

INTERNATIONAL ISO/IEEE STANDARD 11073-20601

Third edition
2022-12

Health informatics — Device interoperability —

Part 20601: Personal health device communication — Application profile — Optimized exchange protocol

Informatique de santé — Interopérabilité des dispositifs —

*Partie 20601: Communication entre dispositifs de santé personnels —
Profil d'application — Protocole d'échange optimisé*



IEEE

Reference number
ISO/IEEE 11073-20601:2022(E)

© IEEE 2019



COPYRIGHT PROTECTED DOCUMENT

© IEEE 2019

All rights reserved. Unless otherwise specified, or required in the context of its implementation, no part of this publication may be reproduced or utilized otherwise in any form or by any means, electronic or mechanical, including photocopying, or posting on the internet or an intranet, without prior written permission. Permission can be requested from IEEE at the address below.

Institute of Electrical and Electronics Engineers, Inc
3 Park Avenue, New York
NY 10016-5997, USA

Email: stds.ipr@ieee.org
Website: www.ieee.org

Published in Switzerland

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.

The procedures used to develop this document and those intended for its further maintenance are described in the ISO/IEC Directives, Part 1. In particular, the different approval criteria needed for the different types of ISO documents should be noted (see www.iso.org/directives).

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.

Attention is drawn to the possibility that some of the elements of this document may be the subject of patent rights. ISO shall not be held responsible for identifying any or all such patent rights. Details of any patent rights identified during the development of the document will be in the Introduction and/or on the ISO list of patent declarations received (see www.iso.org/patents).

Any trade name used in this document is information given for the convenience of users and does not constitute an endorsement.

For an explanation of the voluntary nature of standards, the meaning of ISO specific terms and expressions related to conformity assessment, as well as information about ISO's adherence to the World Trade Organization (WTO) principles in the Technical Barriers to Trade (TBT), see www.iso.org/iso/foreword.html.

ISO/IEEE 11073-20601 was prepared by the *IEEE 11073 Standards Committee of the IEEE Engineering in Medicine and Biology Society* (as IEEE Std 11073-20601-2019) and drafted in accordance with its editorial rules. It was adopted, under the “fast-track procedure” defined in the Partner Standards Development Organization cooperation agreement between ISO and IEEE, by Technical Committee ISO/TC 215, *Health informatics*.

This third edition cancels and replaces the second edition (ISO/IEEE 11073-20601:2016), which has been technically revised. It also incorporates the Technical Corrigendum ISO/IEEE 11073-20601:2016/Cor 1:2016.

A list of all parts in the ISO/IEEE 11073 series can be found on the ISO website.

Any feedback or questions on this document should be directed to the user's national standards body. A complete listing of these bodies can be found at www.iso.org/members.html.

STANDARDSISO.COM : Click to view the full PDF of ISO/IEEE 11073-20601:2022

Health informatics—Personal health device communication

Part 20601: Application profile— Optimized Exchange Protocol

Developed by the

IEEE 11073™ Standards Committee
of the
IEEE Engineering in Medicine and Biology Society

Approved 5 September 2019

IEEE SA Standards Board

STANDARDSISO.COM : Click to view the full PDF of ISO/IEEE 11073-20601:2022

Abstract: Within the context of the ISO/IEEE 11073 family of standards for device communication, 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 is described in this standard. The protocol is optimized to personal health usage requirements and leverages commonly used methods and tools wherever possible.

Keywords: IEEE 11073-20601™, medical device communication, personal health devices

The Institute of Electrical and Electronics Engineers, Inc.
3 Park Avenue, New York, NY 10016-5997, USA

Copyright © 2019 by The Institute of Electrical and Electronics Engineers, Inc.
All rights reserved. Published 20 December 2019. Printed in the United States of America.

IEEE and IEEE 802 are registered trademarks in the U.S. Patent & Trademark Office, owned by The Institute of Electrical and Electronics Engineers, Incorporated.

PDF: ISBN 978-1-5044-6379-9 STD24014
Print: ISBN 978-1-5044-6380-5 STD24014

IEEE prohibits discrimination, harassment, and bullying.

For more information, visit <http://www.ieee.org/web/aboutus/whatis/policies/p9-26.html>.

No part of this publication may be reproduced in any form, in an electronic retrieval system or otherwise, without the prior written permission of the publisher.

Important Notices and Disclaimers Concerning IEEE Standards Documents

IEEE documents are made available for use subject to important notices and legal disclaimers. These notices and disclaimers, or a reference to this page, appear in all standards and may be found under the heading “Important Notices and Disclaimers Concerning IEEE Standards Documents.” They can also be obtained on request from IEEE or viewed at <http://standards.ieee.org/ipr/disclaimers.html>.

Notice and Disclaimer of Liability Concerning the Use of IEEE Standards Documents

IEEE Standards documents (standards, recommended practices, and guides), both full-use and trial-use, are developed within IEEE Societies and the Standards Coordinating Committees of the IEEE Standards Association (“IEEE SA”) Standards Board. IEEE (“the Institute”) develops its standards through a consensus development process, approved by the American National Standards Institute (“ANSI”), which brings together volunteers representing varied viewpoints and interests to achieve the final product. IEEE Standards are documents developed through scientific, academic, and industry-based technical working groups. Volunteers in IEEE working groups are not necessarily members of the Institute and participate without compensation from IEEE. While IEEE administers the process and establishes rules to promote fairness in the consensus development process, IEEE does not independently evaluate, test, or verify the accuracy of any of the information or the soundness of any judgments contained in its standards.

IEEE Standards do not guarantee or ensure safety, security, health, or environmental protection, or ensure against interference with or from other devices or networks. Implementers and users of IEEE Standards documents are responsible for determining and complying with all appropriate safety, security, environmental, health, and interference protection practices and all applicable laws and regulations.

IEEE does not warrant or represent the accuracy or content of the material contained in its standards, and expressly disclaims all warranties (express, implied and statutory) not included in this or any other document relating to the standard, including, but not limited to, the warranties of: merchantability; fitness for a particular purpose; non-infringement; and quality, accuracy, effectiveness, currency, or completeness of material. In addition, IEEE disclaims any and all conditions relating to: results; and workmanlike effort. IEEE standards documents are supplied “AS IS” and “WITH ALL FAULTS.”

Use of an IEEE standard is wholly voluntary. The existence of an IEEE standard does not imply that there are no other ways to produce, test, measure, purchase, market, or provide other goods and services related to the scope of the IEEE standard. Furthermore, the viewpoint expressed at the time a standard is approved and issued is subject to change brought about through developments in the state of the art and comments received from users of the standard.

In publishing and making its standards available, IEEE is not suggesting or rendering professional or other services for, or on behalf of, any person or entity nor is IEEE undertaking to perform any duty owed by any other person or entity to another. Any person utilizing any IEEE Standards document, should rely upon his or her own independent judgment in the exercise of reasonable care in any given circumstances or, as appropriate, seek the advice of a competent professional in determining the appropriateness of a given IEEE standard.

IN NO EVENT SHALL IEEE BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO: PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE PUBLICATION, USE OF, OR RELIANCE UPON ANY STANDARD, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE AND REGARDLESS OF WHETHER SUCH DAMAGE WAS FORESEEABLE.

Translations

The IEEE consensus development process involves the review of documents in English only. In the event that an IEEE standard is translated, only the English version published by IEEE should be considered the approved IEEE standard.

Official statements

A statement, written or oral, that is not processed in accordance with the IEEE SA Standards Board Operations Manual shall not be considered or inferred to be the official position of IEEE or any of its committees and shall not be considered to be, or be relied upon as, a formal position of IEEE. At lectures, symposia, seminars, or educational courses, an individual presenting information on IEEE standards shall make it clear that his or her views should be considered the personal views of that individual rather than the formal position of IEEE.

Comments on standards

Comments for revision of IEEE Standards documents are welcome from any interested party, regardless of membership affiliation with IEEE. However, IEEE does not provide consulting information or advice pertaining to IEEE Standards documents. Suggestions for changes in documents should be in the form of a proposed change of text, together with appropriate supporting comments. Since IEEE standards represent a consensus of concerned interests, it is important that any responses to comments and questions also receive the concurrence of a balance of interests. For this reason, IEEE and the members of its societies and Standards Coordinating Committees are not able to provide an instant response to comments or questions except in those cases where the matter has previously been addressed. For the same reason, IEEE does not respond to interpretation requests. Any person who would like to participate in revisions to an IEEE standard is welcome to join the relevant IEEE working group.

Comments on standards should be submitted to the following address:

Secretary, IEEE SA Standards Board
445 Hoes Lane
Piscataway, NJ 08854 USA

Laws and regulations

Users of IEEE Standards documents should consult all applicable laws and regulations. Compliance with the provisions of any IEEE Standards document does not imply compliance to any applicable regulatory requirements. Implementers of the standard are responsible for observing or referring to the applicable regulatory requirements. IEEE does not, by the publication of its standards, intend to urge action that is not in compliance with applicable laws, and these documents may not be construed as doing so.

Copyrights

IEEE draft and approved standards are copyrighted by IEEE under US and international copyright laws. They are made available by IEEE and are adopted for a wide variety of both public and private uses. These include both use, by reference, in laws and regulations, and use in private self-regulation, standardization, and the promotion of engineering practices and methods. By making these documents available for use and adoption by public authorities and private users, IEEE does not waive any rights in copyright to the documents.

Photocopies

Subject to payment of the appropriate fee, IEEE will grant users a limited, non-exclusive license to photocopy portions of any individual standard for company or organizational internal use or individual, non-commercial use only. To arrange for payment of licensing fees, please contact Copyright Clearance Center, Customer Service, 222 Rosewood Drive, Danvers, MA 01923 USA; +1 978 750 8400. Permission to photocopy portions of any individual standard for educational classroom use can also be obtained through the Copyright Clearance Center.

Updating of IEEE Standards documents

Users of IEEE Standards documents should be aware that these documents may be superseded at any time by the issuance of new editions or may be amended from time to time through the issuance of amendments, corrigenda, or errata. An official IEEE document at any point in time consists of the current edition of the document together with any amendments, corrigenda, or errata then in effect.

Every IEEE standard is subjected to review at least every 10 years. When a document is more than 10 years old and has not undergone a revision process, it is reasonable to conclude that its contents, although still of some value, do not wholly reflect the present state of the art. Users are cautioned to check to determine that they have the latest edition of any IEEE standard.

In order to determine whether a given document is the current edition and whether it has been amended through the issuance of amendments, corrigenda, or errata, visit IEEE Xplore at <http://ieeexplore.ieee.org/> or contact IEEE at the address listed previously. For more information about the IEEE SA or IEEE's standards development process, visit the IEEE SA Website at <http://standards.ieee.org>.

Errata

Errata, if any, for IEEE standards can be accessed via <https://standards.ieee.org/standard/index.html>. Search for standard number and year of approval to access the web page of the published standard. Errata links are located under the Additional Resources Details section. Errata are also available in IEEE Xplore: <https://ieeexplore.ieee.org/browse/standards/collection/ieee/>. Users are encouraged to periodically check for errata.

Patents

Attention is called to the possibility that implementation of this standard may require use of subject matter covered by patent rights. By publication of this standard, no position is taken by the IEEE with respect to the existence or validity of any patent rights in connection therewith. If a patent holder or patent applicant has filed a statement of assurance via an Accepted Letter of Assurance, then the statement is listed on the IEEE SA Website at <https://standards.ieee.org/about/sasb/patcom/patents.html>. Letters of Assurance may indicate whether the Submitter is willing or unwilling to grant licenses under patent rights without compensation or under reasonable rates, with reasonable terms and conditions that are demonstrably free of any unfair discrimination to applicants desiring to obtain such licenses.

Essential Patent Claims may exist for which a Letter of Assurance has not been received. The IEEE is not responsible for identifying Essential Patent Claims for which a license may be required, for conducting inquiries into the legal validity or scope of Patents Claims, or determining whether any licensing terms or conditions provided in connection with submission of a Letter of Assurance, 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 the IEEE Standards Association.

Participants

At the time this standard was completed, the Personal Health Devices Working Group had the following membership:

Daidi Zhong, Co-Chair
Michael J. Kirwan, Co-Chair

Karsten Aalders
Charles R. Abbruscato
Nabil Abujbara
Maher Abuzaid
James Agnew
Haidar Ahmad
Manfred Aigner
Jorge Alberola
Rolf Ambuehl
David Aparisi
Lawrence Arne
Diego B. Arquillo
Serafin Arroyo
Muhammad Asim
Kit August
Doug Baird
David Baker
Anindya Bakshi
Ananth Balasubramanian
Sunlee Bang
M. Jonathan Barkley
Gilberto Barrón
David Bean
John Bell
Rudy Belliardi
Kathryn M. Bennett
Daniel Bernstein
George A. Bertos
Chris Biernacki
Ola Björnsne
Thomas Blackadar
Marc Blanchet
Thomas Bluethner
Douglas P. Bogia
Xavier Boniface
Shannon Boucousis
Julius Broma
Lyle G. Bullock, Jr.
Bernard Burg
Chris Burns
Anthony Butt
Jeremy Byford-Rew
Satya Calloji
Xiaoying Cao
Carole C. Carey
Craig Carlson
Santiago Carot-Nemesio
Randy W. Carroll
Simon Carter
Seungchul Chae
Rahul Chauhan

Wenjuan Chen
James Cheng
Peggy Chien
David Chiu
Jinyong Choi
Chia-Chin Chong
Saeed A. Choudhary
Jinhan Chung
John A. Cogan
John T. Collins
Cory Condek
Todd H. Cooper
David Cornejo
Douglas Coup
Nigel Cox
Hans Crommenacker
Tomio Crosley
Allen Curtis
Eyal Dassau
David Davenport
Russell Davis
Sushil K. Deka
Ciro de la Vega
Pedro de las-Heras-Quiros
Jim Dello Stritto
Matthew d'Entremont
Kent Dicks
Hyoungdo Do
Alistair Donaldson
Xiaolian Duan
Brian Dubreuil
Sourav Dutta
Jakob Ehrensvar
Fredrik Einberg
Michihiro Enokida
Javier Escayola Calvo
Mark Estes
Leonardo Estevez
Hailing Feng
Bosco T. Fernandes
Christoph Fischer
Morten Flintrup
Joseph W. Forler
Russell Foster
Eric Freudenthal
Matthias Frohner
Ken Fuchs
Jing Gao
Qi Gao
Marcus Garbe
John Garguilo

Rick Geimer
Igor Gejdos
Ferenc Gerbovics
Alan Godfrey
Nicolae Goga
Julian Goldman
Raul Gonzalez Gomez
Chris Gough
Channa Gowda
Charles M. Gropper
Amit Gupta
Jeff Guttmacher
Rasmus Haahr
Christian Habermann
Michael Hagerty
Jerry Hahn
Robert Hall
Shu Han
Nathaniel Hamming
Rickey L. Hampton
Sten Hanke
Aki Harma
Jordan Hartmann
Kai Hassing
Wolfgang Heck
Nathaniel Heintzman
Charles Henderson
Jun-Ho Her
Helen B. Hernandez
Takashi Hibino
Timothy L. Hirou
Allen Hobbs
Alex Holland
Arto Holopainen
Kris Holtzclaw
Xinyi Hong
Robert Hoy
Frank Hsu
Anne Huang
Sen-Der Huang
Zhiyong Huang
Ron Huby
David Hughes
Robert D. Hughes
Jiyoung Huh
Hugh Hunter
Hitoshi Ikeda
Yutaka Ikeda
Philip O. Isaacson
Atsushi Ito
Michael Jaffe

Praduman Jain	Jim McCain	Jose A. Santos-Cadenas
Danny Jochelson	László Meleg	Stefan Sauermann
Phaneeth Junga	Alexander Mense	John Sawyer
Akiyoshi Kabe	Behnaz Minaei	Guillaume Schatz
Steve Kahle	Jinsei Miyazaki	Alois Schloegl
Tomio Kamioka	Erik Moll	Paul S. Schluter
James J. Kang	Darr Moore	Lars Schmitt
Kei Kariya	Carsten Mueglitz	Mark G. Schnell
Andy Kaschl	Piotr Murawski	Richard A. Schrenker
Junzo Kashiwara	Soundharya Nagasubramanian	Antonio Scorpiniti
Kohichi Kashiwagi	Jae-Wook Nah	KwangSeok Seo
Ralph Kent	Alex Neefus	Riccardo Serafin
Laurie M. Kermes	Trong-Nghia Nguyen-Dobinsky	Sid Shaw
Ikuo Keshi	Michael E. Nidd	Frank Shen
Ahmad Kheirandish	Jim Niswander	Min Shih
Junhyung Kim	Hongliang Niu	Mazen Shihabi
Minho Kim	Hiroaki Niwamoto	Redmond Shouldice
Min-Joon Kim	Thomas Norgall	Sternly K. Simon
Taekon Kim	Anand Noubade	Marjorie Skubic
Tetsuya Kimura	Yoshiteru Nozoe	Robert Smith
Alfred Kloos	Abraham Ofek	Ivan Son
Jeongmee Koh	Brett Olive	Motoki Sone
Jean-Marc Koller	Begonya Otal	Emily Sopensky
John Koon	Marco Paleari	Rajagopalan Srinivasan
Patty Krantz	Bud Panjwani	Nicholas Steblay
Raymond Krasinski	Carl Pantiskas	Lars Steubesand
Alexander Kraus	Harry P. Pappas	John (Ivo) Stivorice
Ramesh Krishna	Hanna Park	Raymond A. Strickland
Geoffrey Kruse	Jong-Tae Park	Chandrasekaran Subramaniam
Falko Kuester	Myungeun Park	Hermann Suominen
Rafael Lajara	Soojun Park	Lee Surprenant
Pierre Landau	Phillip E. Pash	Ravi Swami
Jaechul Lee	TongBi Pei	Ray Sweidan
JongMuk Lee	Lucian Pestrutu	Yi Tang
Kyong Ho Lee	Soren Petersen	Haruyuyki Tatsumi
Rami Lee	James Petisce	Isabel Tejero
Sungkee Lee	Peter Piction	John W. Thomas
Woojae Lee	Michael Pliskin	Tom Thompson
Yonghee Lee	Varshney Prabodh	Jonas Tirén
Joe Lenart	Jeff Price	Alexandra Todiruta
Kathryn A. Lesh	Harald Prinzhorn	Janet Traub
Qiong Li	Harry Qiu	Jesús Daniel Trigo
Xiangchen Li	Arif Rahman	Gary Tschautscher
Yingsong Li	Tanzilur Rahman	Masato Tsuchid
Zhuofang Li	Steve Ray	Ken Tubman
Patrick Lichter	Phillip Raymond	Yoshihiro Uchida
Jisoon Lim	Terrie Reed	Akib Uddin
Joon-Ho Lim	Tim Reilly	Sunil Unadkat
John Lin	Barry Reinhold	Fabio Urbani
Xiaoming Liu	Brian Reinhold	Philipp Urbauer
Wei-Jung Lo	Melvin I. Reynolds	Laura Vanzago
Charles Lowe	John G. Rhoads	Alpo Värri
Don Ludolph	Jeffrey S. Robbins	Andrei Vasilateanu
Christian Luszick	Chris Roberts	Dalimar Velez
Bob MacWilliams	Moskowitz Robert	Martha Velezis
Srikkanth Madhurbootheswaran	Scott M. Robertson	Rudi Voon
Miriam L. Makhoul	Timothy Robertson	Barry Vornbrock
Romain Marmot	Patricia Order	Isobel Walker
Sandra Martinez	David Rosales	David Wang
Miguel Martínez de Espronceda	Fatemeh Saki	Jerry P. Wang
Cámara	Bill Saltzstein	Yao Wang
Peter Mayhew	Giovanna Sannino	Yi Wang

Steve Warren
Fujio Watanabe
Toru Watsuji
Mike Weng
Yuefeng Weng
Kathleen Wible
Paul Williamson
Jan Wittenber

Jia-Rong Wu
Will Wykeham
Ariton Xhafa
Ricky Yang
Shaoqin Ye
Melanie S. Yeung
Qiang Yin
Done-Sik Yoo

Zhi Yu
Jianchao Zeng
Jason Zhang
Zhiqiang Zhang
Thomas Zhao
Hongyuan Zhong
Miha Zoubek
Szymon Zyskoter

The following members of the individual Standards Association balloting group voted on this standard. Balloters may have voted for approval, disapproval, or abstention.

Juan Carreon
Randy W. Carroll
Lawrence Catchpole
Keith Chow
Malcolm Clarke
Russell Davis
David Fuschi
Randall Groves
Nathaniel Hamming

Werner Hoelzl
Noriyuki Ikeuchi
Atsushi Ito
Stefan Karl
Piotr Karocki
Stuart Kerry
Joerg-Uwe Meyer
Michael Newman

Beth Pumo
Bartien Sayogo
Paul S. Schluter
Carl Singer
Walter Struppler
Lai King (Anna) Tee
Jan Wittenber
Oren Yuen
Daidi Zhong

When the IEEE SA Standards Board approved this standard on 5 September 2019, it had the following membership:

Gary Hoffman, Chair
Ted Burse, Vice Chair
Jean-Philippe Faure, Past Chair
Konstantinos Karachalios, Secretary

Masayuki Ariyoshi
Stephen D. Dukes
J. Travis Griffith
Guido Hiertz
Christel Hunter
Joseph L. Koepfinger*
Thomas Koshy
John D. Kulick

David J. Law
Joseph Levy
Howard Li
Xiaohui Liu
Kevin Lu
Daleep Mohla
Andrew Myles

Annette Reilly
Dorothy Stanley
Sha Wei
Phil Wennblom
Philip Winston
Howard Wolfman
Feng Wu
Jingyi Zhou

*Member Emeritus

Introduction

This introduction is not part of IEEE Std 11073-20601-2019, Health informatics—Personal health device communication—Part 20601: Application profile—Optimized Exchange Protocol.

