in which the arrow tip represents the last bit transferred:



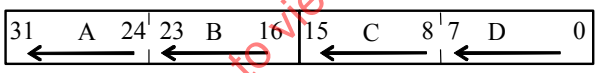
  - Multibyte string
    - Unstructured: an array of octets (i.e., an octet string)
      - Bit sequence: for each byte, as defined for octet
      - Byte sequence: generically numbered from [0] to [*n*-1], e.g., A[0] to A[*n*-1], where *n* = length in octets.



    - Structured: a multibyte ordering of bits, typically in multiples of two (e.g., a short integer is 16 bits, a long integer is 32 bits); floating point numbers in general are multiples of 16 bits, although in this standard, only a 32 bit FLOAT is defined. Two generic examples are given (ABCD refers to byte order):
      - 16 bit structure, e.g., short (integer)
        - Bit sequence: each byte transferred as defined above for octet
        - Byte sequence: transferred in order of most significant byte to least significant byte
        - For signed integers, typically the MSB of the most signficiant byte is the sign (s) bit.



      - 32 bit structure, e.g., long (integer)



      - By convention, multistructure compositions are shown in order of appearance in a serialized string, e.g.,

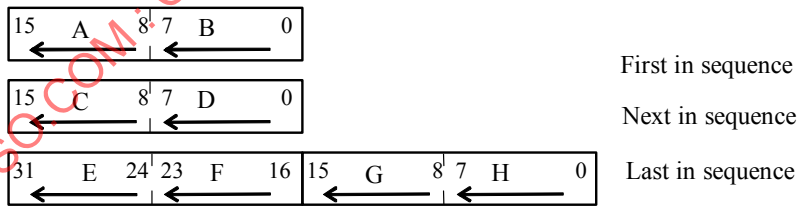


Figure F.1—Binary string [re]presentation conventions—NBO

F.4 Encodings

F.4.1 General

In MDER, there is no tagging for simple types. Tags are used only where a decoder needs to distinguish types (e.g., CHOICE). Length fields are used only for elements with variable length and are restricted to 16 bits (allowing 64k bytes), which should be sufficient for most purposes.

Simple types are defined to have fixed length to optimize overall encoded size.

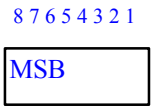
SEQUENCE types have fixed length since OPTIONAL syntax components are not used.

F.4.2 INTEGER

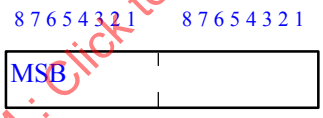
The encoding of an integer value is primitive, and the octets represent the value using a twos-complement binary representation for signed integers and the absolute value for unsigned.

For the size-constrained integer values supported by MDER, Figure F.2 defines octet encodings.[15]

- 8 bit types INT-U8, INT-I8



- 16 bit types INT-U16, INT-I16



- 32 bit types INT-U32, INT-I32

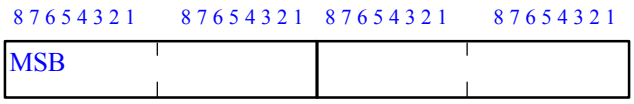


Figure F.2—Integer encodings

The octets contain the twos-complement representation of the encoded integer value.

[15] To promote C programming language standardization for these integer data types, ISO/IEC 9899 definitions are used.

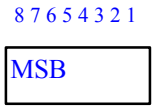
F.4.3 BIT STRING

The encoding of a bit string value is primitive, and the contents octets simply represent the bits set in the bit string. Bit string lengths are constrained to 8-, 16-, or 32-bit lengths.

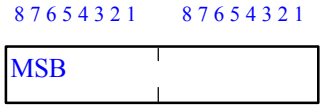
Bit 0 in the encoding is represented by the most significant bit (MSB), bit 1 is represented by the next bit in the octet, etc.

For the size-constrained bit string values supported by MDER, Figure F.3 defines octet encodings.

- 8 bit types BITS-8



- 16 bit types BITS-16



- 32 bit types BITS-32

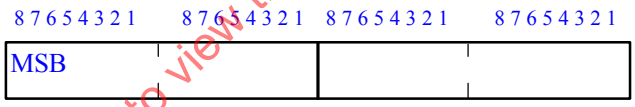


Figure F.3—BIT STRING encodings

Example:

The following definition

state ::= BITS-16 {open(0), locked(1) }

is mapped to a C language type representation as follows:

```
short unsigned int state;
#define locked    0x4000
#define open      0x8000
```

(similar for named bits in BIT STRINGS).

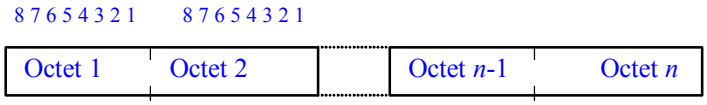
F.4.4 OCTET STRING

The encoding of an octet string value is primitive, and the contents octets simply represent the elements of the string. The encoding of the octets is inherent to the definition of the type of the string.