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:

- IEEE Std 11073-00103™ [B6]¹ provides an overview of the personal health space and defines the underlying use cases and usage models.
- ISO/IEEE 11073-10201:2004 [B17] documents the extensive domain information model (DIM) leveraged by this standard.
- ISO/IEEE 11073-104zz standards define specific device specializations. For example, ISO/IEEE 11073-10404 [B18] defines how interoperable pulse oximeters work.
- ISO/IEEE 11073-20101:2004 [B21] defines the medical device encoding rules (MDER) used in this standard.

¹ The numbers in brackets correspond to the numbers of the bibliography in Annex M.

Contents

1. Overview	13
1.1 Scope	13
1.2 Purpose	13
1.3 Context	13
1.4 Word usage	17
2. Normative references.....	17
3. Definitions, acronyms, and abbreviations	18
3.1 Definitions	18
3.2 Acronyms and abbreviations	19
4. Guiding principles	20
5. Introduction to IEEE 11073 personal health devices.....	21
5.1 General	21
5.2 Domain information model (DIM)	21
5.3 Service model	21
5.4 Communication model.....	22
5.5 Compliance with other standards.....	22
5.6 Security.....	22
6. Personal health device DIM	22
6.1 General	22
6.2 Nomenclature usage	23
6.3 Personal health object class definitions	24
6.4 Information model extensibility rules.....	83
7. Personal health device service model.....	83
7.1 General	83
7.2 Association service	84
7.3 Object access services.....	84
7.4 Specific application of object access EVENT REPORT services for personal health devices.....	85
8. Communication model	93
8.1 General	93
8.2 System context.....	93
8.3 Communications characteristics	94
8.4 State machines.....	97
8.5 Connected procedure	103
8.6 Unassociated procedure	103
8.7 Associating procedure	104
8.8 Configuring procedure.....	111
8.9 Operating procedure	115
8.10 Disassociating procedure	130
8.11 Message encoding.....	131
8.12 Time coordination.....	131
9. Conformance model	135
9.1 Applicability	135
9.2 Conformance specification	135
9.3 Implementation conformance statements (ICSs).....	136
9.4 General conformance.....	136
9.5 Device additions/extensions ICS	140

Annex A (normative) ASN.1 definitions.....	144
A.1 General.....	144
A.2 Common data types	144
A.3 Attribute data types.....	150
A.4 ACTION-method-related data types.....	158
A.5 Message-related data types	160
A.6 Other	160
A.7 Personal health device protocol frame.....	160
A.8 Association protocol definitions	161
A.9 Presentation protocol definitions	164
A.10 Data protocol definitions	164
A.11 Data types for new object attributes and object services	169
Annex B (informative) Scale and range specification example.....	185
B.1 General.....	185
B.2 Thermometer example	185
Annex C (informative) The PM-store concept	187
C.1 General.....	187
C.2 Persistent metric (PM) store object hierarchy	188
Annex D (informative) Transport profile types.....	192
D.1 General.....	192
D.2 Type 1	192
D.3 Type 2	193
D.4 Type 3	193
D.5 Summary.....	194
Annex E (normative) State tables.....	195
E.1 General.....	195
E.2 Events.....	195
E.3 Agent state table.....	197
E.4 Manager state table	206
Annex F (normative) Medical device encoding rules (MDER).....	215
F.1 General	215
F.2 Supported ASN.1 syntax	215
F.3 Byte order.....	216
F.4 Encodings.....	218
F.5 Floating point numbers.....	224
F.6 Floating point data structure—FLOAT-Type.....	224
F.7 Short floating point data structure—SFLOAT-Type.....	225
F.8 Expression of precision of floating point numbers.....	226
Annex G (informative) Encoded data type definitions	227
Annex H (informative) Examples.....	249
H.1 General.....	249
H.2 Weighing scale.....	249
H.3 Pulse oximeter	255
H.4 PM-store and PM-segment transactions	256
Annex I (normative) Nomenclature codes.....	265
Annex J (informative) Derivation and modification history.....	272
J.1 General.....	272
J.2 ASN.1 structures	272
J.3 Medical device encoding rules (MDER).....	272
J.4 Nomenclature codes.....	272

Annex K (informative) The schedule-store concept.....	275
K.1 General.....	275
K.2 Schedule-store object hierarchy.....	276
Annex L (informative) Revision history.....	279
Annex M (informative) Bibliography.....	280

STANDARDSISO.COM : Click to view the full PDF of ISO/IEEE 11073-20601:2022

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 personal health device standard family, this standard defines an optimized exchange protocol and modeling techniques to be used by implementers of personal health devices to create interoperability between device types and vendors. This standard establishes a common framework for 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 standard addresses a need for an openly defined, independent standard for controlling information exchange to and from personal health devices and managers (e.g., cell phones, personal computers, personal health appliances, and set top boxes). Interoperability is key to growing the potential market for these devices and enabling people to be better-informed participants in the management of their health.

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.

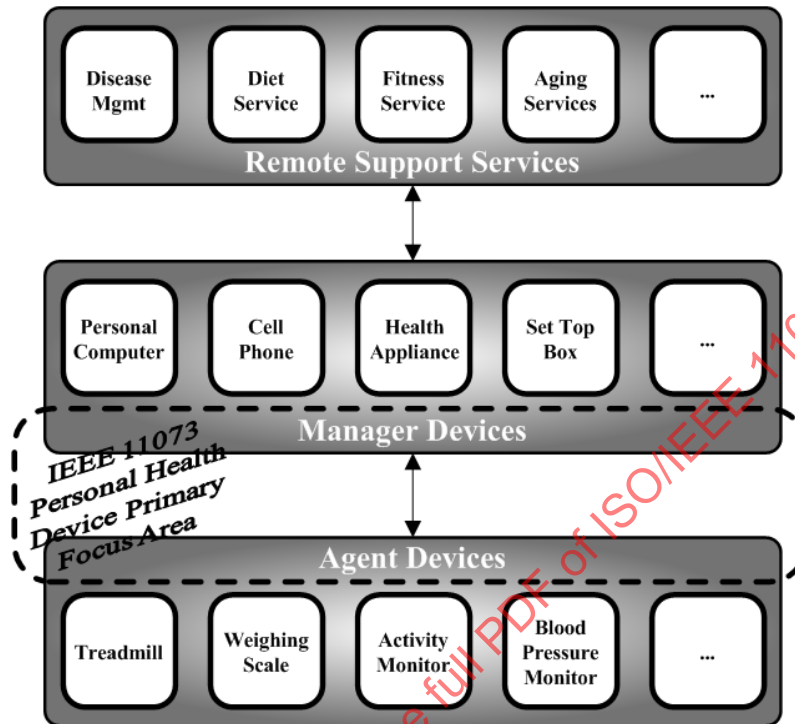


Figure 1—Overall context of work

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 move data reasonably 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 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.

IEEE Std 11073-20601-2019
 Health informatics—Personal health device communication
 Part 20601: Application profile—Optimized Exchange Protocol

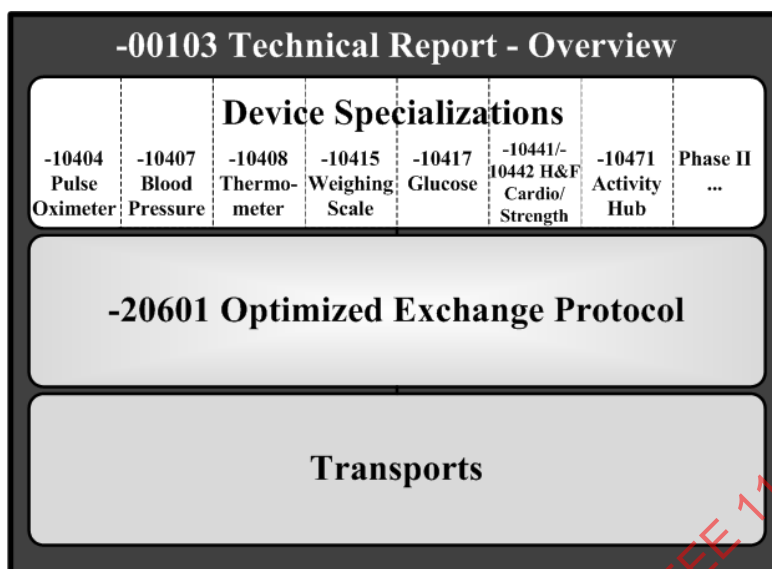


Figure 2—Document map

Above the transport layer is the Optimized Exchange Protocol. 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.

Some device specializations describe broad categories of device types (e.g., the IEEE 11073-10441™ model device types that promote cardiovascular activity such as step counters or exercise cycles). Other device specializations have a narrow focus on a single device type (e.g., IEEE 11073-10408™ model thermometers). Specializations that address one or more device types may also define *profiles*. A profile further constrains the model defined in a specialization to increase interoperability (e.g., the step counter profile utilizes a limited portion of IEEE 11073-10441 modeling).

Two device specializations are defined by this document itself, “hydra” and “generic”. These two specializations are not covered in a separate IEEE 11073-104zz standard. These specializations are defined to support signaling of multi-function or generic agents and managers at the transport level. See 7.4.3.4 for more details.

Note that, in the medical device system (MDS) object defined in this specification, an agent can indicate the full list of supported specializations.

IEEE Std 11073-00103™ [B6]² technical report describes the overall personal health space with further definition of the underlying use cases and usage models.

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:

- 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).

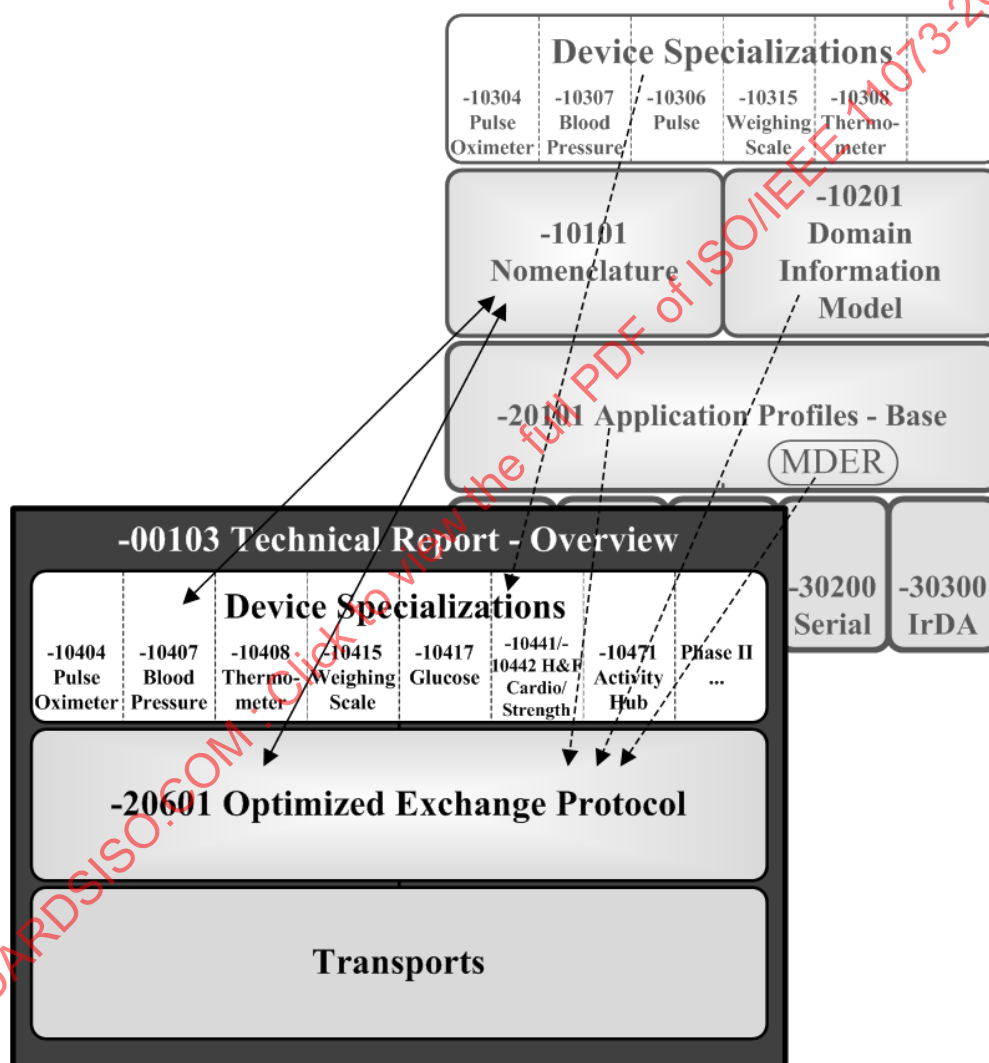


Figure 3—Relationship to other IEEE 11073 documents

This standard imports information from ISO/IEEE 11073-10201:2004 [B17] and ISO/IEEE 11073-20101:2004 [B21] as normative annexes. If there is a discrepancy between these standards, this standard

² The numbers in brackets correspond to the numbers of the bibliography in Annex M.

takes priority. Because of the reuse of constructs from these standards, some of the names appear to be more clinically focused (e.g., MDS instead of personal health device system); however, to maintain consistency, the traditional names have been preserved.

This standard reproduces relevant portions of IEEE Std 11073-1010.³ Between this standard, IEEE Std 11073-10101, and IEEE 11073-104zz, all required nomenclature codes for implementation are documented. New codes may be defined in newer versions/revisions of each of these documents. In the case of a conflict, where one term code has been assigned to two separate semantic concepts with different RefIDs, in general the oldest definition that is in actual use should take precedence. The same policy applies when one RefID has two different code values assigned in different specifications. The resolution of such conflicts will be determined through joint action by the responsible working groups and other stakeholders, and any corrective actions will be published as corrigenda.

1.4 Word usage

The word *shall* indicates mandatory requirements strictly to be followed in order to conform to the standard and from which no deviation is permitted (*shall* equals *is required to*).^{4,5}

The word *should* indicates that among several possibilities one is recommended as particularly suitable, without mentioning or excluding others; or that a certain course of action is preferred but not necessarily required (*should* equals *is recommended that*).

The word *may* is used to indicate a course of action permissible within the limits of the standard (*may* equals *is permitted to*).

The word *can* is used for statements of possibility and capability, whether material, physical, or causal (*can* equals *is able to*).

2. Normative references

The following referenced documents are indispensable for the application of this document (i.e., they must be understood and used, so each referenced document is cited in text and its relationship to this document 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®-2014, IEEE Standard for Local and Metropolitan Area Networks: Overview and Architecture.^{6,7}

IEEE Std 1541™-2002 (Reaff 2008), IEEE Standard for Prefixes for Binary Multiples.

IEEE Std 11073-10101™, Health informatics—Point-of-care medical device communication—Part 10101: Nomenclature.

³ Information about normative references can be found in Clause 2.

⁴ The use of the word *must* is deprecated and cannot be used when stating mandatory requirements; *must* is used only to describe unavoidable situations.

⁵ The use of *will* is deprecated and cannot be used when stating mandatory requirements; *will* is only used in statements of fact.

⁶ IEEE publications are available from The Institute of Electrical and Electronics Engineers (<http://standards.ieee.org>).

⁷ The IEEE standards or products referred to in this clause are trademarks of The Institute of Electrical and Electronics Engineers, Inc.

ISO/IEC 80000-13:2008, Quantities and units—Part 13: Information science and technology.^{8, 9}

ITU-T Rec. X.667 (09/2004), 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.¹⁰

3. Definitions, acronyms, and abbreviations

3.1 Definitions

For the purposes of this document, the following terms and definitions apply. The *IEEE Standards Dictionary Online* should be consulted for terms not defined in this clause.¹¹

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

attribute: Data representing a property of an object. Attributes, together with actions, define an object.

AttributeChangeSet: The set of attribute value changes that represents an atomic update of an object. The medical device system (MDS) or scanner is notified when an AttributeChangeSet is completed. Collections of these AttributeChangeSets are mapped by the MDS or scanner into one of the ObservationScan structures in a scan event report that is sent to the manager. The manager updates its object with the set of attribute value changes contained in the ObservationScan before deriving any semantic behavior.

compute engine: *See: manager.*

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 piece of 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.

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

dynamic attribute: An attribute with a value that may change during an association.

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

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.

metric: An object that models different forms of measurements.

⁸ This standard cancels and replaces sections 3.8 and 3.9 of IEC 60027-2 (2005).

⁹ ISO/IEC publications are available from the International Organization for Standardization (<http://www.iso.ch/>). ISO/IEC publications are also available in the United States from Global Engineering Documents (<http://global.ihs.com/>). Electronic copies are available in the United States from the American National Standards Institute (<http://www.ansi.org/>).

¹⁰ ITU-T publications are available from the International Telecommunications Union (<http://www.itu.int/>).

¹¹ *IEEE Standards Dictionary Online* is available at <http://dictionary.ieee.org>.

object: A unit that represents some functionality or item in a device whose properties are described by attributes. Metric objects represent measurements (such as blood pressure, weight, or temperature), the medical device system (MDS) represents the device, persistent metric store (PM-store) objects represent the permanent storage mechanisms on an agent, and scanners represent a control and reporting mechanism.

observational attribute: An attribute value that may change during the life of an association. The value may be sent in a scan or segment data event report. When a set of observational attribute values are received, these values are combined with the available context information (i.e., all related dynamic and static attribute values) to represent the observation at the observation time. Unlike dynamic and static attribute values, the observational attribute values are combined only once with context information (i.e., the observational attribute values are not reused when any new attribute values are received in the future).

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

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

static attribute: An attribute value that does not change (its value remains fixed) during the life of an association. The value is sent in the configuration event report. The value remains in effect until the system exits the associated state.

NOTE—Do not confuse the use of *static* in this document with *static* as used in the C programming language.¹²

3.2 Acronyms and abbreviations

APDU	application protocol data unit
API	Application Programming Interface
ASCII	American Standard Code for Information Interchange ¹³
ASN.1	Abstract Syntax Notation One
AVA	attribute value assertion
BER	binary encoding rules
DIM	domain information model
DST	daylight savings time
ECG	electrocardiogram or electrocardiograph
EUI-64	extended unique identifier (64 bits)
FIFO	first-in, first-out
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

¹² Notes in text, tables, and figures of a standard are given for information only and do not contain requirements needed to implement the standard.

¹³ Note that throughout this standard the term *ASCII* is used to mean the character set as defined in ISO/IEC 646:1991 [B15].

PER	packed encoding rules
PM	persistent metric
POC	<i>personal health device domain information model</i> object and class
RC _{assoc}	retry count: association procedure
RTC	real-time clock
RT-SA	real-time sample array
SCADA	supervisory control and data acquisition
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 _{cer-sched}	timeout: confirmed event report service for the Schedule-store 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
TO _{sp-sched}	timeout: special segment transfer timeout for the Schedule-store segment object
USB	universal serial bus
UTC	coordinated universal time
UUID	universally unique identifier
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:

- a) Personal health agents typically have very limited computing capabilities.
- b) Personal health agents typically have a fixed configuration, and they are used with a single manager device.
- c) Personal health agents are frequently battery powered, mobile devices, using a wireless communication link. Therefore, energy efficiency of the protocol is an important aspect.
- d) 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.
- e) Personal health managers tend to have more processing power, memory, and storage space so the protocol intentionally places more load on the managers.
- f) 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. 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.

5.2 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.

5.5 Compliance with other standards

Devices that comply with this standard may also be required to comply with other domain- and device-specific standards that supersede the requirements of this standard with respect to issues including safety, reliability, and risk management. A user of this standard is expected to be familiar with all other such standards that apply and to comply with any higher specifications thus imposed. Typically, medical devices will comply with the IEC 60601-1 (2005) +A1 (2012) [B2] base standards with respect to electrical and mechanical safety and any device-specific standard as might be defined in the IEC 60601-2 [B3] series of standards. Software aspects may apply through standards such as IEC 62304 (2006)/EN 62304 (2006) [B4].

Devices that comply with this standard implement higher layers of network software and utilize lower layers as appropriate to the application. The requirements on performance of such applications and conformance are defined elsewhere and are outside the scope of this standard. Moreover, the use of any medical equipment is subject to risk assessment and risk management appropriate to the application. Relevant examples are ISO 14971:2007 [B13] and IEC 80001-1 (2010) [B5]. The requirements of such risk assessment and risk management and conformance are outside the scope of this standard.

5.6 Security

This standard does not provide any method to ensure security of data exchange. It assumes that data exchange is secured by other means, for example, a secure transport channel.

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 [B17], 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 IEEE Std 11073-10101. 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 IEEE Std 11073-10101.

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

For each nomenclature term, IEEE Std 11073-10101 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_YY (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
0	Unspecified
1	Object-oriented
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	File Exchange Format
10	Electrocardiogram (ECG) Extensions
11	IDCO Extensions
12–127	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
1025–65535	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 persistent metric store (PM-store) objects associated with an MDS object. There are zero or more PM-segments that contain PMs 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. The PHD DM Status object (see 6.3.10) allows generic notification handling for personal health devices. The Schedule-store (see 6.3.11) and Schedule-segment (see 6.3.12) allow a manager to retrieve the schedules from an agent.

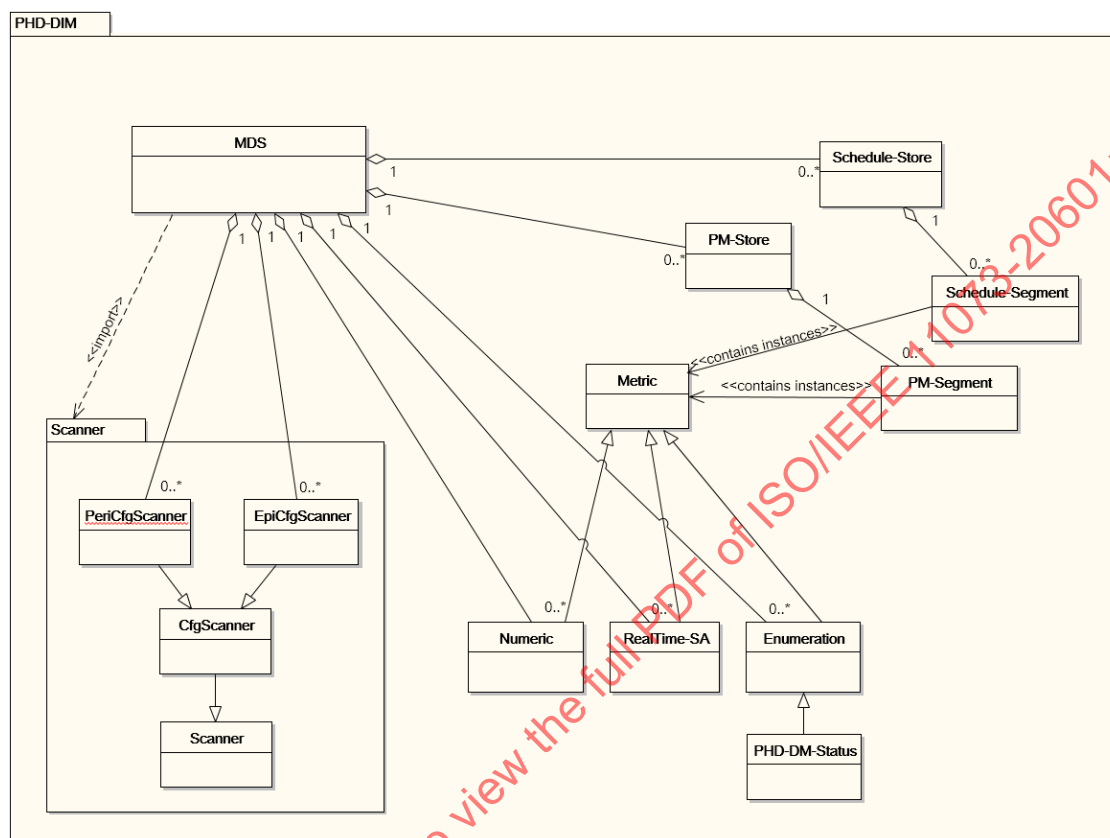


Figure 4—Personal health device—DIM

Subclauses 6.3.2 through 6.3.12 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) [B22]. 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 qualifiers. The qualifiers Conditional, Optional, and Mandatory identify when the attribute is to be implemented in the object. A conditional attribute is

implemented based on the condition(s) stated in the Remark column. Conditional attributes shall be implemented if the remark applies and may be implemented otherwise. Optional attributes may be implemented by an agent. Mandatory attributes shall be implemented by an agent.

An attribute is further qualified as static, dynamic, or observational. Static attributes shall not change value during the life of an association. The value is sent in the configuration event report. The value remains in effect until the system exits the associated state. Dynamic attributes have a value that may change during the life of an association. The attribute value should be sent at configuration time and shall be sent at or before the time when the value would be needed for interpreting a reported observation. The value may be updated later (e.g., in a scan or segment data event report). The attribute value remains in effect until updated by a later scan or segment data event report or the system exits the association. Observational attributes have a value that may change during the life of an association. The value may be sent in a scan or segment data event report. When a set of observational attribute values are received, these values are combined with the available context information (i.e., all related dynamic and static attribute values) to represent the observation at the observation time. Unlike dynamic and static attribute values, the observational attribute values are combined only once with context information (i.e., the observational attribute values are not reused when any new attribute values are received in the future). In the metric object tables, attributes that are labeled as observations may be additionally flagged as a (1) setting, (2) manual entry, or (3) calculation entirely from manual and/or setting entry attributes. In those three cases, the observational attribute is to be considered valid for the time interval between when it is first set until it, or any other attribute in the object in which it is contained, is updated. Without these settings, the observational attribute is only considered valid at the time it is taken. The agent sets these flags in the Metric-Spec-Small attribute. The attribute types are summarized in Table 2.

Table 2—Attribute types

Attribute type	Can be reported by...	Attribute value remains in effect until...
Dynamic attribute may change during the life of an association	<ul style="list-style-type: none"> Should be reported in a configuration event report and shall be sent at or before the time when the value would be needed for interpreting a report observation May be reported in a scan or segment data event report 	<ul style="list-style-type: none"> Updated by a later scan or segment data event report. The system exits the association state.
Observational attribute may change during the life of an association	Shall be reported in a scan or segment data event report	<ul style="list-style-type: none"> These values are combined with the available context information (i.e., all related dynamic and static attribute values) to represent the observation at the observation time.
Static attribute shall not change during the life of an association	<ul style="list-style-type: none"> Shall be reported in the configuration event report 	<ul style="list-style-type: none"> The system exits the association state.

The nomenclature code of the object class (e.g., numeric or 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 3 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 3—MDS attributes

Attribute name	Attribute ID	Attribute type	Remark	Qualifiers
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.	Mandatory Static
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 IEEE Std 11073-10101, nom-part-object partition, and subpartition MD-Gen (Medical Device – Generic). One and only one of System-Type or System-Type-Spec-List shall be present.	Conditional Static
System-Model	MDC_ATTR_ID_MODEL	SystemModel	This attribute defines manufacturer and model number of the agent device.	Mandatory Static
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 802-2014.	Mandatory Static

(Table continues)

Table 3—MDS attributes (*continued*)

Attribute name	Attribute ID	Attribute type	Remark	Qualifiers
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 8.8.	Mandatory Static
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). If the agent uses fixed format value messages to report dynamic data for the object, this attribute shall be set prior to sending such a report. This attribute value may change between sending such event reports and shall be sent in an event report if it changes while associated.	Conditional Dynamic
Production-Specification	MDC_ATTR_ID_PROD_SPECN	ProductionSpec	This attribute defines component revisions, serial numbers, and so on in a manufacturer-specific format.	Optional Static
Udi	MDC_ATTR_ID_UDI	UdiSpec	This attribute defines the UDI(s) under which this product is listed with an authority such as the U.S. Food and Drug Administration (FDA).	Optional Static
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, settable time or BO time is supported.	Conditional Dynamic

(Table continues)

Table 3—MDS attributes (*continued*)

Attribute name	Attribute ID	Attribute type	Remark	Qualifiers
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 message, it shall report its current value of AbsoluteTime in this attribute. If this attribute is used, the Base-Offset-Time attribute shall not be used.	Conditional Observational
Base-Offset-Time	MDC_ATTR_TIME_BO	BaseOffsetTime	This attribute defines the date and time of an agent as a base time with the offset in minutes to local time. For more information on this attribute, see 8.12. If the agent reports BaseOffsetTime in any message, it shall report its current value of base time in this attribute. If this attribute is used, the Date-and-Time attribute shall not be used.	Conditional Observational
Relative-Time	MDC_ATTR_TIME_REL	RelativeTime	If the agent reports RelativeTime in any message, it shall report its current value of RelativeTime in this attribute.	Conditional Observational
HiRes-Relative-Time	MDC_ATTR_TIME_REL_HI_RES	HighResRelativeTime	If the agent reports HighResRelativeTime in any message, it shall report its current value of HighResRelativeTime in this attribute.	Conditional Observational
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.	Conditional Observational
Power-Status	MDC_ATTR_POWER_STATUS	PowerStatus	This attribute reports whether power is being drawn from battery or main power lines and the status of charging.	Optional Dynamic
Battery-Level	MDC_ATTR_VAL_BATTERY_CHARGE	INT-U16	This attribute reports the percentage of battery capacity remaining, which is undefined if value > 100.	Optional Observational

(Table continues)

Table 3—MDS attributes (*continued*)

Attribute name	Attribute ID	Attribute type	Remark	Qualifiers
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.	Optional Observational
Reg-Cert-Data-List	MDC_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 Conformance Statements (see Clause 9) take precedence over this attribute and are the legally binding claims.	Optional Static
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 IEEE Std 11073-10101, nom-part-infrastruct partition, subpartition DEVspec, and reference ISO/IEEE 11073-104zz specializations. If an agent follows a profile within a specialization, it shall report both the specialization and the profile nomenclature values in this list. This list shall also contain the version of the specialization(s)/profile(s). One and only one of System-Type or System-Type-Spec-List shall be present. If the agent is multifunction, this attribute shall be present.	Conditional Static

(Table continues)

Table 3—MDS attributes (*continued*)

Attribute name	Attribute ID	Attribute type	Remark	Qualifiers
Confirm-Timeout	MDC_ATTR_CONFIRM_TIMEOUT	RelativeTime	<p>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.</p> <p>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.</p> <p>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.</p>	Optional Dynamic
Transport-Timeout	MDC_ATTR_TRANSPORT_TIMEOUT	RelativeTime	<p>This timeout attribute defines the minimum time that the manager shall wait for a response message from the agent after sending a message over the underlying transport before timing out and transitioning to the Disconnected state.</p> <p>It is highly recommended for any agent using a transport that may cause long latency to include this attribute in the option-list of PhdAssociationInformation structure so that the manager can prepare for a long delay during Configuring state.</p>	Optional Dynamic
Tick-Resolution	MDC_ATTR_TICK_RESOLUTION	FLOAT-Type	This attribute defines the resolution for the agent’s internal oscillator (i.e., it specifies the number of ticks per second).	Optional Static

Note that the attribute System-Type is independent of the similarly named System-Type field in the PhdAssociationInformation structure of an association request.

The MDS attributes represent a device-level view and are independent of the particular configuration that has been offered or accepted. For instance, the System-Type-Spec-List or Reg-Cert-Data-List provides all capabilities that the device may offer. The current configuration may or may not provide everything listed in the attribute.

The attribute data types are defined in Annex A.

6.3.2.4 MDS object methods

Table 4 defines the methods (actions) available for the MDS object. These methods are invoked using the ACTION service. In Table 4, 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 4—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
Set-Base-Offset-Time	Confirmed	MDC_ACT_SET_BO_TIME	SetBOTimeInvoke	None

— **MDS-Data-Request:**

This method allows the manager system to request measurement data 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 3). If the agent supports Set-Time, it shall respond with a rors-cmip-confirmed-action, but the action-info-args is empty in this response. If the agent does not support Set-Time, it shall respond with a no-such-action error (roer). The Set-Time method can be supported only if the Absolute-Time-Stamp attribute is supported.

— **Set-Base-Offset-Time:**

This method allows the manager system to set a RTC with a coordinated universal time- (UTC-) aligned base time and the offset in minutes to local time. The agent indicates whether the Set-Base-Offset-Time command is valid by using the mds-time-capab-set-clock bit in the Mds-Time-Info attribute (see Table 3). If the agent supports Set-Base-Offset-Time, it shall respond with a rors-cmip-confirmed-action, but the action-info-args is empty in this response. If the agent does not support Set-Base-Offset-Time, it shall respond with a no-such-action error (roer). If the base time seconds field and base time fraction of a second field are set to 0x0 in the arguments of the Set-Base-Offset-Time action [these values are undefined in Network Time Protocol (NTP)], then only the offset to local time shall be set. The Set-Base-Offset-Time method can be supported only if the Base-Offset-Time-Stamp attribute is supported.

An agent may support either absolute time or base offset time but not both.

6.3.2.5 MDS object events

Table 5 defines the potential events sent by the MDS object. A manager shall support all methods defined in Table 5. If a confirmed event report is used by the MDS object, there shall be at most one unacknowledged confirmed event report outstanding from this object at any point in time.

Table 5—MDS object events

Event	Mode	Event-type	Event-info parameter	Event-reply-info
MDS-Configuration-Event	Confirmed	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 sent as an unsolicited message by the agent. Refer to 8.9.3.3.2 for information on the control a manager can assert on the transmission of data from an agent.

— **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 sent as an unsolicited message by the agent (i.e., an agent-initiated measurement data transmission. Refer to 8.9.3.3.2 for information on the control a manager can assert on the transmission of data from an agent.

— **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 shall be invoked by the manager on reaching the Configuring/Sending GetMDS state. The manager shall wait for the GET response before invoking any actions. Waiting for the Get response allows the manager to determine whether the agent needs the time to be set. An agent that has requested its time to be set shall wait for the Set-Time action before entering Operating state.

The GET service can be invoked when the agent is in Configuring/Waiting MDS substate, Configuring/Waiting SetTime substate, and Operating state.

With the exception of the Date-and-Time-Adjustment attribute, if a manager does not have the current value of a needed MDS attribute, then the GET service shall be used. An agent may also send scan event reports providing the manager with updates of the current attribute values, but this is not a mandated agent behavior with the exceptions described for attribute-value-map and date-and-time-adjustment in Table 3.

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. See 8.9.3.2 for a full explanation of the GET operation.

NOTE—Requirements such as the need for an agent to request a manager to set its time or to send temporarily stored data with Date-and-Time-Adjustment in fixed format necessitate that a manager request the MDS object from an agent. Experience has also shown that the demands placed on the agent device when sending the GET request after an AARE accepted-unknown-config (which requires the agent to process potentially large Config Reports and GET responses asynchronously) have often led to timeouts and other issues. The current guidelines with respect to the GET request are to address these issues.

6.3.2.6.2 SET service

There are currently no uses of the MDS SET service defined in this standard; however, ISO/IEEE 11073-104zz device specializations or proprietary definitions may exist.

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

Table 6—Metric attributes

Attribute name	Attribute ID	Attribute type	Remark	Qualifiers
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.	Mandatory Static
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.	Mandatory Static
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™ [B12] defines location nomenclature for specifying the location of a sensor in a home and ISO/IEEE 11073-10404 [B18] defines three supplemental types for fast response, slow response, and spot checking of the pulse rate or blood oxygenation.	Optional Dynamic
Metric-Spec-Small	MDC_ATTR_METRIC_SPEC_SMALL	MetricSpecSmall	This attribute describes the characteristics of the measurements. This attribute is dynamic to allow an agent to alter the bit settings prior to sending the first observation (e.g., to adjust a standard configuration). Once an observation is sent, the agent shall not update this attribute's value.	Mandatory Dynamic

(Table continues)

Table 6—Metric attributes (continued)

Attribute name	Attribute ID	Attribute type	Remark	Qualifiers
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}. This attribute is dynamic to allow an agent to alter the settings prior to sending the first observation (e.g., to adjust a standard configuration). Once an observation is sent, the agent shall not update this attribute's value.	Optional Dynamic
Measurement-Status	MDC_ATTR_MSMT_STAT	MeasurementStatus	This attribute indicates the validity of a particular value or set of samples. If a scanner reports an RTSA and there is a possibility that the scanner may need to report missing observations (see 6.3.9.2), this attribute shall be present. If thresholding is to be used, this attribute is mandatory and reflects whether observed value is at or outside threshold boundaries: Use bit msmt-value-exceed-boundaries(14) to indicate that the measurement is outside threshold boundaries. Use msmt-state-ann-inhibited(15) to indicate that the threshold indication is disabled and should not cause a displayed annunciation. If this bit is set, the bit 14 shall not be set.	Conditional Observational
Metric-Id	MDC_ATTR_ID_PHY_SIO	OID-Type	<p>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.</p> <p>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.</p>	Optional Dynamic

(Table continues)

Table 6—Metric attributes (*continued*)

Attribute name	Attribute ID	Attribute type	Remark	Qualifiers
Metric-Id-List	MDC_ATTR_ID_PHY_SIO_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.	Conditional Dynamic
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.	Optional Dynamic
Attribute-Value-Map	MDC_ATTR_ATTRIBUTE_VAL_MAP	AttrValMap	This attribute defines the attributes that are reported in the fixed format data update messages. If the agent uses fixed format value messages to report dynamic measurement data for the object, this attribute shall be set prior to sending such a report. This attribute value may change in between event reports.	Conditional Dynamic
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 SpO ₂). 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. A metric object may contain one of Source-Handle-Reference or Source-Handle-Reference-List, but not both. If a measurement reports a Source-Handle-Reference or Source-Handle-Reference-List attribute, the measurement from the object(s) that it references shall be sent by the agent to the manager prior to the sending of this measurement.	Optional Dynamic

(Table continues)

Table 6—Metric attributes (*continued*)

Attribute name	Attribute ID	Attribute type	Remark	Qualifiers
Source-Handle-Reference-List	MDC_ATTR_SOURCE_HANDLE_REF_LIST	HANDLEList	This attribute establishes a relation of this object instance to more than one source objects (e.g., body mass index (BMI) references sourcing Height and Weight). 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. A metric object may contain one of Source-Handle-Reference or Source-Handle-Reference-List, but not both. If a measurement reports a Source-Handle-Reference or Source-Handle-Reference-List attribute, the measurement from the object(s) that it references shall be sent by the agent to the manager prior to the sending of this measurement.	Optional Dynamic
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.	Optional Dynamic
Absolute-Time-Stamp	MDC_ATTR_TIME_STAMP_ABS	AbsoluteTime	This attribute defines the date and time of observation at the end of the measurement period 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 one and only one time stamp (Absolute-Time-Stamp, Base-Offset-Time-Stamp, Relative-Time-Stamp, or HiRes-Time-Stamp) with the data. If this attribute is used, the Base-Offset-Time-Stamp attribute shall not be used.	Conditional Observational

(Table continues)

Table 6—Metric attributes (continued)

Attribute name	Attribute ID	Attribute type	Remark	Qualifiers
Base-Offset-Time-Stamp	MDC_ATTR_TIME_S TAMP_BO	BaseOffsetTime	This attribute defines the base date and time of observation at the end of the measurement period and the offset in minutes to local time. 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 one and only one time stamp (Absolute-Time-Stamp, Base-Offset-Time-Stamp, Relative-Time-Stamp, or HiRes-Time-Stamp) with the data. If this attribute is used, the Absolute-Time-Stamp attribute shall not be used.	Conditional Observational
Relative-Time-Stamp	MDC_ATTR_TIME_S TAMP_REL	RelativeTime	This attribute defines the time of observation at the end of the measurement period (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 one and only one time stamp (Absolute-Time-Stamp, Base-Offset-Time-Stamp, Relative-Time-Stamp, or HiRes-Time-Stamp) with the data.	Conditional Observational
HiRes-Time-Stamp	MDC_ATTR_TIME_S TAMP_REL_HI_RES	HighResRelativeTime	This attribute defines the time of observation at the end of the measurement period (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 one and only one time stamp (Absolute-Time-Stamp, Base-Offset-Time-Stamp, Relative-Time-Stamp, or HiRes-Time-Stamp) with the data.	Conditional Observational
Measure-Active-Period	MDC_ATTR_TIME_P D_MSMT_ACTIVE	FLOAT-Type	This attribute defines the time duration of the observation period in seconds. By default, this attribute is dynamic; however, ISO/IEEE 11073-104zz may define it as observational based on the need of its use case.	Optional Dynamic or Observational

(Table continues)

Table 6—Metric attributes (continued)

Attribute name	Attribute ID	Attribute type	Remark	Qualifiers
Context-Key	MDC_ATTR_CONTEXT_KEY	OCTET STRING	Context key for the current relevant device contents and settings. This attribute is optional and may be utilized to differentiate contexts in cases where the System-Identifier is not sufficient such as implementations that have loadable/changeable contents. It could contain a URL, an EUI-64, or some other external ID. This should be defined in specializations using this attribute.	Optional Dynamic
Event-Context-Id	MDC_ATTR_EVENT_CONTEXT	UINT32	An Event Context Id identifies an event context. This attribute is optional and may be used to tie together events from metric objects.	Optional Observational

6.3.3.4 Metric object methods

There are currently no metric object methods defined in this standard; however, ISO/IEEE 11073-104zz device specializations or proprietary definitions may exist.

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

There are currently no uses of the metric SET or GET services defined in this standard; however, ISO/IEEE 11073-104zz device specializations or proprietary definitions may exist.

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.

It also includes attributes to support monitoring of thresholds of the measurement value.

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

Table 7—Numeric attributes

Attribute name	Attribute ID	Attribute type	Remark	Qualifiers
Simple-Nu-Observed-Value	MDC_ATTR_NU_VAL_OBS_SIMP	SimpleNuObsValue	This attribute defines the numerical observed value of the object. One and only one of Simple-Nu-Observed-Value, Basic-Nu-Observed-Value, Compound-Nu-Observed-Value, Compound-Simple-Nu-Observed-Value, or Compound-Basic-Nu-Observed-Value shall be present.	Conditional Observational
Compound-Simple-Nu-Observed-Value	MDC_ATTR_NU_CMPD_VAL_OBS_SIMP	SimpleNuObsValue Cmp	This attribute represents an array of Simple-Nu-Observed-Values. One and only one of Simple-Nu-Observed-Value, Basic-Nu-Observed-Value, Compound-Nu-Observed-Value, Compound-Simple-Nu-Observed-Value, or Compound-Basic-Nu-Observed-Value shall be present.	Conditional Observational
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 and only one of Simple-Nu-Observed-Value, Basic-Nu-Observed-Value, Compound-Nu-Observed-Value, Compound-Simple-Nu-Observed-Value, or Compound-Basic-Nu-Observed-Value shall be present.	Conditional Observational
Compound-Basic-Nu-Observed-Value	MDC_ATTR_NU_CMPD_VAL_OBS_BASIC	BasicNuObsValueC mp	This attribute represents an array of Basic-Nu-Observed-Values. One and only one of Simple-Nu-Observed-Value, Basic-Nu-Observed-Value, Compound-Nu-Observed-Value, Compound-Simple-Nu-Observed-Value, or Compound-Basic-Nu-Observed-Value shall be present.	Conditional Observational

(Table continues)

Table 7—Numeric attributes (continued)

Attribute name	Attribute ID	Attribute type	Remark	Qualifiers
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 and only one of Simple-Nu-Observed-Value, Basic-Nu-Observed-Value, Compound-Nu-Observed-Value, Compound-Simple-Nu-Observed-Value, or Compound-Basic-Nu-Observed-Value shall be present.	Conditional Observational
Accuracy	MDC_ATTR_NU_ACCUR_MSMT	FLOAT-Type	This attribute is an absolute value, and it defines the maximum deviation of actual value from reported observed value (if it can be specified) over the entire range of measurement. It is expressed in the same unit as the observation. When reporting a measurement with certain accuracy, a reported value should have a precision (see F.8) sufficient to express this accuracy.	Optional Static
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). The prefixes of unit shall be generated according to IEEE Std 1541-2002 and ISO/IEC 80000-13:2008. If the observation attribute contains the unit, this attribute does not need to be present.	Conditional Dynamic
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.	Optional Dynamic

(Table continues)

Table 7—Numeric attributes (continued)

Attribute name	Attribute ID	Attribute type	Remark	Qualifiers
Alert-Op-State	MDC_ATTR_AL_OP_STAT	CurLimAlStat	Reflects the current state of the threshold annunciation masking bits. If thresholding is to be used, this attribute is mandatory.	Conditional Dynamic
Current-Limits	MDC_ATTR_LIMIT_CURR	CurLimAlVal	Current threshold values. If thresholding is to be used, this attribute is mandatory.	Conditional Dynamic
Alert-Op-Text-String	MDC_ATTR_AL_OP_TEXT_STRING	AlertOpTextString	Individual text for upper and lower threshold.	Conditional Optional Dynamic
Measurement-Confidence-95	MDC_ATTR_MSMT_CONFIDENCE_95	MeasurementConfidence95	This attribute defines the lower and upper bounds for a range within which the manufacture is 95% confident that the actual value resides. The unit for the lower bound and upper bound is the same as the measurement.	Optional Observational
Threshold-Notification-Text-String	MDC_ATTR_THRES_NOTIFICATION_TEXT_STRING	OCTET STRING	Text related to the current threshold notification.	Optional Dynamic

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.”