The octets may contain ASCII printable characters or may contain encapsulated binary data. OCTET STRINGs containing ASCII printable characters shall be even length using a NULL character as padding. Note that strings that are naturally even length may not be NULL terminated.

MDER distinguishes between the fixed-length (size-constrained) OCTET STRING and the variable-length OCTET STRING as shown in Figure F.4:

- Fixed (size-constrained): OCTET STRING (SIZE(*n*))



- Variable-length OCTET STRINGs

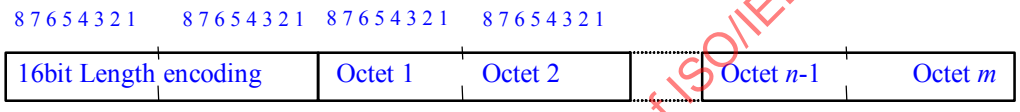


Figure F.4—OCTET STRING encodings

Fixed OCTET STRING types are encoded without a length field and have only the content octets.

Variable-length OCTET STRING types are encoded with a 16-bit length field (unsigned integer, twos-complement), followed by the specific number of content octets.

Example:

The following definitions

```
fixed-sized-label        ::= OCTET STRING (SIZE(12))
variable-label           ::= OCTET STRING
```

can be mapped to C language type representations as follows:

```
typedef unsigned char    fixed_size_label[12];

typedef struct {
        unsigned short   length;
        unsigned char    data[1];    /* this is a placeholder for an appropriately sized array */
} variable_label;
```

F.4.5 SEQUENCE

A SEQUENCE is encoded by encoding each element of the SEQUENCE in the order in which it is defined in the ASN.1 SEQUENCE. No alignment is performed.

Example:

The following definitions

```
IdentType ::= SEQUENCE {
        id                      INT-U16,
        instance                INT-U16
        }
```

are mapped to C language type representations as follows:

```
typedef struct {
        unsigned short  id,
        unsigned short  instance
        } IdentType;
```

and have the MDER encoding in Figure F.5.

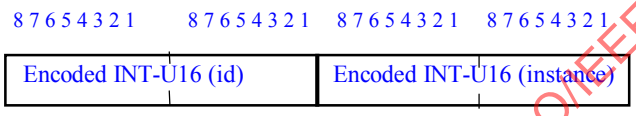


Figure F.5— Sample encoding of a SEQUENCE

F.4.6 SEQUENCE OF

SEQUENCE OF is encoded by a header of a count field to specify the number of encoded elements, *n*, that follow and a length field to specify the total number of octets, *m*, that follow. The length, *m*, shall be equal to *n* × size, where size is the length of each encoded element. Note that length does not include the size of the count and length elements. The header is followed by the encoded elements in order. See Figure F.6.

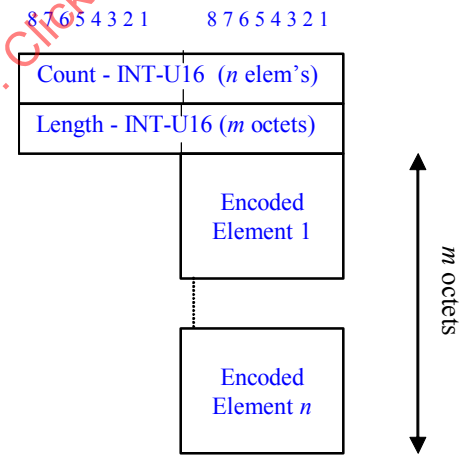


Figure F.6—Encoding of SEQUENCE OF

A count and length field with contents 0 indicates an empty list data structure and is an allowed value.

Example:

The following definition

Array1 ::= SEQUENCE OF Entry

is mapped to a C language type representation as follows:

```
typedef struct {
        unsigned short    count;
        unsigned short    length;
        Entry             data[1];        /* placeholder for sufficient number of entries */
} Array1;
```

F.4.7 CHOICE

CHOICE is encoded by a header of a tag field to specify the encoding of the chosen alternative and a length field to specify the number of octets in the encoding of the chosen alternative that follows. See Figure F.7.

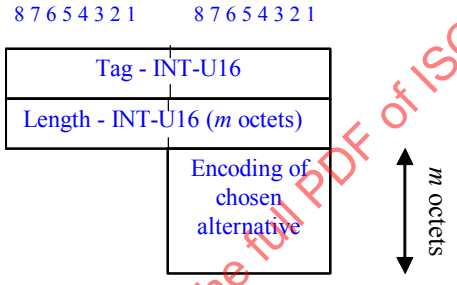


Figure F.7—Encoding of CHOICE

Example:

The following definitions

```
ChoiceType ::= CHOICE {
        one             OneType,
        two             TwoType
        }
```

are mapped to a C language type representation as follows:

```
typedef struct {
        unsigned short    choice_id;
        unsigned short    length;
        union {
                          OneType         one;
                          TwoType         two;
        } data;
} ChoiceType;

#define one_type_chosen                   1
#define two_type_chosen                   2
```

The rules for tag values are defined as follows:

— Tags are implicit or explicit.