For a non-compound numeric observation, thresholding may be used. This implies that threshold values are present in the Current-Limits attribute and that the Alert-Op-State is present and reflects the annunciation. In addition, an Alert-Op-Text-String attribute may be present as well.

6.3.4.4 Numeric object methods

There are currently no numeric object methods defined in this standard; however, ISO/IEEE 11073-104zz device specializations or proprietary definitions may exist.

6.3.4.5 Numeric object events

There are currently no numeric object events defined in this standard.

6.3.4.6 Other numeric services

There are currently no uses of the numeric SET or GET services defined in this standard; however, ISO/IEEE 11073-104zz device specializations or proprietary definitions may exist.

6.3.5 RT-SA class

6.3.5.1 General

An instance of the RT-SA class represents a waveform 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 8 defines the set of RT-SA attributes that are supported for personal health device communication.

Table 8—RT-SA attributes

Attribute name	Attribute ID	Attribute type	Remark	Qualifiers
Sample-Period	MDC_ATTR_TIME_PD_SAMP	RelativeTime	This attribute defines time intervals between successive samples given in 1/8 of a millisecond. Thus, 8000 = 1 s.	Mandatory Static
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.	Mandatory Observational
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). The prefixes of unit shall be generated according to IEEE Std 1541-2002 and ISO/IEC 80000-13:2008.	Mandatory Dynamic

(Table continues)

Table 8—RT-SA attributes (*continued*)

Attribute name	Attribute ID	Attribute type	Remark	Qualifiers
Unit-LabelString	MDC_ATTR_UNIT_LAB EL_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.	Optional Dynamic
Scale-and-Range-Specification	MDC_ATTR_SCALE_SP ECN_I8 MDC_ATTR_SCALE_SP ECN_I16 MDC_ATTR_SCALE_SP ECN_I32	ScaleRangeSpec8 ScaleRangeSpec1 6 ScaleRangeSpec3 2	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.	Mandatory Dynamic
Sa-Specification	MDC_ATTR_SA_SPECN	SaSpec	This attribute describes the sample array and sample types.	Mandatory Static

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.

Refer to 8.9.3.3.2 for information on the control a manager can assert on the transmission of data from an agent.

— **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

There are currently no RT-SA object methods defined in this standard; however, ISO/IEEE 11073-104zz device specializations or proprietary definitions may exist.

6.3.5.5 RT-SA object events

There are currently no RT-SA object events defined in this standard.

6.3.5.6 Other RT-SA services

There are currently no uses of the RT-SA SET or GET services defined in this standard; however, ISO/IEEE 11073-104zz device specializations or proprietary definitions may exist.

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 IEEE Std 11073-10101) 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 9 defines the set of enumeration attributes that are supported for personal health device communication.

Table 9—Enumeration attributes

Attribute name	Attribute ID	Attribute type	Remark	Qualifiers
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 and only 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.	Conditional Observational

(Table continues)

IEEE Std 11073-20601-2019
 Health informatics—Personal health device communication
 Part 20601: Application profile—Optimized Exchange Protocol

Table 9—Enumeration attributes (*continued*)

Attribute name	Attribute ID	Attribute type	Remark	Qualifiers
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 and only 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.	Conditional Observational
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 and only 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.	Conditional Observational
Enum-Observed-Value-Simple-Str	MDC_ATTR_ENUM_OBS_VAL_SIMP_STR	EnumPrintableString	The value is reported as an ASCII printable string. One and only 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.	Conditional Observational
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 and only 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.	Conditional Observational
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.	Optional Dynamic
Capability-Mask-Simple	MDC_ATTR_ENUM_CAPABILITY_MASK_SIMPLE	BITS-32	This attribute indicates whether the corresponding bit in the Enum-Observed-Value-Simple-Bit-Str attribute is supported by the agent. If the Enum-Observed-Value-Simple-Bit-Str attribute is used, this attribute is mandatory. The value of this attribute is implementation specific. This attribute is dynamic, meaning that the attribute may change at some point after configuration.	Conditional Dynamic

(Table continues)

Table 9—Enumeration attributes (continued)

Attribute name	Attribute ID	Attribute type	Remark	Qualifiers
Capability-Mask-Basic	MDC_ATTR_ENUM_CAPABILITY_MASK_BASIC	BITS-16	This attribute indicates whether the corresponding bit in the Enum-Observed-Value-Basic-Bit-Str attribute is supported by the agent. If the Enum-Observed-Value-Basic-Bit-Str attribute is used, this attribute is mandatory. The value of this attribute is implementation specific. This attribute is dynamic, meaning that the attribute may change at some point after configuration.	Conditional Dynamic
State-Flag-Simple	MDC_ATTR_ENUM_STATE_FLAG_SIMPLE	BITS-32	This attribute indicates whether the corresponding bit in the Enum-Observed-Value-Simple-Bit-Str attribute is a state (1) or event (0). If the Enum-Observed-Value-Simple-Bit-Str attribute is used, this attribute is mandatory. The value of this attribute is implementation specific. This attribute is static, meaning that the attribute shall remain unchanged after the configuration is agreed upon.	Conditional Static
State-Flag-Basic	MDC_ATTR_ENUM_STATE_FLAG_BASIC	BITS-16	This attribute indicates whether the corresponding bit in the Enum-Observed-Value-Basic-Bit-Str attribute is a state (1) or event (0). If the Enum-Observed-Value-Basic-Bit-Str attribute is used, this attribute is mandatory. The value of this attribute is implementation specific. This attribute is static, meaning that the attribute shall remain unchanged after the configuration is agreed upon.	Conditional Static

6.3.6.4 Enumeration object methods

There are currently no enumeration object methods defined in this standard; however, ISO/IEEE 11073-104zz device specializations or proprietary definitions may exist.

6.3.6.5 Enumeration object events

There are currently no enumeration object events defined in this standard.

6.3.6.6 Other enumeration services

There are currently no uses of the enumeration SET or GET services defined in this standard; however, ISO/IEEE 11073-104zz device specializations or proprietary definitions may exist.

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.

Attribute values stored in a PM-segment may require the use of additional attributes from that object to describe the measurement; a common example is the unit of measure. If an attribute value in a PM-segment depends on an attribute value not stored in the PM-segment, then that dependent attribute shall not change value during the lifetime of the PM-segment. Otherwise, the agent shall store the dependent attribute value in the PM-segment.

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

Table 10—PM-store attributes

Attribute name	Attribute ID	Attribute type	Remark	Qualifiers
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.	Mandatory Static
PM-Store-Capab	MDC_ATTR_PM_STORE_CAPAB	PmStoreCapab	This attribute defines basic capabilities of the PM-store object instance.	Mandatory Static
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.	Mandatory Static

(Table continues)

Table 10—PM store attributes (*continued*)

Attribute name	Attribute ID	Attribute type	Remark	Qualifiers
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).	Optional Static
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).	Optional Dynamic
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.	Mandatory Dynamic
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.	Optional Static
Sample-Period	MDC_ATTR_TIME_PD_SAMP	RelativeTime	This attribute determines the frequency at which entries are added to the PM-segments. If values are sampled periodically, 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 so that 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).	Conditional Static
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 or the Get-Segment-Id-List method.	Mandatory Dynamic

(Table continues)

Table 10—PM store attributes (*continued*)

Attribute name	Attribute ID	Attribute type	Remark	Qualifiers
Clear-Timeout	MDC_ATTR_CLEAR_TIMEOUT	RelativeTime	<p>This timeout attribute defines the minimum time that the manager shall wait for the completion of a PM-store clear command.</p> <p>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. This attribute is required if the agent supports the clear segment action.</p>	Conditional Dynamic

6.3.7.4 PM-store object methods

Table 11 defines the methods (actions) of a PM-store object. These methods can be invoked using the ACTION service.

Table 11—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
Get-Segment-Id-List	Confirmed	MDC_ACT_SEG_GET_ID_LIST	(empty)	SegmIdList
Trig-Segment-Data-Xfer	Confirmed	MDC_ACT_SEG_TRIG_XFER	TrigSegmDataXferReq	TrigSegmDataXferRsp

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

If a manager supports the PM-store class, the support of sending the Get-Segment-Info, Get-Segment-Id-List, and Trig-Segment-Data-Xfer methods is mandatory. Support for sending the Clear-Segments method is optional.

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 valid, referencing an empty PM-segment, for the duration of the association. The Instance-Number of all other PM-segments shall be unaffected by clearing a segment.

Deletion of all selected PM-segments is not guaranteed by this method. While a PM-segment has the Operational-State attribute set to enabled it will not perform the requested deletion. Also, the agent may decide to protect certain segments from deletion making them read only (e.g., the user of the agent chose to “lock” certain data). Both protected and enabled segments will remain unaffected by a clear-segments operation.

If any of the selected segments are cleared, success (rors) shall be reported. However, success does not necessarily mean that all targeted segments were actually cleared (and potentially removed) since there may be a subset that were protected or enabled.

If all of the selected segments fail to clear (due to being protected or in the enabled state), the agent shall reply with a not-allowed-by-object error (roer). The return code shall be set to MDC_RET_CODE_OBJ_BUSY if any of the segments failed due to being in the enabled state. Otherwise, the return code shall be MDC_RET_CODE_UNKNOWN, which indicates that only agent protected segments were encountered during the operation.

For PM-segments cleared using the by time method and using absolute time, only PM-segments having Segment-Start-Abs-Time and Segment-End-Abs-Time fields entirely within the specified time period are cleared.

For PM-segments cleared using the by time method and using base time with offset, only PM-Segments having Segment-Start-BO-Time and Segment-End-BO-Time fields entirely within the specified time period are cleared. In using Segment-Start-BO-Time and Segment-End-BO-Time, the base time shall have a valid time (i.e., a nonzero value). If the offset field has a value 0x7FFF (32767), then only PM-segments having base time entirely within the specified base time period are cleared, otherwise for any other value of the offset field only PM-segments having local time (base time with offset added) entirely within the specified time period are cleared.

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.

If the agent supports the Clear-Segment method, the agent shall support at least the all-segments choice in the SegmSelection action-info-args of the Clear-Segments method.

If the agent supports the all-segments choice in the SegmSelection action-info-args of the Clear-Segments method, the agent shall set the pmsc-clear-segm-all-sup flag in the PM-Store-Capab attribute. If the agent supports the segm-id-list choice in the SegmSelection action-info-args of the Clear-Segments method, the agent shall set the pmsc-clear-segm-by-list-sup flag in the PM-Store-Capab attribute. If the agent supports the abs-time-range or bo-time-range choice in the SegmSelection action-info-args of the Clear-Segments method, the agent shall set the pmsc-clear-segm-by-time-sup flag in the PM-Store-Capab attribute.

If the agent does not support a Clear-Segments action from the manager, the agent shall respond with a no-such-action error (roer).

If the manager supports sending the Clear-Segments method, the manager shall support at least the all-segments choice in the SegmSelection action-info-args of the Clear-Segments method. The manager may support additional choices.

If no PM-Segment matches the selection criteria in the SegmSelection action-info-args so that no PM-segments are cleared by the action, then this is not an error, and a normal response is sent.

— **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 attributes and their data contents from the PM-segment object instances identified by the SegmSelection parameter.

The agent shall support the all-segments choice in the SegmSelection action-info-args of the Get-Segment-Info method. The agent may support the segm-id-list and/or abs-time-range and/or bo-time-range choice in the SegmSelection action-info-args of the Get-Segment-Info method. In this case the agent shall set the pmsc-segm-id-list-select and/or pmsc-abs-time-select flag in the PM-Store-Capab attribute. If the manager sends the Get-Segment-Info method with the choice that the agent does not support, the agent shall reply with an unsupported-choice error (roer).

For PM-segment info returned by time, the segments are selected using the same mechanism as described under Clear-Segments.

If the manager supports sending the Get-Segment-Info method, the manager shall support at least the choice all-segments in the SegmSelection action-info-args of the Get-Segment-Info method. The manager may support additional choices.

If a standard configuration contains any PM-Store object, the manager must send Get-Segment-info or Get-Segment-Id-List at the beginning of accessing any PM-Store object.

If no PM-Segment matches the selection criteria in the SegmSelection action-info-args so that no PM-segments are found by the action, then this is not an error, a normal response is sent, and the segment info list shall be empty.

If the choice of SegmSelection in the Get-Segment-Info method is segm-id-list and the segm-id-list is empty, then the response shall be a segment-info-list that is empty.

— **Get-Segment-Id-List**

This method allows the manager to retrieve a list of the instance numbers of all the PM-segments of a PM-store. In particular, the Get-Segm-Id-List method allows the manager to then retrieve the attributes of selected PM-segment object instances and their data contents without needing to retrieve information of all PM-segments. This also allows the manager to retrieve multiple PM-segments as a series of requests.

If a standard configuration contains any PM-Store object, the manager must send Get-Segment-Info or Get-Segment-Id-List at the beginning of accessing any PM-Store object.

If the agent supports the Get-Segment-Id-List method, the agent shall set the pmsc-get-segm-id-list-sup flag in the PM-Store-Capab attribute.

If the agent supports the Get-Segment-Id-List method, the agent shall also support the Get-Segment-Info method with the segm-id-list choice in the SegmSelection action info-args.

— **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 12 defines the potential events sent by a PM-store object.

Table 12—PM-store object events

Event	Mode	Event-type	Event-info parameter	Event-reply-info
Segment-Data-Event	Confirmed	MDC_NOTI_SEGMENT_DATA	SegmentDataEvent	SegmentDataResult
MDS-Dynamic-Data-Update-Var	Confirmed	MDC_NOTI_SCAN_REPORT_VAR	ScanReportInfoVar	—

— **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. The agent shall transfer all segment entries in order, first entry first [first-in, first out (FIFO)].

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

If confirmed event report is used by a PM-Store object, there shall be at most one unacknowledged confirmed event report outstanding from this object at any point in time.

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. If a manager does not have the current value of a needed PM-store attribute, then the GET service shall be used. An agent may also send scan event reports providing the manager with updates of the current attribute values, but this is not a mandated agent behavior.

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). See 8.9.3.4.2 for a full explanation of the GET operation.

6.3.7.6.2 SET service

There are currently no uses of the PM-store SET service defined in this standard; however, ISO/IEEE 11073-104zz device specializations or proprietary definitions may exist.

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 change dynamically. 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 13 defines the set of PM-segment attributes that are supported for personal health device communication.

Table 13 —PM-segment attributes

Attribute name	Attribute ID	Attribute type	Remark	Qualifiers
Instance-Number	MDC_ATTR_ID_INSTNO	InstNumber	The Instance-Number is the ID of a specific PM-segment object instance. Each instance shall have a unique number, within the context of a PM-store, assigned by the agent. It is used by the manager to address a PM-segment.	Mandatory
PM-Segment-Entry-Map	MDC_ATTR_PM_SEGMENT AP	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.	Mandatory

(Table continues)

Table 13—PM-segment attributes (continued)

Attribute name	Attribute ID	Attribute type	Remark	Qualifiers
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.	Conditional
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.	Mandatory
Sample-Period	MDC_ATTR_TIME_PD_SAMP	RelativeTime	This attribute defines the frequency at which entries are added to the PM-segment. If values are sampled periodically, 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, then the pmsc-peri-seg-entries bit in the PM-Store-Capab attribute shall be set.	Conditional
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.	Optional
Segment-Start-Abs-Time	MDC_ATTR_TIME_START_SEG	AbsoluteTime	This attribute defines the start time of segment. This attribute is required if the segment supports actions on the segment by time (i.e., the pmsc-abs-time-select and/or the pmsc-clear-segm-by-time-sup bits are set). If this attribute is used, the Segment-Start-BO-Time shall not be used.	Conditional

(Table continues)

IEEE Std 11073-20601-2019
 Health informatics—Personal health device communication
 Part 20601: Application profile—Optimized Exchange Protocol

Table 13—PM-segment attributes (*continued*)

Attribute name	Attribute ID	Attribute type	Remark	Qualifiers
Segment-End-Abs-Time	MDC_ATTR_TIME_END_SEG	AbsoluteTime	This attribute defines the end time of segment. This attribute is required if the segment supports actions on the segment by time (i.e., the pmsc-abs-time-select and/or the pmsc-clear-segm-by-time-sup bits are set). If this attribute is used, the Segment-End-BO-Time shall not be used.	Conditional
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 reports such an adjustment.	Conditional
Segment-Start-BO-Time	MDC_ATTR_TIME_START_SEG_BO	BaseOffsetTime	This attribute defines the start time of the segment as base time and the offset in minutes to local time. This attribute is required if the segment supports actions on the segment by time (i.e., the pmsc-abs-time-select and/or the pmsc-clear-segm-by-time-sup bits are set). Base time with offset is recommend if time adjustments are expected. If this attribute is used, the Segment-Start-Abs-Time shall not be used.	Conditional
Segment-End-BO-Time	MDC_ATTR_TIME_END_SEG_BO	BaseOffsetTime	This attribute defines the end time of segment as base time and the offset in minutes to local time. This attribute is required if the segment supports actions on the segment by time (i.e., the pmsc-abs-time-select and/or the pmsc-clear-segm-by-time-sup bits are set). Base time with offset is recommend if time adjustments are expected. If this attribute is used, the Segment-End-Abs-Time shall not be used.	Conditional
Segment-Usage-Count	MDC_ATTR_SEG_USAGE_CNT	INT-U32	This attribute gives the actual (current) number of stored entries.	Optional

(Table continues)

Table 13—PM-segment attributes (continued)

Attribute name	Attribute ID	Attribute type	Remark	Qualifiers
Segment-Statistics	MDC_ATTR_SEG_STAT_S	SegmentStatistics	This attribute defines the array for reporting minimum, mean, and maximum statistics for each element to be tagged.	Optional
Fixed-Segment-Data	MDC_ATTR_SEG_FIXED_DATA	N/A The data are stored internal to the device; therefore, 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-Segment-Data-Xfer method.	Mandatory
Confirm-Timeout	MDC_ATTR_CONFIRM_TIMEOUT	RelativeTime	<p>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.</p> <p>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.</p> <p>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 timeout 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.</p>	Optional

(Table continues)

Table 13—PM-segment attributes (continued)

Attribute name	Attribute ID	Attribute type	Remark	Qualifiers
Transfer-Timeout	MDC_ATTR_TRANSFER_TIMEOUT	RelativeTime	<p>This timeout attribute defines the minimum time that the manager shall wait for the complete transfer of PM-segment information.</p> <p>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.</p>	Mandatory

NOTE—The attribute qualifiers “static,” “dynamic,” and “observational” are omitted from Table 13 since PM-segments are dynamic (the object itself may change during the lifetime of an association).

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 that supports PM-stores shall be able to support the transfer of Fixed-Segment-Data attributes in multiple application messages.

6.3.8.4 PM-segment object methods

There are currently no PM-segment object methods defined in this standard.

6.3.8.5 PM-segment object events

There are currently no PM-segment object events defined in this standard.

6.3.8.6 Other PM-segment services

There are currently no uses of the PM-segment SET or GET services defined in this standard.

6.3.9 Scanner classes

6.3.9.1 General

A scanner serves two purposes: (1) It allows the manager to control the flow of data, and (2) it is an optimized packaging and reporting mechanism. It enables collected sets of attribute value changes (AttributeChangeSets) from one or more metric objects to be packaged together in a single event report in a

more efficient manner than can be done by using MDS events. A scanner implementation is either episodic or periodic, and both types of scanners may use variable, fixed, or group formats (see 7.4.5) in scan event reports. See Figure 5 for the class hierarchy of the scanner classes. Each class is described in 6.3.9.3 through 6.3.9.6, respectively.

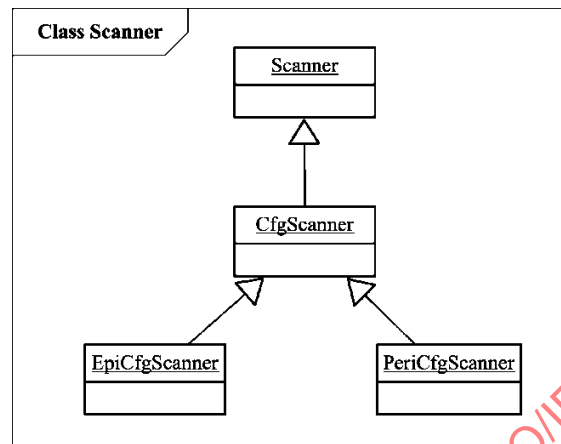


Figure 5—Personal health device—DIM—scanner model

6.3.9.2 Conceptual model

A scanner does not scan objects (that is it does not read the state of the object and report what is there and whether or not any changes have occurred). Instead, a scanner collects AttributeChangeSets and maps them to ObservationScans in scan event reports: Episodic scanners dispatch scan event reports when an episode completes. An episode is defined by the application but in general it is one or more AttributeChangeSets (where no two AttributeChangeSets come from the same object) that occur episodically (the time between episodes is unknown). Periodic scanners dispatch scan event reports when the period reported in the Reporting-Interval attribute expires, and in this case, more than one AttributeChangeSet may come from the same object.

Error conditions with the sensor, or other conditions such as no data available when the scanner is first enabled, may result in AttributeChangeSets not being present when the scan event report is generated. The scan event report that is generated in these cases depends upon the scanner type and format used as follows.

Episodic and periodic scanners using the group format shall create scan event reports where

- Any AttributeChangeSet containing observational attribute values from the numeric object (Table 7) is set to NaN. If the AttributeChangeSet contains the Measurement-Status attribute, then this value shall indicate either invalid or not-available as appropriate.
- Any AttributeChangeSet containing the Simple-Sa-Observed-Value of an RT-SA object shall include the Measurement-Status attribute. The Measurement-Status attribute shall indicate either invalid or not-available as appropriate. The Simple-Sa-Observed-Value values in this case are undefined and the manager shall attach no significance to the reported values.
- Any AttributeChangeSet containing observational attribute values from the enumeration object (Table 9) shall be set to a suitable enumeration, and where appropriate include the Measurement-Status attribute. The Measurement-Status attribute shall indicate either invalid or not-available as appropriate. If the Measurement-Status attribute is used to indicate invalid or not-available, the manager shall attach no significance to the reported value.

- If the scanner is episodic and no AttributeChangeSets are collected, the scan event report shall not be sent.
- If the scanner is periodic and no AttributeChangeSets are collected, an empty scan event report shall be sent.

Episodic and periodic scanners using the variable or fixed format shall create scan event reports where

- Only the collected AttributeChangeSets are mapped to the ObservationScans.
- If the scanner is episodic and no AttributeChangeSets are collected, the scan event report shall not be sent.
- If the scanner is periodic and no AttributeChangeSets are collected, an empty scan event report shall be sent.

The periodic scanner is distinct from the episodic scanner by being able to collect multiple AttributeChangeSets from the same object before dispatching. The periodic scanner also requires that the rate of generation of all collected AttributeChangeSets shall have a fixed timing relationship with the period of the periodic scanner. Where an AttributeChangeSet does not have an explicit time stamp, its time stamp shall be inferred from the time stamp of the scan event report. It follows that any AttributeChangeSet that is collected at a time different from the time of the scan event report shall be reported with its own time stamp. The periodic scanner shall insert AttributeChangeSets from the same object in the scan event report in strict chronological order starting with the oldest at the head of message.

Different observation reporting requirements may be accomplished by using a set of periodic and episodic scanners, one to manage the characteristics of each observation stream. For example, a pulse oximeter might use a periodic configurable scanner with a Reporting-Interval of 50 ms for the RT-SA object representing the plethysmogram, a periodic configurable scanner with a Reporting-Interval of 1 s for the numeric representing the saturated oxygen level and any enumeration objects reporting on events associated with the value, and an episodic configurable scanner for beat-to-beat metric objects (numeric or enumeration).

6.3.9.3 Scanner class

6.3.9.3.1 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 6.3.9.5.5, 6.3.9.6.5, and A.11.5, respectively.

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

6.3.9.3.2 Scanner class identification

The nomenclature code to identify the scanner class is MDC_MOC_SCAN.

6.3.9.3.3 Scanner class attributes

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

Table 14—Scanner attributes

Attribute name	Attribute ID	Attribute type	Remark	Qualifiers
Handle	MDC_ATTR_ID_HANDLE	HANDLE	Scanners are identified by handles.	Mandatory Static
Operational-State	MDC_ATTR_OP_STAT	OperationalState	This attribute indicates whether the scanner is sending event reports or not. If the scanner is sending event reports, the attribute value shall be set to enabled; otherwise, it shall be set to disabled. The manager shall use the SET action to request a change in the value of this attribute.	Mandatory Dynamic
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 an AttributeChangeSet emanates from that object. For periodic scanners, the collected AttributeChangeSets from the objects are reported in each period. The manager shall not assume the order of the ObservationScans 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. This attribute shall be set prior to sending such a report. This attribute value may change in between event reports as long as the scanner is disabled. The attribute value change is communicated to the manager using an agent-initiated event report from the MDS object.	Conditional Dynamic

(Table continues)

Table 14—Scanner attributes (*continued*)

Attribute name	Attribute ID	Attribute type	Remark	Qualifiers
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. If any of these four reporting styles are used, this attribute shall be set prior to sending such a report. This attribute value may change in between event reports as long as the scanner is disabled. The attribute value change is communicated to the manager using an agent-initiated event report from the MDS object.	Conditional Dynamic

6.3.9.3.4 Scanner object methods

There are currently no scanner object methods defined in this standard; however, ISO/IEEE 11073-104zz device specializations or proprietary definitions may exist.

6.3.9.3.5 Scanner object events

See the derived class event descriptions in 6.3.9.5.5 and 6.3.9.6.5.

6.3.9.3.6 Other scanner services— **GET service:**

There is currently no GET service defined in this standard; however, ISO/IEEE 11073-104zz device specializations or proprietary definitions may exist.

— **SET service:**

Agents that have scanner-derived objects shall support the SET service for the Operational-State attribute of the scanner objects, and this SET service may be invoked as either a confirmed action or an unconfirmed action.

6.3.9.4 CfgScanner class

6.3.9.4.1 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.

6.3.9.4.2 Configurable scanner class identification

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

6.3.9.4.3 Configurable scanner class attributes

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

Table 15—Configurable scanner attributes

Attribute name	Attribute ID	Attribute type	Remark	Qualifiers
Confirm-Mode	MDC_ATTR_CONFIRM_MODE	ConfirmMode	<p>This attribute defines whether event reports are sent confirmed or unconfirmed.</p> <p>Currently the attribute is dynamic, but agent implementations should treat as static since it is likely this will change to static in the future. If this attribute value is changed, it shall be changed only when the scanner is disabled. The attribute value change is communicated to the manager using an agent-initiated event report from the MDS object.</p>	Mandatory Dynamic

(Table continues)

Table 15—Configurable scanner attributes (*continued*)

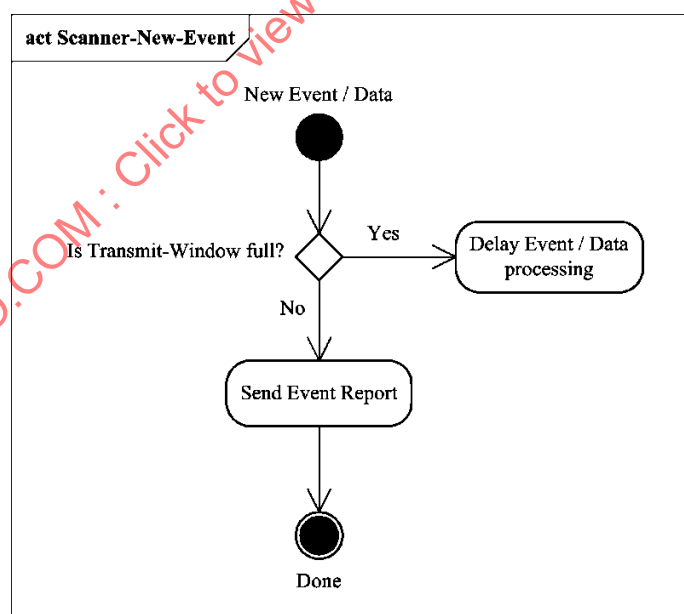
Attribute name	Attribute ID	Attribute type	Remark	Qualifiers
Confirm-Timeout	MDC_ATTR_CONFIRM_TIMEOUT	RelativeTime	<p>This informational timeout attribute defines the minimum time that the agent shall wait for a Response message from the manager after issuing a Confirmed Event Report invoke message before timing out and transitioning to the Unassociated state.</p> <p>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.</p> <p>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.</p> <p>Currently the attribute is dynamic, but agent implementations should treat as static since it is likely this will change to static in the future. If this attribute value is changed, it shall be changed only when the scanner is disabled. The attribute value change is communicated to the manager using an agent-initiated event report from the MDS object.</p>	Optional Dynamic

(Table continues)

Table 15—Configurable scanner attributes (*continued*)

Attribute name	Attribute ID	Attribute type	Remark	Qualifiers
Transmit-Window	MDC_ATTR_TX_WIND	INT-U16	<p>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 standard, the attribute shall have only a value of 1.</p> <p>Currently the attribute is dynamic, but agent implementations should treat as static since it is likely this will change to static in the future. If this attribute value is changed, it shall be changed only when the scanner is disabled. The attribute value change is communicated to the manager using an agent-initiated event report from the MDS object.</p>	Optional Dynamic

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

**Figure 6—Configurable scanner Transmit-Window handling**

6.3.9.4.4 Configurable scanner object methods

There are currently no configurable scanner object methods defined in this standard; however, ISO/IEEE 11073-104zz device specializations or proprietary definitions may exist.

6.3.9.4.5 Configurable scanner object events

See the derived class event descriptions in 6.3.9.5.5 and 6.3.9.6.5.

6.3.9.4.6 Other configurable scanner services

There are currently no uses of the configurable scanner SET or GET services defined in this standard; however, ISO/IEEE 11073-104zz device specializations or proprietary definitions may exist.

6.3.9.5 EpiCfgScanner class

6.3.9.5.1 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.

6.3.9.5.2 Episodic configurable scanner class identification

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

6.3.9.5.3 Episodic configurable scanner class attributes

Table 16 defines the set of episodic configurable scanner attributes that are supported for personal health device communication

Table 16—Episodic configurable scanner attributes

Attribute name	Attribute ID	Attribute type	Remark	Qualifiers
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.	Optional Dynamic

6.3.9.5.4 Episodic configurable scanner object methods

There are currently no episodic configurable scanner object methods defined in this standard; however, ISO/IEEE 11073-104zz device specializations or proprietary definitions may exist.

6.3.9.5.5 Episodic configurable scanner object events

Table 17 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 17. If a manager supports episodic scanners, it shall support all the events identified in Table 17.

Table 17—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—For variable and fixed format reports, if no AttributeChangeSet is collected from an object, then no AttributeChangeSet from this object is included in the scan report. If no AttributeChangeSets are collected, no scan event report is sent.

— **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 event style is used when the scanner object is used to send the data in its most compact format. The Scan-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 event style is the same as Unbuf-Scan-Report-Var, but allows inclusion of data from multiple persons.

— **Unbuf-Scan-Report-MP-Fixed:**

This event style is the same as Unbuf-Scan-Report-Fixed, but allows inclusion of data from multiple persons.

— **Unbuf-Scan-Report-MP-Grouped:**

This event style is the same as Unbuf-Scan-Report-Grouped, but allows inclusion of data from multiple persons.

6.3.9.5.6 Other episodic configurable scanner services

There are currently no uses of the episodic configurable scanner SET or GET services defined in this standard; however, ISO/IEEE 11073-104zz device specializations or proprietary definitions may exist.

6.3.9.6 PeriCfgrScanner class

6.3.9.6.1 General

The PeriCfgrScanner class represents a class that can be instantiated. PeriCfgrScanner objects are used to send reports containing periodic data. 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.

When a periodic configurable scanner is enabled by a manager, scan reports should be sent within a reasonable time and synchronized to the reporting interval of the scanner. The time between the scanner being enabled and the sending of the first scan report should be within the reporting interval plus 15 s.

NOTE—The 15 s is expected to provide an ample initialization time.

Example: A periodic configurable scanner is set up to report on 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 0.5 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. The objects in a Scan-Handle-Attr-Val-Map would contain two entries for the object with a 0.5 s update interval.

6.3.9.6.2 Periodic configurable scanner object identification

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

6.3.9.6.3 Periodic configurable scanner object attributes

Table 18 defines the set of scanner object attributes that are supported for personal health device communication.

Table 18—Periodic configurable scanner object attributes

Attribute name	Attribute ID	Attribute type	Remark	Qualifiers
Reporting-Interval	MDC_ATTR_SCAN_REP_PD	RelativeTime	Reporting period of the event reports.	Mandatory Dynamic

6.3.9.6.4 Periodic configurable scanner object methods

There are currently no periodic configurable scanner object methods defined in this standard; however, ISO/IEEE 11073-104zz device specializations or proprietary definitions may exist.

6.3.9.6.5 Periodic configurable scanner object events

Table 19 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 19. If a manager supports periodic scanners, it shall support all the events identified in Table 19.

Table 19—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_RE PORT_VAR	ScanReportInfoVar	—
Buf-Scan-Report-Fixed	Confirmed or unconfirmed	MDC_NOTI_BUF_SCAN_RE PORT_FIXED	ScanReportInfoFixed	—
Buf-Scan-Report-Grouped	Confirmed or unconfirmed	MDC_NOTI_BUF_SCAN_RE PORT_GROUPED	ScanReportInfoGrouped	—
Buf-Scan-Report-MP-Var	Confirmed or unconfirmed	MDC_NOTI_BUF_SCAN_RE PORT_MP_VAR	ScanReportInfoMPVar	—
Buf-Scan-Report-MP-Fixed	Confirmed or unconfirmed	MDC_NOTI_BUF_SCAN_RE PORT_MP_FIXED	ScanReportInfoMPFixed	—
Buf-Scan-Report-MP-Grouped	Confirmed or unconfirmed	MDC_NOTI_BUF_SCAN_RE PORT_MP_GROUPED	ScanReportInfoMPGrouped	—

NOTE—For variable and fixed format reports, if no AttributeChangeSet is collected from an object, no AttributeChangeSets from that object are included in the scan event report. If no AttributeChangeSets are collected, then an empty scan event report is generated when the period expires.

All of the event report styles listed in Table 19 are buffered equivalents to their unbuffered counterparts in 6.3.9.5.5. 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.

6.3.9.6.6 Other periodic configurable scanner services

There are currently no uses of the periodic configurable scanner SET or GET services defined in this standard; however, ISO/IEEE 11073-104zz device specializations or proprietary definitions may exist.

6.3.10 PHD DM Status object

6.3.10.1 General

This object is an instance of the enumeration class. The PHD DM Status object allows generic notification handling for personal health devices. The status of the device is represented in a number of bit flags covering warning, error, service, and undetermined statuses. Changes in status are sent as events notifying the manager.

This represents a generic notification mechanism for a device to report warnings, errors, and other abnormal statuses to a manager that may provide visibility of these status change events to the user.

6.3.10.2 PHD DM Status object attributes

Table 20 summarizes the attributes of the PHD DM status enumeration object.

Table 20—PHD DM status enumeration object attributes

Attribute name	Attribute ID	Attribute type	Remark	Qualifier
Handle	MDC_ATTR_ID_HANDLE	HANDLE	See 6.3.3.3.	Mandatory
Type	MDC_ATTR_ID_TYPE	TYPE	MDC_PART_PHD_DM MDC_PHD_DM_DEV_STAT	Mandatory
Supplemental-Types	MDC_ATTR_SUPPLEMENTAL_TYPES	SupplementalTypeList	See 6.3.3.3.	Optional
Metric-Spec-Small	MDC_ATTR_METRIC_SPEC_SMALL	MetricSpecSmall	mss-avail-intermittent, mss-avail-stored-data, mss-upd-aperiodic, mss- acc-manager-initiated, mss-acc-agent-initiated, mss-msmt-manager- initiated-immediate	Mandatory
Metric-Structure-Small	MDC_ATTR_METRIC_STRUCTURE_SMALL	MetricStructureSmall	See 6.3.3.3.	Not Recommended
Measurement-Status	MDC_ATTR_MSMT_STATUS	MeasurementStatus	See 6.3.3.3.	Not Recommended
Metric-Id	MDC_ATTR_ID_PHYSIO	OID-Type	See 6.3.3.3.	Not Recommended
Metric-Id-List	MDC_ATTR_ID_PHYSIO_LIST	MetricIdList	See 6.3.3.3.	Not Recommended
Metric-Id-Partition	MDC_ATTR_METRIC_ID_PARTITION	NomPartition	See 6.3.3.3.	Not Recommended
Attribute-Value-Map	MDC_ATTR_ATTRIBUTE_VALUE_MAP	AttrValMap	See 6.3.3.3.	Conditional
Source-Handle-Reference	MDC_ATTR_SOURCE_HANDLE_REF	HANDLE	See 6.3.3.3.	Not Recommended
Label-String	MDC_ATTR_ID_LABEL_STRING	OCTET STRING	See 6.3.3.3.	Not Recommended
Absolute-Time-Stamp	MDC_ATTR_TIME_STAMP_ABS	AbsoluteTime	See 6.3.3.3.	Conditional
Base-Offset-Time-Stamp	MDC_ATTR_TIME_STAMP_BO	BaseOffsetTime	See 6.3.3.3.	Conditional
Relative-Time-Stamp	MDC_ATTR_TIME_STAMP_REL	RelativeTime	See 6.3.3.3.	Conditional
HiRes-Time-Stamp	MDC_ATTR_TIME_STAMP_REL_HI_RES	HighResRelativeTime	See 6.3.3.3.	Conditional
Measure-Active-Period	MDC_ATTR_TIME_PERIOD_MSMT_ACTIVE	FLOAT-Type	See 6.3.3.3.	Not Recommended
Enum-Observed-Value-Simple-Bit-Str	MDC_ATTR_ENUM_OBSERVED_VALUE_SIMP_BIT_STR	BITS-32	See Table 21.	Mandatory
Capability-Mask-Simple	MDC_ATTR_ENUM_CAPABILITY_MASK_SIMPLE	BITS-32	See 6.3.6.3.	Mandatory
State-Flag-Simple	MDC_ATTR_ENUM_STATE_FLAG_SIMPLE	BITS-32	See 6.3.6.3.	Mandatory

NOTE—See Table 9 for information on whether an attribute is static or dynamic.

The observed value reported in this object is the general device status. Some attributes of Enumeration class are not allowed in this object and thus are not contained in Table 20.

The explicit expression of the existence of annunciations is realized by the setting of the appropriate bit in the Enum-Observed-Value-Simple-Bit-Str attribute, as defined in Table 21. Anytime the status changes for any monitored condition, the agent shall report on all monitored conditions.

The detection of the condition change could take some time. If a delay occurs in detecting the start or stop of a condition, then the event shall be reported with a time stamp that is the time of the occurrence of the respective event rather than the time that the event is reported.

Within a device specification, if an acceptable, existing bit is not available, device-status-undetermined shall be used. A manager shall interpret these bits only within the context of this attribute and only within that device specialization, as other specializations may use corresponding terms for different purposes.

Table 21 —Mapping of PHD DM status to object Bit-Str attribute

PHD DM status condition	PHDDMStat mnemonic
Agent reports that an undetermined or not supported condition occurred.	device-status-undetermined
Agent reports that a reset has occurred.	device-status-reset
Agent reports that a general fault occurred.	device-status-error
Agent reports that a mechanical fault occurred.	device-status-error-mechanical
Agent reports that an electronic fault occurred.	device-status-error-electronic
Agent reports that a software error occurred.	device-status-error-software
Agent reports that a battery fault occurred.	device-status-error-battery
Agent reports that a general service is required.	device-status-service
Agent reports that a time synchronization is required.	device-status-service-time-sync-required
Agent reports that a calibration is required.	device-status-service-calibration-required
Agent reports that a component replenishment is required.	device-status-service-replenishment-required
Agent reports that battery power is low.	device-status-battery-low
Agent reports that battery is depleted.	device-status-battery-depleted
Agent reports that battery has been replaced.	device-status-battery-replaced
Agent reports that battery is interrupted. The use of battery is interrupted, e.g., due to the switch of device mode or switch to wall power.	device-status-battery-interrupted

NOTE 1—The bits in Table 21 are defined to 0 = False and 1 = True.

NOTE 2—The specific bit mappings of PHDDMStat are defined in A.3.11.

NOTE 3—All bits not defined in Table 21 or A.3.11 are reserved for future use.

NOTE 4—An agent is not required to implement all the features specified in Table 21.

NOTE 5—The device-status-battery-interrupted is used only to report unexpected disconnection of the battery, rather than any planned switch between mains and battery, which is reported through the power-status attribute of MDS.

6.3.10.3 PHD DM Status object methods

There are currently no PHD DM Status object methods defined in this standard; however, ISO/IEEE 11073-104zz device specializations or proprietary definitions may exist.

6.3.10.4 PHD DM Status object events

There are currently no PHD DM Status object events defined in this standard.

6.3.10.5 Other PHD DM Status services

There are currently no uses of the PHD DM Status SET or GET services defined in this standard; however, ISO/IEEE 11073-104zz device specializations or proprietary definitions may exist.

6.3.11 Schedule-store class

6.3.11.1 General

In the context of personal health devices, an agent may contain schedules that indicate when certain events will take place. An example of such an agent is the insulin pump, which would contain a schedule to indicate the time and dose of the insulin delivery. The schedule-store class allows a manager to retrieve the schedules from an agent and to determine which is the active schedule when several schedules are defined. The data held in schedule-store objects may be deleted by user actions via the manager or user interface on the device, and the capacity is limited only by the agent's data storage capabilities.

An instance of the schedule-store class provides storage capabilities for a series of schedules. Data related to these schedules are stored in a variable number of schedule-segment objects (see 6.3.12). The stored schedule data of the schedule-store object are requested from the agent by the manager using object access services (see 7.3). Anybody not familiar with the schedule-store concept may wish to read Annex K for a conceptual overview prior to reading the following subclauses.

Attribute values stored in a schedule-segment may require the use of additional attributes from that object to describe the schedule; a common example is the duration. If an attribute value in a schedule-segment depends on an attribute value not stored in the schedule-segment, then that dependent attribute shall not change value during the lifetime of the schedule-segment. Otherwise, the agent shall store the dependent attribute value in the schedule-segment.

6.3.11.2 Schedule-store class identification

The nomenclature code to identify the schedule-store class is MDC_MOC_VMO_SCHEDSTORE.

6.3.11.3 Schedule-store class attributes

Table 22 defines the set of schedule-store attributes that are supported for personal health device communication.

Table 22 —Schedule-store attributes

Attribute name	Attribute ID	Attribute type	Remark	Qualifiers
Schedule-Handle	MDC_ATTR_SCHED_STORE_HANDLE	HANDLE	The Schedule 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.	Mandatory Static
Active-Schedule-Segment-Instance-Number	MDC_ATTR_SCHED_STORE_ACTIVE_INSTNO	InstNumber	The Active-Schedule-Segment-Instance-Number is the unique ID of the specific schedule-segment object that is active. If no schedule segment is currently active or there are currently no schedule segments, the value shall be 0.	Mandatory Dynamic
Updated-Schedule-Segment-Instance-Number-List	MDC_ATTR_SCHED_STORE_UPDATED_INSTNO	InstNumberList	The Updated-Schedule-Segment-Instance-Number-list is the list of schedule-segment object IDs reporting all updated schedule segments (i.e., created, modified, or deleted).	Mandatory Observational
Schedule-Store-Capab	MDC_ATTR_SCHED_STORE_CAPAB	SchedStoreCapab	This attribute defines basic capabilities of the schedule-store object instance.	Mandatory Static
Schedule-Store-Capacity-Count	MDC_ATTR_SCHED_STORE_CAPAC_CNT	INT-U32	This attribute is the maximum number of stored schedule-segment entries (entries in all contained schedule segments).	Optional Static
Schedule-Store-Usage-Count	MDC_ATTR_SCHED_STORE_USAGE_CNT	INT-U32	This attribute is the actual number of currently stored schedule-segment entries (entries in all contained Schedule segments).	Optional Dynamic
Schedule-Store-Operational-Status	MDC_ATTR_SCHED_STORE_OP_STAT	OperationalState	The attribute indicates if new entries are currently being inserted in any of the contained schedule segments. If any schedule segment contained by this schedule-store is being modified, this attribute shall be set to enabled. Otherwise, it shall be set to disabled.	Mandatory Dynamic
Schedule-Store-Label	MDC_ATTR_SCHED_STORE_LABEL_STRING	OCTET STRING	This attribute is an application-dependent label for the schedule-store in printable ASCII to indicate its intended use and may be used for display purposes.	Optional Static

(Table continues)

Table 22—Schedule-store attributes (continued)

Attribute name	Attribute ID	Attribute type	Remark	Qualifiers
Number-Of-Schedule-Segments	MDC_ATTR_SCHED_SEG_NUM	INT-U16	This attribute is the number of currently instantiated schedule segments contained in the schedule-store. Note that the schedule-segment attribute Instance-Number is NOT related to this number (i.e., Instance-Number does not need to be in the range from 1 to Number-Of-Schedule-Segments).	Mandatory Dynamic

6.3.11.4 Schedule-store object methods

Table 23 defines the methods (actions) of a schedule-store object. These methods can be invoked using the ACTION service.

Table 23—Schedule-store methods

Method/Action	Mode	Action-type	action-info-args	Resulting action-info-args
Get-Schedule-Segment-Info	Confirmed	MDC_ACT_SCHED_SEG_GET_INFO	SchedSegmSelection	SchedSegmentInfoList
Get-Schedule-Segment-Id-List	Confirmed	MDC_ACT_SCHED_SEG_GET_ID_LIST	(empty)	SchedSegmIdList
Trig-Schedule-Segment-Data-Xfer	Confirmed	MDC_ACT_SCHED_SEG_TRIG_XFER	TrigSchedSegmDataXferReq	TrigSchedSegmDataXferRsp

If an agent supports the schedule-store class, the support of the Get-Schedule-Segment-Info or Get-Schedule-Segment-Id-List methods is mandatory, and support of the Trig-Schedule-Segment-Data-Xfer method is mandatory.

If a manager supports the schedule-store class, the support of sending the Get-Schedule-Segment-Info, Get-Schedule-Segment-Id-List, and Trig-Schedule-Segment-Data-Xfer methods is mandatory.

— Get-Schedule-Segment-Info:

This method allows the manager to retrieve Schedule-segment attributes of one or more schedule segments, with the exception of the Fixed-Schedule-Segment-Data attribute which contains the actual schedule data and is retrieved by using the Trig-Schedule-Segment-Data-Xfer method. In particular, the Get-Schedule-Segment-Info method allows the manager to retrieve the attributes and their data contents from the Schedule-segment object instances identified by the SchedSegmSelection parameter.

The agent shall support the all-sched-segments choice in the SchedSegmSelection action-info-args of the Get-Schedule-Segment-Info method. The agent may support the sched-segm-id-list choice in the SchedSegmSelection action-info-args of the Get-Schedule-Segment-Info method. In this case the agent shall set the schedsc-segm-id-list-select flag in the Schedule-Store-Capab attribute. If the manager sends the Get-Schedule-Segment-Info method with the choice that the agent does not support, the agent shall reply with an unsupported-choice error (roer).

If the manager supports sending the Get-Schedule-Segment-Info method, the manager shall support at least the choice all-sched-segments in the SchedSegmSelection action-info-args of the Get-Schedule-Segment-Info method. The manager may support additional choices.

If there is no schedule segment that matches the selection criteria in the SchedSegmSelection action-info-args such that no schedule segments are found by the action, then this is not an error and a normal response is sent, and the segment info list shall be empty.

If the choice of SchedSegmSelection in the Get-Schedule-Segment-Info method is sched-segm-id-list and the sched-segm-id-list is empty then the response shall be a sched-segment-info-list that is empty.

If the agent supports the Get-Schedule-Segment-Info method, the agent shall set the schedsc-get-segm-info-sup flag in the Schedule-Store-Capab attribute.

— **Get-Schedule-Segment-Id-List:**

This method allows the manager to retrieve a list of the instance numbers of all the schedule segments of a schedule-store. In particular, the Get-Schedule-Segm-Id-List method allows the manager to then retrieve the attributes of selected schedule segment object instances and their data contents without need to retrieve information of all schedule segments. This also allows the manager to retrieve multiple schedule segments as a series of requests.

If the agent supports the Get-Schedule-Segment-Id-List method, the agent shall set the schedsc-get-segm-id-list-sup flag in the Schedule-Store-Capab attribute.

If the agent supports the Get-Schedule-Segment-Id-List method, the agent shall support the action of Get-Schedule-Segment-Info by ID list.

— **Trig-Schedule-Segment-Data-Xfer:**

This method allows the manager to start the transfer of the Fixed-Schedule-Segment-Data attribute of a specified schedule segment. The agent indicates in the response if it accepts or denies this request. If the agent accepts the request, the agent sends Schedule-Segment-Data-Event messages as described in 6.3.11.5. If this method is invoked on a schedule segment that is being updated such that the Operational-State attribute of the schedule store is 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.

6.3.11.5 Schedule-store object events

Table 24 defines the potential events sent by a schedule-store object.

Table 24—Schedule-store events

Event	Mode	Event-type	Event-info parameter	Event-reply-info
Schedule-Segment-Data-Event	Confirmed	MDC_NOTI_SCHED_SEG_DATA	ScheduleSegmentDataEvent	ScheduleSegmentDataResult
MDS-Dynamic-Data-Update-Var	Confirmed	MDC_NOTI_SCAN_REPORT_VAR	ScanReportInfoVar	—

— **Schedule-Segment-Data-Event:**

This event sends data stored in the Fixed-Schedule-Segment-Data of a schedule segment from the agent to the manager. The event is triggered by the manager by the Trig-Schedule-Segment-Data-Xfer method. Once the data transfer is triggered, the agent sends Schedule-Segment-Data-Event

messages until the complete Fixed-Schedule-Segment-Data is transferred or the transfer is aborted by the manager or agent. See Transfer Schedule-segment content in 8.9.3.5.2 for a full description. It is encouraged to place as many segment entries contained in a Schedule-Segment-Data-Event as possible to reduce the number of messages required for the transfer of the segment. The agent shall transfer all segment entries in order, first entry first.

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

If the confirmed event report is used by a schedule-store object, there shall be at most one unacknowledged confirmed event report outstanding from this object at any point in time.

6.3.11.6 Other Schedule-store services

6.3.11.6.1 GET service

Support for the GET service shall be provided by any agent that supports one or more schedule-store objects only while in the Operating state. The manager uses the GET service to retrieve the values of all schedule-store object attributes. If a manager does not have the current value of a needed schedule-store attribute, then the GET service shall be used. An agent shall send scan event reports providing the manager with updates of the current attribute values.

The manager may request the schedule-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 schedule-store object, as defined in the agent’s configuration. The agent shall respond by reporting its schedule-store object attributes to the manager using the “Remote Operation Response | Get” response (see *rors-cmip-get* in A.10.2). See 8.9.3.5.2 for a full explanation of the GET operation for schedule-store data transmission.

6.3.11.6.2 SET service

There are currently no SET services defined for schedule-store objects in this .

6.3.12 Schedule-segment class

6.3.12.1 General

An instance of the schedule-segment represents stored settings that define a series of scheduled entries. A schedule-segment object is not part of the static agent configuration because the number of instantiated schedule-segment instances may change dynamically. The manager accesses schedule-segment objects indirectly by methods and events of the schedule-store object.

6.3.12.2 Schedule-segment class identification

The nomenclature code to identify the schedule-segment class is `MDC_MOC_SCHEDULE_SEG`.

6.3.12.3 Schedule-segment class attributes

Table 25 defines the set of schedule-segment attributes that are supported for personal health device communication.

Table 25—Schedule-segment attributes

Attribute name	Attribute ID	Attribute type	Remark	Qualifiers
Schedule-Segment-Instance-Number	MDC_ATTR_SCHED_SEG_INSTNO	InstNumber	The Schedule-Segment-Instance-Number is the ID of a specific schedule-segment object instance. Each instance shall have a unique number assigned by the agent starting at 1. It is used by the manager to address a schedule segment. The Instance-Number value of 0 shall be reserved for reporting no active schedule-segment in the schedule-store.	Mandatory Static
Schedule-Segment-Entry-Map	MDC_ATTR_SCHED_SEG_MAP	ScheduleSegmentEntryMap	This attribute defines the format and contents of one schedule entry. An entry has a conditional header containing information applicable to all elements in the entry. If the schedule-segment period, start time, and entry interval are used, then the header information shall not be used. The entry then contains one or more elements, defined by the handle and an attribute value map defining the object attributes for each element in the schedule segment.	Mandatory Dynamic
Schedule-Segment-Period	MDC_ATTR_SCHED_SEG_PERIOD	HighResRelativeTime	This attribute specifies the period over which the entries the schedule segment shall repeat. If the schedule segment executes only once, then the value shall be 0.	Mandatory Dynamic
Schedule-Segment-Entry-Interval	MDC_ATTR_SCHED_SEG_ENTRY_INTERVAL	HighResRelativeTime	This attribute specifies the interval between each entry in the schedule segment.	Conditional Dynamic

(Table continues)

Table 25—Schedule-segment attributes (*continued*)

Attribute name	Attribute ID	Attribute type	Remark	Qualifiers
Schedule-Segment-Person-Id	MDC_ATTR_SCHED_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 schedule segment is able to have schedule data for multiple persons, it shall set the schedsc-multi-person bit in the Sched-Store-Capab attribute. If this bit is set, all schedule-segment instances shall support the Schedule-Segment-Person-Id attribute. Otherwise, this attribute is not defined.	Conditional Dynamic
Schedule-Segment-Entry-Count	MDC_ATTR_SCHED_SEG_ENTRY_CNT	INT-U32	This attribute gives the actual number of schedule-segment entries.	Optional Dynamic
Schedule-Segment-Label	MDC_ATTR_SCHED_SEG_LABEL_STRING	OCTET STRING	This attribute is an application-dependent label in printable ASCII for the schedule segment to indicate its intended use and may be used for display purposes.	Optional Static
Schedule-Segment-LastUpdated-Abs-Time	MDC_ATTR_SCHED_SEG_LAST_UPDATED_ABS_TIME	AbsoluteTime	This attribute defines the time the schedule segment was last updated. If this attribute is used, neither the Schedule-Segment-LastUpdated-HiRes-Time nor Schedule-Segment-LastUpdated-BO-Time shall be used.	Conditional Dynamic
Schedule-Segment-LastUpdated-HiRes-Time	MDC_ATTR_SCHED_SEG_LAST_UPDATED_HIRES_TIME	HiResRelativeTime	This attribute defines the time the schedule segment was last updated. If this attribute is used, neither the Schedule-Segment-LastUpdated-Abs-Time nor Schedule-Segment-LastUpdated-BO-Time shall be used.	Conditional Dynamic
Schedule-Segment-LastUpdated-BO-Time	MDC_ATTR_SCHED_SEG_LAST_UPDATED_BO_TIME	BaseOffsetTime	This attribute defines the time the schedule segment was last updated. If this attribute is used, neither the Schedule-Segment-LastUpdated-Abs-Time nor Schedule-Segment-LastUpdated-HiRes-Time shall be used.	Conditional Dynamic

(Table continues)

Table 25—Schedule-segment attributes (*continued*)

Attribute name	Attribute ID	Attribute type	Remark	Qualifiers
Schedule-Segment-Reference-Abs-Time	MDC_ATTR_SCHED_SEG_REF_ABS_TIME	AbsoluteTime	This attribute defines the time from which all entries within the schedule segment are referenced. This attribute shall not be used, if the Schedule-Segment-Reference-BO-Time is used. If neither of Schedule-Segment-Reference-Abs-Time or Schedule-Segment-Reference-BO-Time is used, the reference time is when the schedule segment is activated.	Conditional Dynamic
Schedule-Segment-Reference-BO-Time	MDC_ATTR_SCHED_SEG_REF_BO_TIME	BaseOffsetTime	This attribute defines the time from which all entries within the schedule segment are referenced. This attribute shall not be used, if the Schedule-Segment-Reference-Abs-Time is used. If neither of Schedule-Segment-Reference-Abs-Time or Schedule-Segment-Reference-BO-Time is used, the reference time is when the schedule segment is activated.	Conditional Dynamic
Schedule-Segment-Start-Abs-Time	MDC_ATTR_SCHED_SEG_START_ABS_TIME	AbsoluteTime	This attribute defines the time of the first execution of the schedule segment. If this attribute is used, the Schedule-Segment-Start-BO-Time shall not be used. If this attribute is used, the schedule segment becomes the active-schedule-segment in the schedule-store at the start time. A schedule segment shall not be active before the start time.	Optional Dynamic
Schedule-Segment-End-Abs-Time	MDC_ATTR_SCHED_SEG_END_ABS_TIME	AbsoluteTime	This attribute defines the expiration of the schedule segment. If this attribute is used, the Schedule-Segment-End-BO-Time shall not be used. If this attribute is used and the end time is reached, the schedule-segment shall no longer be the active-schedule-segment in the schedule-store.	Optional Dynamic

(Table continues)

IEEE Std 11073-20601-2019
 Health informatics—Personal health device communication
 Part 20601: Application profile—Optimized Exchange Protocol

Table 25—Schedule-segment attributes (*continued*)

Attribute name	Attribute ID	Attribute type	Remark	Qualifiers
Schedule-Segment-Start-BO-Time	MDC_ATTR_SCHED_SEG_START_BO_TIME	BaseOffsetTime	This attribute defines the time of the first execution of the schedule segment. If this attribute is used, the Schedule-Segment-Start-Abs-Time shall not be used. If this attribute is used, the schedule-segment becomes the active-schedule-segment in the schedule-store at the start time. A schedule segment shall not be active before the start time.	Optional Dynamic
Schedule-Segment-End-BO-Time	MDC_ATTR_SCHED_SEG_END_BO_TIME	BaseOffsetTime	This attribute defines the expiration of the schedule segment. If this attribute is used, the Schedule-Segment-End-Abs-Time shall not be used. If this attribute is used and the end time is reached, the schedule-segment shall no longer be the active-schedule-segment in the schedule-store.	Optional Dynamic
Fixed-Schedule-Segment-Data	MDC_ATTR_SCHED_SEG_FIXED_DATA	N/A The data are stored internal to the device; therefore, this data type never occurs in any protocol definition directly.	This attribute defines the schedule data transferred as an array of entries in a format as specified in the Schedule-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 retrievable by the manager only by using the Trig-Schedule-Segment-Data-Xfer method.	Mandatory Dynamic

(Table continues)

Table 25—Schedule-segment attributes (*continued*)

Attribute name	Attribute ID	Attribute type	Remark	Qualifiers
Schedule-Segment-Confirm-Timeout	MDC_ATTR_SCHED_SEG_CONFIRM_TIMEOUT	RelativeTime	<p>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.</p> <p>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 schedule-store object.</p> <p>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 timeout 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.</p>	Optional Dynamic
Schedule-Segment-Transfer-Timeout	MDC_ATTR_SCHED_SEG_TRANSFER_TIMEOUT	RelativeTime	<p>This timeout attribute defines the minimum time that the manager shall wait for the complete transfer of schedule-segment information.</p> <p>If the timeout expires prior to the reception of the complete schedule segment, the manager shall transition to the Unassociated state as described in 8.9.5.6.</p>	Mandatory Dynamic

The Fixed-Schedule-Segment-Data attribute may hold a sizable amount of data, depending on the agent capabilities and the application. An agent may choose to restrict the maximum size of the Fixed-Schedule-Segment-Data attribute in a way that is aligned with the maximum transmission unit of the transport system.

If the Schedule-Segment-End-Abs-Time or Schedule-Segment-End-BO-Time of an active schedule has occurred, then the agent shall update the value of the Active-Schedule-Segment.

6.3.12.4 Schedule-segment object methods

There are currently no schedule-segment object methods defined in this standard.

6.3.12.5 Schedule-segment object events

There are currently no schedule-segment object events defined in this standard.

6.3.12.6 Other schedule-segment object services

There are currently no uses of the schedule-segment SET or GET services defined in this standard.

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 [B17].

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 IEEE Std 11073-10101.

Vendor-specified classes may be defined. Vendor-specified objects may be created from those classes or any classes defined in this series of standards.

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.

- Bit settings are often used to govern or specify behaviors and/or options. In many cases not all the bits are assigned and/or are indicated as reserved. For future compatibility, managers shall ignore the settings of any reserved or unassigned bits (see A.2.1).

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, PM-store and Schedule-store attributes. The list of MDS object attributes is given in 6.3.2.3, the list of PM-store attributes is given in 6.3.7.3, and the list of Schedule-store attributes is given in 6.3.11.3.
- 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 6.3.9.3.6).
- 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, 6.3.9.5.5, 6.3.9.6.5, and 6.3.11.5.
- 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, 6.3.7.4, and 6.3.11.4.

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 shall 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.

If any object in an agent is accessed by confirmed object access services, there shall be at most one unacknowledged confirmed object access service outstanding from this object at any point in time.

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. If confirmed event report is used by certain object, there shall be at most one unacknowledged confirmed event report outstanding from this object at any point in time.

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 non-MDS objects and attributes that exists in an agent denotes the agent device configuration and is associated with a Dev-Configuration-Id value (see Table 3). 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.

The MDS object is not considered part of the configuration. A manager reassociating with an agent offering the same Dev-Configuration-Id cannot expect the MDS attribute values to be the same; for example, an agent may clear the mds-time-mgr bit as its clock has already been set.

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. Even if the manager knows the agent's device configuration, the manager may ask to enter the Configuring state in order to check attributes from the MDS object before deciding to accept the association. 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 except the MDS, the configuration event report shall contain the static attributes and should contain the dynamic attributes used by the object. These attributes are reported in a list of ConfigObject structures (A.11.5). The Handle attribute value is reported in the obj-handle field of the ConfigObject and is not included in the attribute-list of the ConfigObject. The observational attributes of the objects shall not be included in the ConfigObject. Observational attribute values are sent in later scan event reports (see 7.4.5 and 7.4.6). The agent may add new attributes to an object or change nonstatic attribute values while in the Operating state without sending a new configuration, even if this attribute is not initially present in the standard configuration defined in a certain ISO/IEEE 11073-104zz device specialization.

Changes to any attribute values of metric and scanner objects shall be reported to the manager in scan event reports prior to sending event reports that depend on those values (e.g., scan-handle-attr-val-map and a group format event report or unit-code and the observed value). Changes to any nonstatic attributes values on PM-stores or the MDS may be reported to the manager in event reports at the discretion of the agent. 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.

NOTE—A manager can always poll PM-stores and the MDS object with the GET service; thus, it is not necessary for the agent to send updates or additions to these objects in scan event reports. However, if the agent application deems it is important for the manager to have this information at the time of the change, the agent is free to notify the manager with a scan event report.

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.

Two special device specializations are defined: "hydra" and "generic".

- The "hydra" device specialization, identified by MDC_DEV_SPEC_PROFILE_HYDRA, is used in certain instances to indicate where an agent implements multiple device specializations or a

manager supports any defined specialization(s), but where a protocol is unable to indicate such a list of specializations. It may be used in the following cases:

- “Hydra” may be used by an agent in out-of-band (e.g., transport level) signaling to indicate that the agent implements multiple specializations.
- “Hydra” may be used by a manager in out-of-band (e.g., transport level) signaling to indicate that it supports any defined specialization(s).
- The “generic” device specialization, identified by `MDC_DEV_SPEC_PROFILE_GENERIC`, indicates an agent that adheres to this standard, but for which no specification exists in an ISO/IEEE 11073-104zz device specialization; or a manager that might support an agent with any configuration adhering to this standard. It may be used in the following cases:
 - “Generic” may be used by an agent as its device type as specified in the System-Type or System-Type-Spec-List attribute of the MDS; it may also be used in out-of-band (e.g., transport level) signaling for the same purpose.
 - “Generic” may be used by a manager in out-of-band (e.g., transport level) signaling to indicate support for an agent with any configuration adhering to this standard.
- No (sub-) profiles or standard configurations are defined for the “generic” and “hydra” device specializations.

7.4.3.5 Profiles

7.4.3.5.1 General

A profile further constrains the objects, services, and communication model of a specialization. Some specializations are written to cover a broad category of device types. They define common objects that are useful to the set of devices and other objects that are specific to a limited number of devices. By further profiling the device specialization, the standard provides more guidance on the specific mandatory objects that shall be implemented, the objects that are optional, and the objects that are not needed. The profile can also define smaller application protocol data unit (APDU) sizes, specific standard configurations, and help the manager know exactly which device type has been implemented (e.g., step counter versus exercise cycle).

A profile is expected to be identified by a name and a nomenclature value; for example, within the Activity hub standard (IEEE Std 11073-10471 [B12]), an implementer could declare adherence to a smoke sensor profile or a carbon monoxide sensor profile.

Implementing a smoke sensor or carbon monoxide profile would choose from those objects, services and communication facilities defined within the specialization. A device that claims conformance to these profiles adheres to the requirements stated in the clauses of the profiles for the domain information, service and communication models.

7.4.3.5.2 DIM constraints

Objects that are mandatory or conditional in the specialization remain mandatory or conditional in a profile. Objects that are optional in the specialization may either remain optional or become mandatory within a profile. A profile is not intended to do the following:

- Define additional objects.
- Extend the objects with additional attributes and attribute value ranges.
- Extend the conditions in addition to those that are already defined in the specialization.

7.4.3.5.3 Service model constraints

A profile is not expected to modify or extend the Association service or the object access services. In particular, a profile is not intended to do the following:

- Extend the set of events described by the specialization.
- Extend the set of methods described by the specialization.

A configuration event report pertaining to a particular profile may be distinct from other configuration event reports defined in the device specialization.

7.4.3.5.4 Communication model constraints

A profile may reduce the maximum APDU size compared to the maximum APDU defined in the device specialization.

7.4.3.6 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.

7.4.3.6.1 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 $zz \times 100$ to $zz \times 100 + 99$, inclusive. For example, the 1500–1599 range is reserved for IEEE Std 11073-10415™ [B8]. All unused values in the standard range are reserved for future use. A manager encountering such a reserved value shall assume the value to be an unrecognized standard configuration and process it as described in 8.7.3.3 and 8.8.3.

A manager that supports one (or more) of the ISO/IEEE 11073-104zz device specializations shall be able to accept all the standard device configurations specified for the profiles listed in conformance Table 30 (in 9.4.1) under Gen-4. If there are standard configurations that are generally applicable for the supported specializations, then the manager shall be able to accept all of those as well. Every time an agent requests to associate with that manager using a Dev-Configuration-Id value of a standard configuration, the manager may 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. Alternatively, the manager may request the agent to send the standard configuration in order to enter the Configuring state and check attributes from the MDS object prior to final acceptance (or rejection) of the agent.

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.

7.4.3.6.2 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 in the dev-config-id field 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 Dev-Configuration-Id should be used by an agent for subsequent associations consistently. This has two implications as follows:

- a) The same Dev-Configuration-Id shall not be used by an agent for subsequent associations to identify a different device configuration.
- b) An agent should use the same value for Dev-Configuration-Id in future Association Requests with the manager to denote the same configuration of the device.

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.

This set of mandatory and conditional items guarantees meaningful interaction according to the supported specializations between agents and managers. An agent's implementer shall ensure that it is sufficient for a manager to support the minimum set of mandatory and conditional items to achieve interoperability and to exchange metric data as defined by the specialization.

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 the MDS-Data-Request command to instruct the agent to send measurement data.

In agent-initiated measurement data transmission, event reports are used to carry attribute change sets and/or attribute additions. Attribute change sets and/or additions in these event reports are organized into one or more ObservationScans. A manager applies the changes in the ObservationScan as a unit to the respective objects attaching no semantic behavior to the order in which the attributes appear in these ObservationScans for that object.

NOTE 1—Example 1: If the ObservationScan for a temperature object contains the set of attribute values representing a temperature and a Metric-Id containing a body location, the correct semantic interpretation is that of a single measurement with those respective values. If the ObservationScan is from an RT-SA containing a stream of temperature values and a body location the correct interpretation is that the body location applies to the entire stream of temperature values. If the ObservationScan only contains the set of attribute values representing a unit code and body location (both dynamic attributes and no observation attributes), the correct interpretation is that the new values will apply to the next observation received (assuming these two dynamic values are not further updated).

NOTE 2—Example 2: If an ObservationScan is received containing just attribute A followed by an ObservationScan for a temperature measurement, if attribute A is an observational attribute (for example measurement status) that value could NOT be applied to the temperature measurement. If attribute A is dynamic (for example a unit code) that value would be applied to the temperature measurement. If the attribute A and the temperature measurement were in one ObservationScan, the value of attribute A would apply to the temperature regardless of attribute A's qualifier.

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 optimizes data transmission by defining a specific list of attributes to communicate and the order in which they appear in the message. The Attribute-Value-Map of the object holds this definition including the attribute IDs and the length of the attribute's value. The specific choice of which attributes to include depends on which attribute values are expected to change. For instance, one weighing scale implementation may communicate the weight observation and time stamp, but another may send weight observation, measurement status, and time stamp. In the first case, the attribute-value-map would be set to include the weight observational attribute ID (MDC_ATTR_NU_VAL_OBS_SIMP) and its length (4 bytes long) followed by the time stamp attribute ID (MDC_ATTR_TIME_STAMP_ABS) and its length (8 bytes long). The second case would be similar but also would include the attribute ID for measurement-status (MDC_ATTR_MSMT_STAT) and its length (2 bytes long). The Attribute-Value-Map attribute shall be defined and transmitted to the manager before fixed format event report 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 previously given Attribute-Value-Map to know how to extract the data. For example, in the first case described earlier, the manager knows that the weight observation is the first element in the fixed event report and that it is 4 bytes long so it can extract that into the Simple-Nu-Observed-Value attribute and then it extracts the final 8 bytes into the Absolute-Time-Stamp. The order of these elements is defined by the order in which the attribute IDs are listed in the Attribute-Value-Map. The agent controls the order and communicates it to the manager via the Attribute-Value-Map attribute.

Comparison between different reporting formats

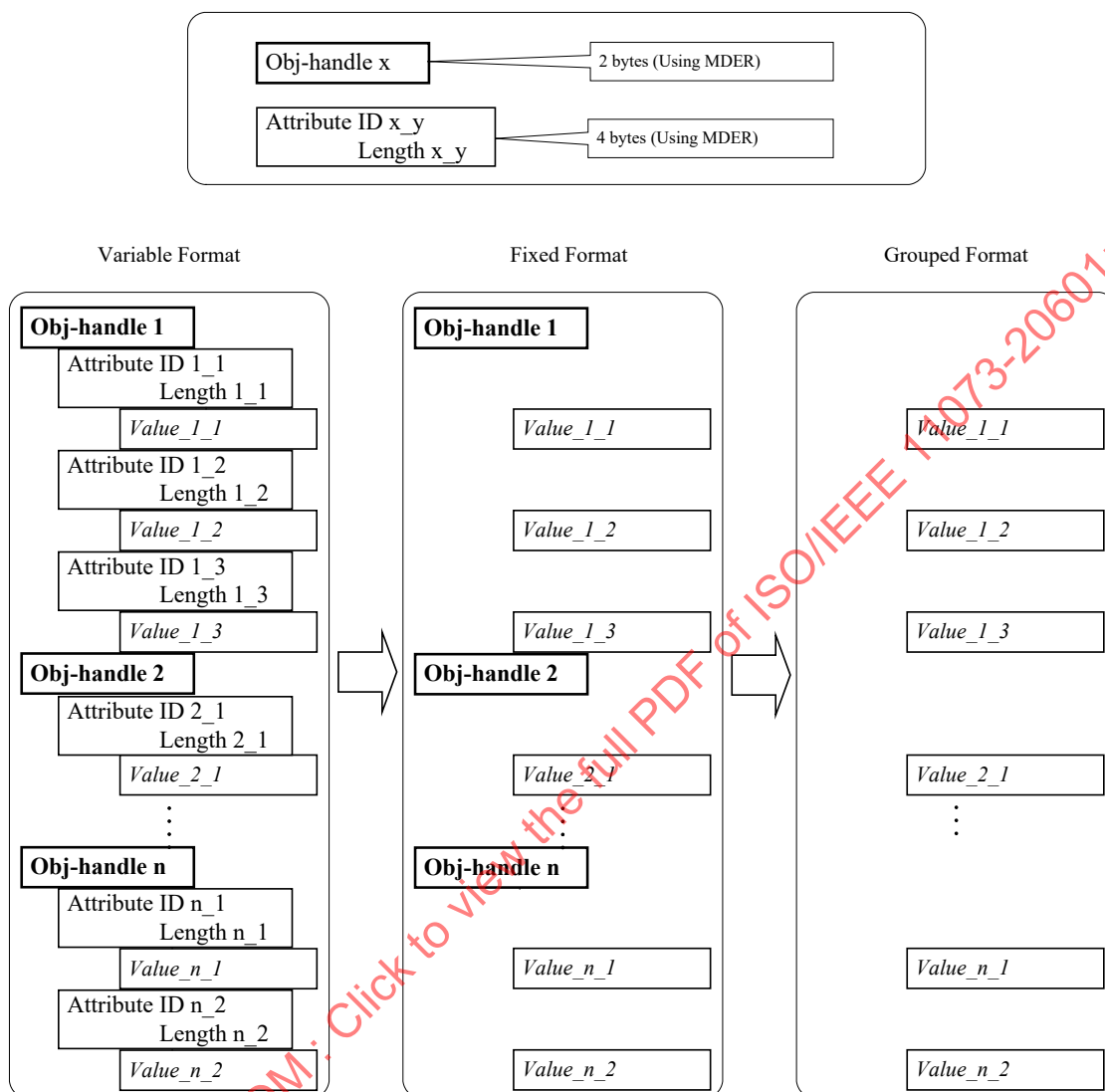


Figure 7— Variable format, fixed format, and grouped format relationships

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 Scan-Handle-Attr-Val-Map. The notion is similar to the Attribute-Value-Map, but the Scan-Handle-Attr-Val-Map allows the agent to report on multiple objects at once by referencing other object handles and the attributes within that object. This attribute shall be defined before grouped event report 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 Scan-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 previously given Scan-Handle-Attr-Val-Map to know how to extract the data.

A manager shall support variable format and fixed format event reports, and if the manager supports scanners, it shall support 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 or by inspecting the Scan-Handle-Attr-Val-Map attribute for scanner objects.

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.

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 following event report mechanisms: manager- or agent-initiated; 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 confirm 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.
- The temporarily stored measurements shall be sent in FIFO order.

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. See Figure 8.

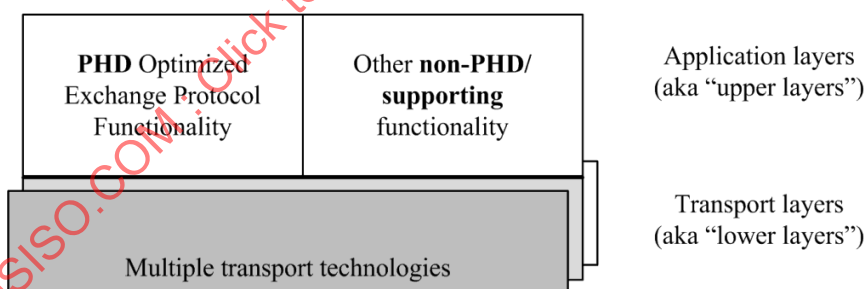


Figure 8—System context

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. See Figure 9.

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.roiv-cmip-confirmed-action, prst.roiv-cmip-confirmed-event-report, prst.roiv-cmip-get, prst.roiv-cmip-confirmed-set
 - prst.rors-cmip-confirmed-action, prst.rors-cmip-confirmed-event-report, prst.rors-cmip-get, prst.rors-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.roiv-cmip-action, prst.roiv-cmip-event-report, prst.roiv-cmip-set

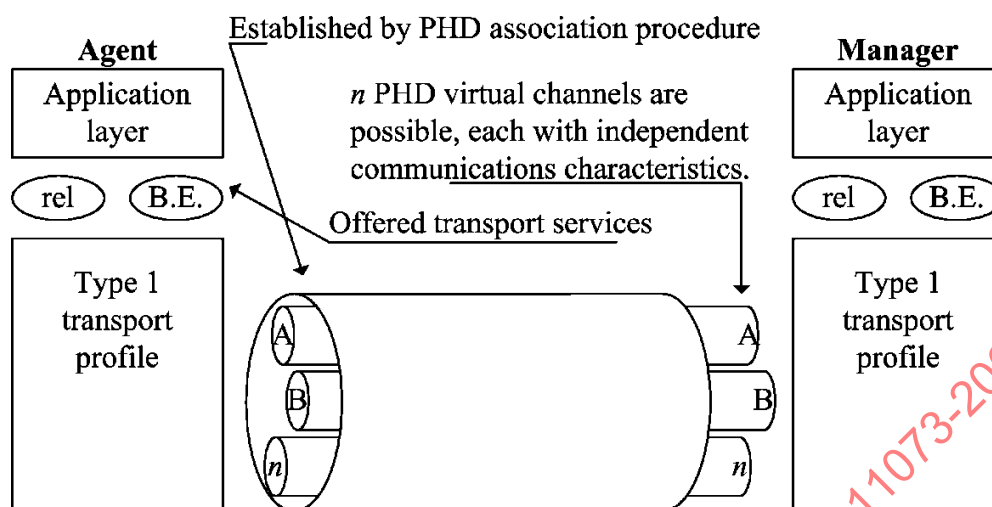


Figure 9—General communications model

In general, the term *metadata* means data about data. In the context of IEEE 11073-20601 communications characteristics, metadata are used to mean supporting information or data relating to the 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 universal serial bus (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, profiles, 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 an APDU that is larger than the manager's receive buffer, it shall reply with an error (roer) code of protocol-violation. The manager's receive buffer shall be at least as large as the largest buffer specified in the specializations the manager supports. The buffer size limitations in this item and the next one apply to all APDUs regardless of whether a standard or extended configuration is being used.
- d) APDUs, in the manager-to-agent direction, shall be no larger than 8K (8192) bytes in size. Specific device specializations, profiles, 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:

- a) APDUs shall be received in the order they are sent.
- b) APDUs shall be free of detectable errors.
- c) APDUs shall not be duplicated.
- d) APDUs shall not be missing.
- e) APDUs are generally sent in an expeditious manner, but may be delayed due to retries.
- f) The communications layers should provide a mechanism to indicate to the application layer when a complete APDU has been received.
- g) 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.
- h) The communications layers should provide a mechanism to indicate to the application layer when a connection is terminated or disconnected.
- i) The communications layers shall provide a mechanism to indicate to the application layer when it is unable to send an APDU.
- j) 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:

- 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.
- An APDU may be lost or duplicated.
- 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 E.2. Table 26 provides a description of each of the states. Table 27 provides a description of each of the state transitions.

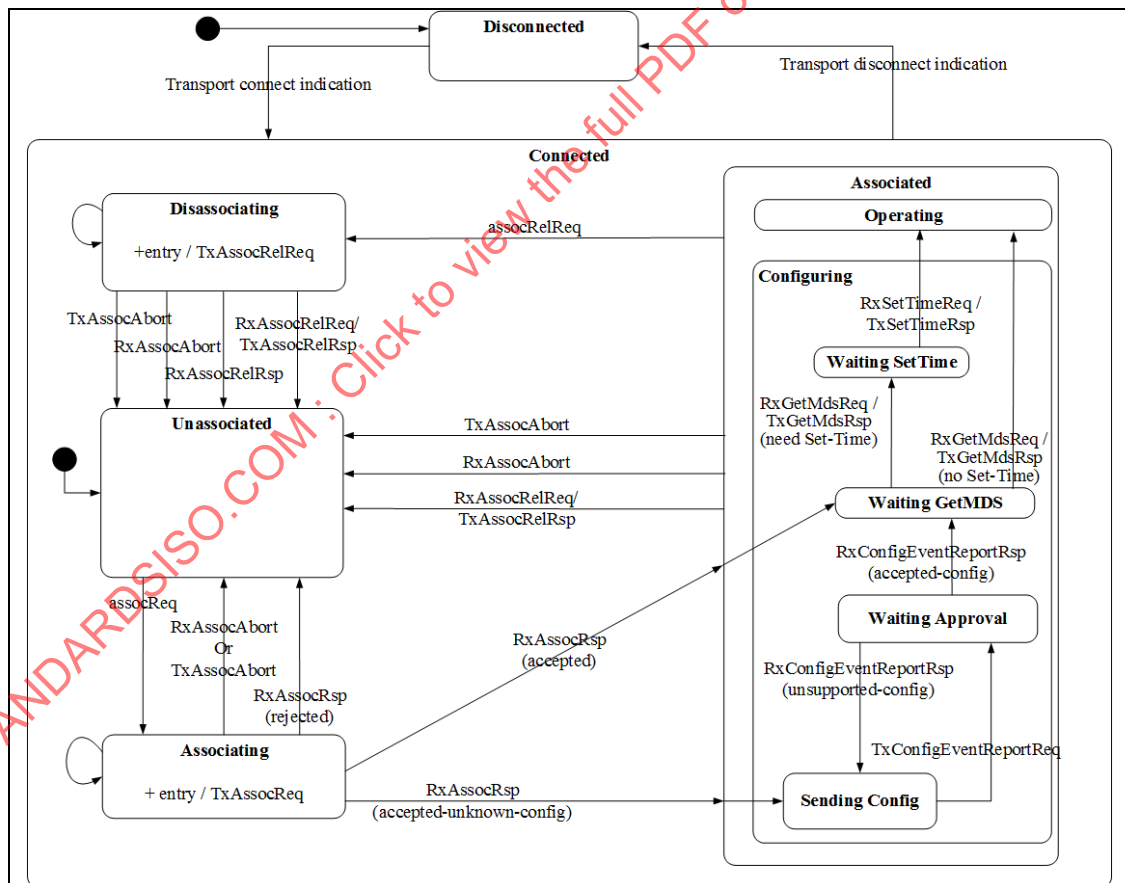


Figure 10—Agent state machine diagram

Table 26—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.
Unassociated	<p>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:</p> <ul style="list-style-type: none"> — 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. <p>The agent remains in the Unassociated state until the agent determines that it should begin associating with the manager.</p>
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	See 8.9 for a description of possible procedures while in the Operating state.
Configuring	<p>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 8.8).</p> <p>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 Waiting GetMDS substate.</p> <p>Within TO_{get} (timeout: get service) seconds after the agent enters the Waiting GetMDS substate, the manager shall invoke a GET service, and the agent sends the GET response to the manager. If the mds-time-mgr-set-time bit is not set, the agent moves directly into Operating state. If the mds-time-mgr-set-time bit is set, the agent shall wait for TO_{ca} (timeout: confirmed action service) seconds to receive the Set-Time action command from manager. Upon receiving the Set-Time action, the agent shall clear the mds-time-mgr-set-time bit before sending the Set-Time confirmation. After sending the Set-Time confirmation, the agent moves into Operating state.</p> <p>If the agent uses base offset time to report, the Set-Base-Offset-Time action shall be used instead of Set-Time action in the above description.</p> <p>In cases of a timeout, the agent sends an abort request and moves to the Unassociated state.</p>
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 27 —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.
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.
RxGetMdsReq/TxGetMdsRsp	Within TO_{get} seconds after an agent enters the Configuring/Waiting GetMDS substate, the manager shall invoke a GET service request, and the agent sends the GET response with all implemented MDS object attributes. After that, the agent moves into either the Operating state or Configuring/Waiting SetTime substate, depending on the value of mds-time-mgr-set-time bit. If the mds-time-mgr-set-time bit is set, the agent moves into Configuring/Waiting SetTime substate; otherwise, it moves into Operating state.
RxSetTimeReq/TxSetTimeRsp	Within TO_{ca} seconds after the agent enters the Waiting SetTime substate, the manager shall send the Set-Time (or Set-Base-Offset-Time) action command to the agent. Once its internal time setting operation is completed, the agent responds to the manager and moves into Operating state.
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 26 and Table 27. The key differences are as follows:

- The manager shall wait in the Waiting for Config state for at least TO_{config} (timeout: configuration procedure) seconds before sending an 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 manager shall invoke a GET service request within TO_{get} seconds after an agent enters the Configuring/Waiting GetMDS substate.
- The manager shall send the Set-Time (or Set-Base-Offset-Time) action command to the agent within TO_{ca} seconds after an agent enters the Configuring/Waiting SetTime substate.

The detailed manager state table is described in E.4.

STANDARDSISO.COM : Click to view the full PDF of ISO/IEEE 11073-20601:2022



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 28.

Table 28 —Timeout variables

	Communications service	Timeout		Subclause
		Variable	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}	MDS.Transport-Timeout. If the attribute is not present, the agent and manager shall use the value 3 s.	8.10.5
Operating procedure				
MDS object	Confirm action	TO _{ca}	MDS.Transport-Timeout. If the attribute is not present, the agent and manager shall use the value 3 s.	8.9.5.2
	Confirm event report	TO _{cer-mds}	MDS.Confirm-Timeout. If the attribute is not present, the agent and manager shall use the value 3 s.	8.9.5.3
	Get	TO _{get}	MDS.Transport-Timeout. If the attribute is not present, the agent and manager shall use the value 3 s.	8.9.5.4
	Confirm set	TO _{cs}	MDS.Transport-Timeout. If the attribute is not present, the agent and manager shall use the value 3 s.	8.9.5.5
	<inter-service timeout>	TO _{sp-mds}	MDS.Transport-Timeout. If the attribute is not present, the agent and manager shall use the value 3 s.	8.9.5.6
PM-store object	Confirm action	TO _{ca}	MDS.Transport-Timeout. If the attribute is not present, the agent and manager shall use the value 3 s.	8.9.5.2
	Confirm event report	TO _{cer-pms}	Segm.Confirm-Timeout. If the attribute is not present, the agent and manager shall use the value 3 s.	8.9.5.3
	Get	TO _{get}	MDS.Transport-Timeout. If the attribute is not present, the agent and manager shall use the value 3 s.	8.9.5.4
	Confirm set	TO _{cs}	MDS.Transport-Timeout. If the attribute is not present, the agent and manager shall use the value 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
Schedule-store object	Confirm action	TO _{ca}	MDS.Transport-Timeout. If the attribute is not present, the agent and manager shall use the value 3 s.	8.9.5.2
	Confirm event report	TO _{cer-sched}	SchedSegm.Confirm-Timeout. If the attribute is not present, the agent and manager shall use the value 3 s.	8.9.5.3
	Get	TO _{get}	MDS.Transport-Timeout. If the attribute is not present, the agent and manager shall use the value 3 s.	8.9.5.4
	Confirm set	TO _{cs}	MDS.Transport-Timeout. If the attribute is not present, the agent and manager shall use the value 3 s.	8.9.5.5
	<end of Segm timeout>	TO _{sp-sched}	SchedSegm.Transfer-Timeout	8.9.5.6
Scanner object	Confirm set	TO _{cs}	MDS.Transport-Timeout. If the attribute is not present, the agent and manager shall use the value 3 s.	8.9.5.5
	Confirm event report	TO _{cer-scan}	Scan.Confirm-Timeout. If the attribute is not present, the agent and manager shall use the value 3 s.	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.

Table 29 lists the valid combinations of protocol version and nomenclature version. In the association procedure, an agent indicating support to a specific protocol version shall indicate support to the corresponding nomenclature version as well. In the association procedure, a manager selecting a specific protocol version shall select the corresponding nomenclature version.

To indicate support for multiple protocol versions, the bit values are combined. For example, if the agent supports protocol-version2, protocol-version3, and protocol-version4, it shall use protocol version bits 0x70000000 and nomenclature-version bits 0xE0000000.

Further valid combinations from future versions of this specification may be used by implementations that comply with those future versions.

Table 29 Valid combinations of protocol and nomenclature version

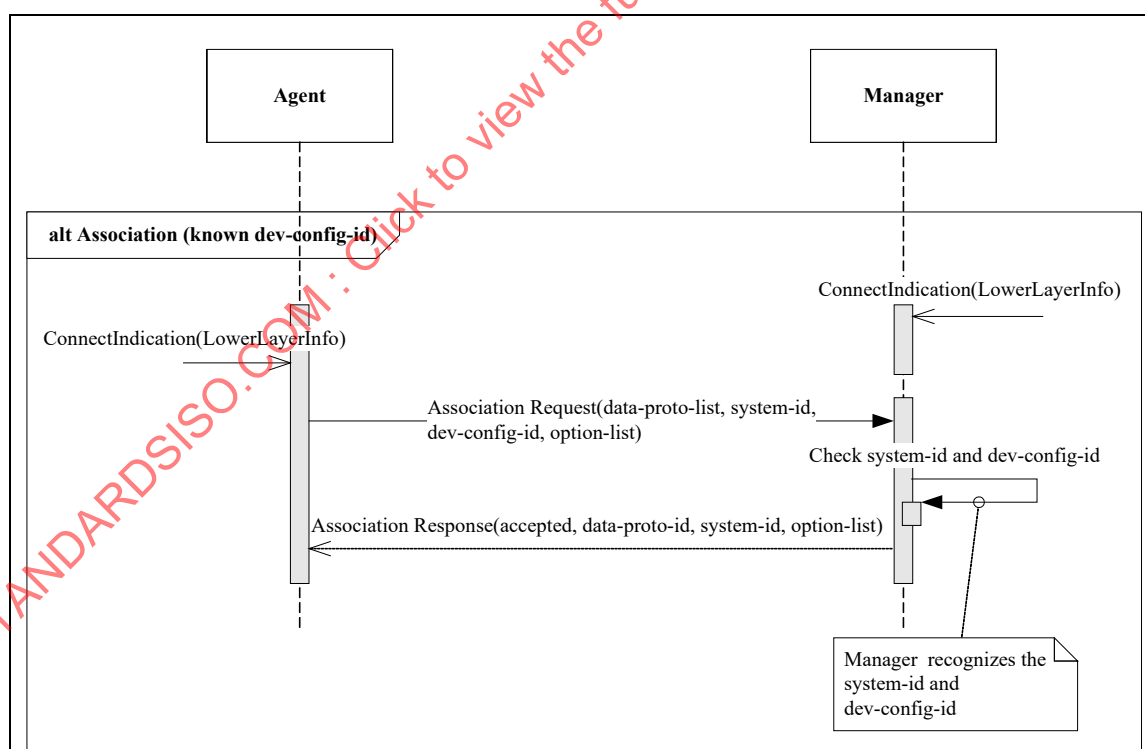
Protocol version	Bit value	Corresponding nomenclature version	Bit value
1	0x80000000	1	0x80000000
2	0x40000000	1	0x80000000
3	0x20000000	2	0x40000000
4	0x10000000	3	0x20000000

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, or the case where the manager knows the agent's configuration, but manager wants to enter Configuring state in order to check attributes from the MDS object.

**Figure 12—Association procedure (known configuration)**

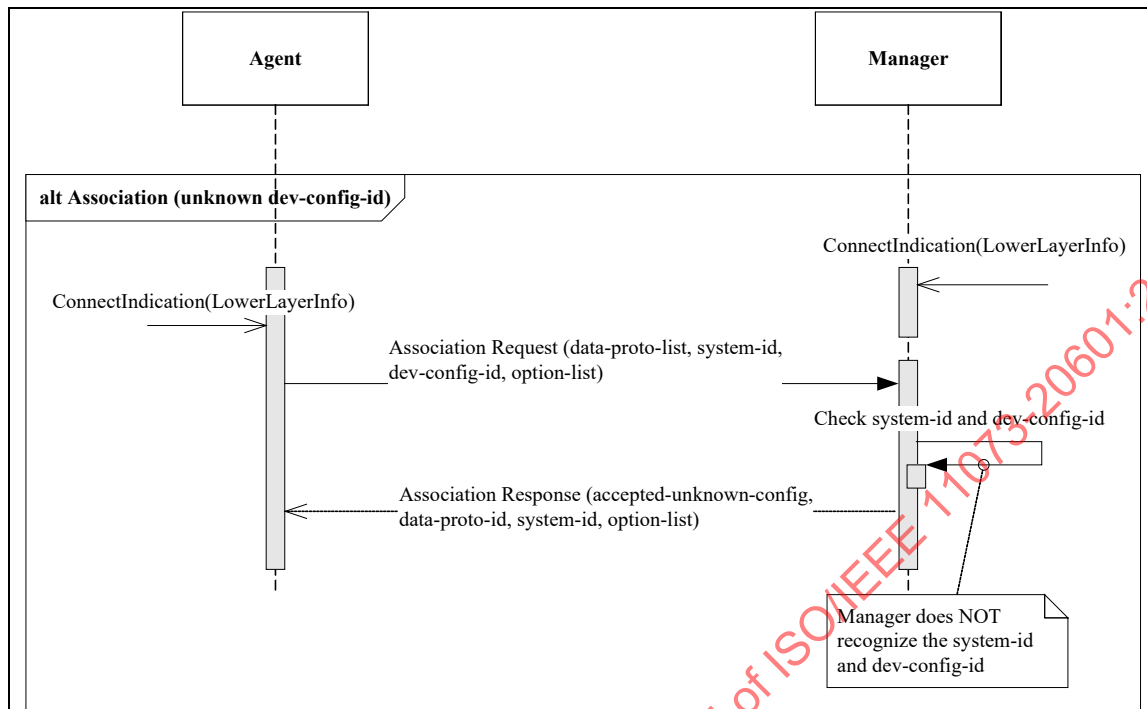


Figure 13—Association procedure (unknown configuration)

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 AarqAdu 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 confirm 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. The agent shall place at most one data-proto element containing the field data-proto-id set to data-proto-id-20601 in the data-proto-list.

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 protocol-version field contains the versions of the data-exchange protocol the agent can support.
- The encoding-rules field contains the specific DataAdu 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 nomenclature-version field contains the version of the nomenclature used. The protocol version bits and nomenclature version bits shall consist of valid combinations of bits as defined in Table 29.
- The functional-units field indicates all functional units and optional features supported by the agent.
- The system-type field indicates the system type (agent in this case).
- The system-id field reports the unique System-Id attribute value (see Table 3) 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.
- The dev-config-id field identifies the configuration being offered for initial consideration during this association as described in 7.4.3. For standard configurations, the value placed in the dev-config-id field shall lie between standard-config-start and standard-config-end, inclusive. For extended configurations, the value placed in the dev-config-id field shall lie between extended-config-start and extended-config-end, inclusive.
- The data-req-mode-capab field defines the data request modes supported by the agent (see 8.9.3.3.3).
- The option-list field contains a list of additional attributes the agent wishes to communicate.

8.7.3.1.3 Data-exchange protocol—defined by the manufacturer

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 universally unique identifier (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 (09/2004).

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:

- The result field representing the outcome of the association procedure.
- The protocol-version field contains the version of the common data protocol chosen by the manager if the result field is equal to accepted or accepted-unknown-config.
- This standard is not fully compatible with IEEE Std 11073-20601-2008, IEEE Std 11073-20601a-2010, and IEEE Std 11073-20601-2014. Any manager that wishes to communicate with an agent based on protocol-version1, protocol-version2, or protocol-version3 shall set the corresponding bit in Association Response.
- The encoding-rules field contains 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 nomenclature-version field contains the version of the nomenclature chosen by the manager if the result field is equal to accepted or accepted-unknown-config. The manager shall select a valid combination of protocol version and nomenclature version as defined in Table 29.
- The functional-units field indicates the common functional units and optional features chosen by the manager if the result field is equal to accepted or accepted-unknown-config.
- The system-type field contains the system type (manager in this case since the message originated from the manager).
- The system-id field has 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 field shall be zero in the response.

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 Waiting GetMDS 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 8.8 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.

A manager may reject the association for any of the possible rejection reasons 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 value in the dev-config-id field as representing a known standard device specialization or as a configuration from a previous association, the manager shall send an Association Response message with a result field of accepted and transition to the Sending GetMDS state or may send an Association Response message with a result field of accepted-unknown-config to force the agent to enter Configuring state in order to check attributes from the MDS object prior to final acceptance of the association.

If the manager does not recognize the value in the dev-config-id field, 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 with a new TO_{assoc} period. This process shall be repeated until an Association Response is received, or RC_{assoc} (retry count: association procedure) attempts have been made after the first timeout, whichever comes first. This results in a maximum of $RC_{assoc} + 1$ Association Requests. 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 reduce the risk that test data are 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.
- Try to confirm 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.
- Try to confirm 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.
- Try to confirm 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 functional-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/Sending Config substate and send its configuration to the manager. The manager enters the Configuring/Waiting for Config substate immediately after it sends the Association Response with the accepted-unknown-config result.

An Association Response message with an accepted result field shall trigger the agent to enter the Configuring/Waiting GetMDS substate. The manager enters the Configuring/Sending GetMDS substate immediately after it sends the Association Response with the accepted 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).

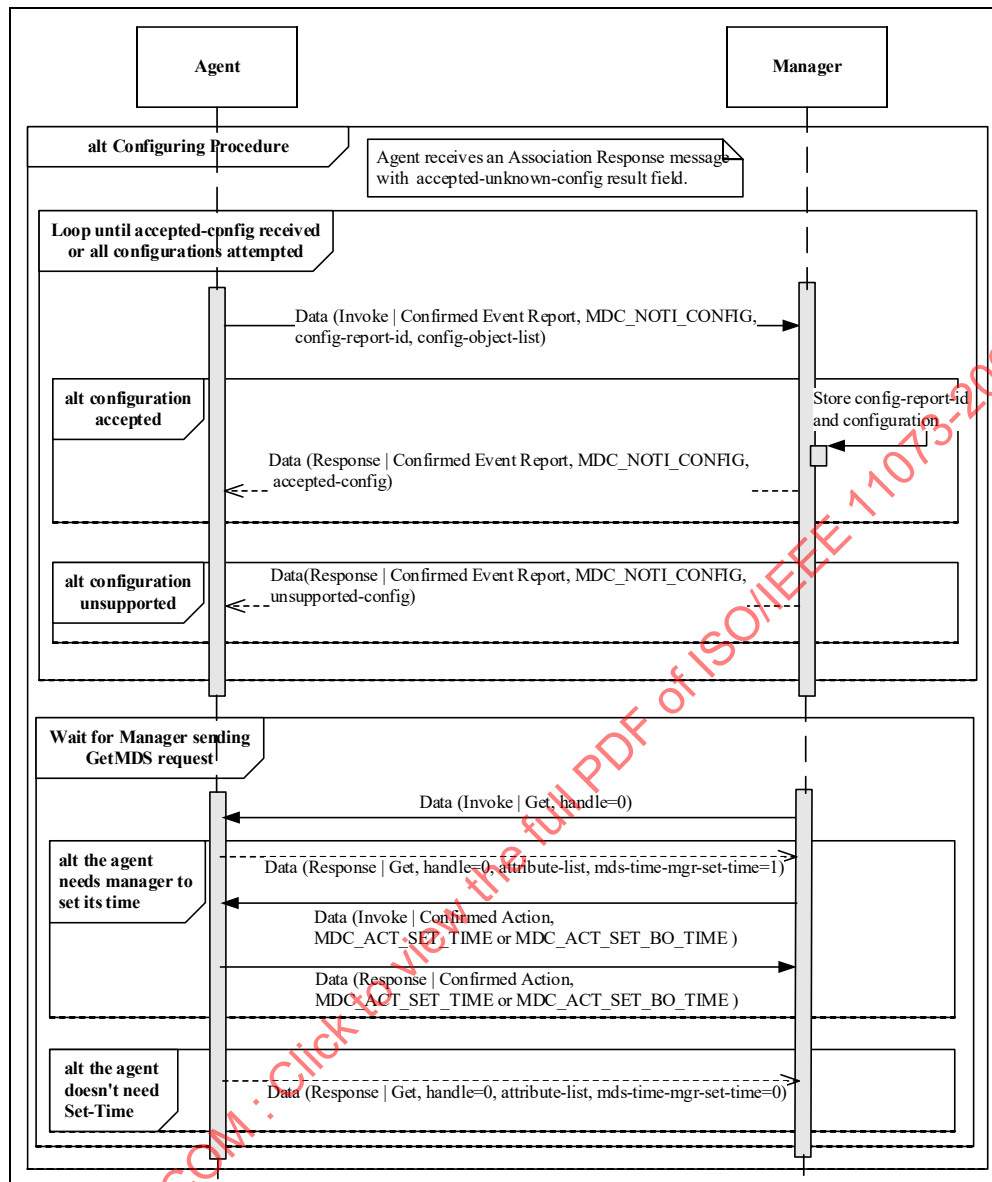


Figure 14—Configuration procedure

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

- The Dev-Configuration-Id attribute of the MDS object that corresponds to the configuration being described,
- 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 prior to sending a fixed format event report for the object. In the case of scanner objects that use the grouped format event reports, the agent shall send the Scan-Handle-Attr-Val-Map describing the layout prior to sending grouped format event reports from that scanner. Typically, the value maps are reported in the configuration event report.

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.

The agent's configuration is identified by the Dev-Configuration-Id attribute of its MDS object, and the agent passes this value to the manager in the dev-config-id field in the Association Request message or the config-report-id field of the configuration event report message.

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 the manager moves to the Sending GetMDS state, and the agent moves to the Waiting GetMDS 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 and/or profiles 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. However, if the extended configuration of the specialization implemented by an agent is one that makes use of PM stores and only reports measurements using segment data events and if the standard configuration does not support PM stores, an agent is not required to support the corresponding standard configuration. If an agent implements multiple ISO/IEEE 11073-104zz specializations, its System-Type-Spec-List attribute contains a list of type/version pairs, each referencing the respective device specialization and version of that specialization.

If the agent conforms to a standard configuration, it shall set the value of Dev-Configuration-Id as defined in the specific ISO/IEEE 11073-104zz device specialization. These standard configuration Dev-Configuration-Id values are assigned in the range between standard-config-start and standard-config-end, inclusive.

When an agent submits a configuration event report 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 in the config-report-id field and an empty config-obj-list. 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 configuration, but on the retry it shall send the full configuration information instead of an empty config-obj-list.

NOTE—If the manager is able to interoperate with the provided standard configuration, it may accept that configuration. If the manager stores configurations, it may store this configuration for future reference when *any* agent uses this standard configuration identification and henceforth may treat the configuration as recognized.

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

An agent should use the same value for Dev-Configuration-Id in future Association Requests consistently. This has two implications. The same Dev-Configuration-Id shall not be used by an agent for subsequent associations to identify a different device configuration. An agent should use the same value for Dev-Configuration-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.

An agent may send an extended configuration with an empty config-object-list. This can happen, for instance, when an agent accepts plug-in components, but presently does not have any inserted. The manager responds with either accepted-config or unsupported-config.

Within a TO_{get} period after the manager has accepted the configuration sent by the agent, the manager shall 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 or a specific list of attributes that contains at least the Mds-Time-Info. See 8.9.3.2 for more details.

If the mds-time-mgr-set-time bit is not set, the agent moves directly into Operating state. If the mds-time-mgr-set-time bit is set, the agent shall wait for TO_{ca} period to receive the Set-Time action command from manager. Upon receiving the Set-Time action, the agent shall clear the mds-time-mgr-set-time bit before sending the Set-Time confirmation. After sending the Set-Time confirmation, the agent moves into Operating state. See 8.12.2.1 for more details.

If the agent uses base offset time to report, the Set-Base-Offset-Time action shall be used instead of Set-Time action in the above process.

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 Sending GetMDS 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 Waiting GetMDS 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 transition to

the Disassociating state and send an Association Release Request message with a reason of no-more-configurations.

In cases of a timeout, the agent or manager shall send an abort request and move to 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} 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 Sending GetMDS state, the manager may request the MDS object attributes of an agent by sending a data message with the “Remote Operation Invoke | Get” command. The agent shall report its implemented MDS object attributes to the manager using a data message with the “Remote Operation Response | Get” response. If the handle value of the “Remote Operation Invoke | Get” command is 0, all the implemented MDS object attributes in the agent shall be reported; otherwise, the agent only reports requested implemented attributes. 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) and shall support retrieval of a specific list of attributes. The handle is entered in the obj-handle field (A.10.4), and it is not included in the attribute ID list of the request or in the attribute list of the response. If the manager requests specific MDS object attributes, indicated by the elements in attribute-id-list, then the agent shall respond

with a `rs-cmip-get` message in which the attribute-list contains a list of the requested attributes of the MDS object that are implemented.

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

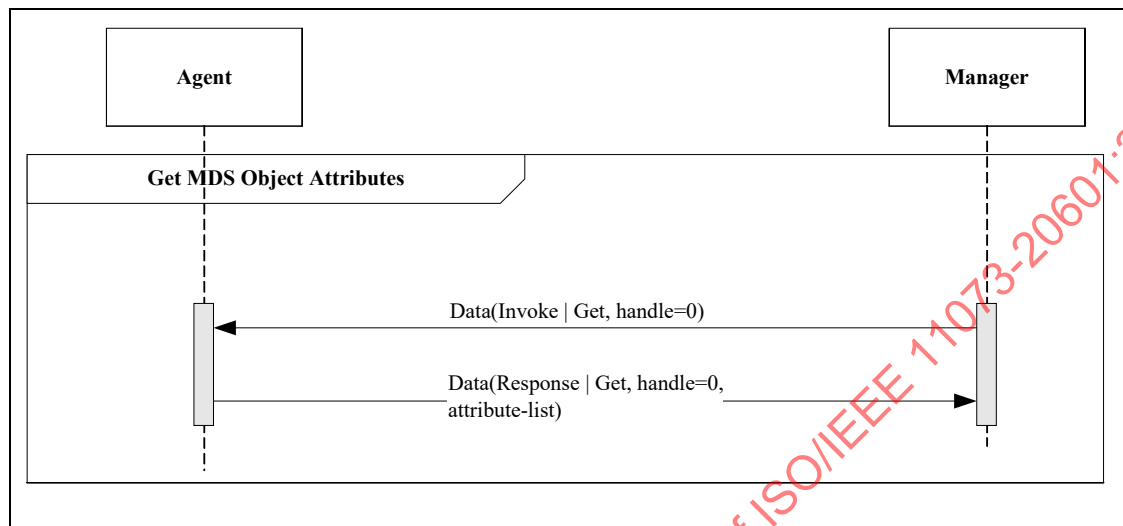


Figure 15—Get MDS object attributes sequence diagram

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 where an external event triggers a measurement to be taken (e.g., a person stands on the weigh scale) and the measurement is transferred to the manager. Agents with large amounts of data, frequent data transfer, or streaming data should use the scanner for transfer. Use of the scanner is preferred in this case as this approach provides a mechanism to control data flow.

Agents with sensors that have no explicit external event to initiate a measurement to be taken can use manager-initiated transfer to request that a measurement is made if practical and that measurement is transferred. Note that receipt of a request to transfer a measurement does not require that an agent perform a measurement. If no measurement is performed, the agent shall transfer the most recent measurement data, setting (`mss-cat-setting`), or calculated value (`mss-cat-calculation`) that is available. An agent that has transferred its most recent data shall not resend the same data in response to further manager-initiated data requests; if no new data are available, then the agent shall send an empty entry as response.

All variants of the two styles are described in detail in 8.9.3.3.2 through 8.9.3.3.4.

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 via the MDS object, the data-req-id field in the Scan Report (MDC_NOTI_SCAN_REPORT_*) shall be set to data-req-id-agent-initiated-confirmed or data-req-id-agent-initiated-unconfirmed depending on whether the event report is confirmed or unconfirmed.

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.

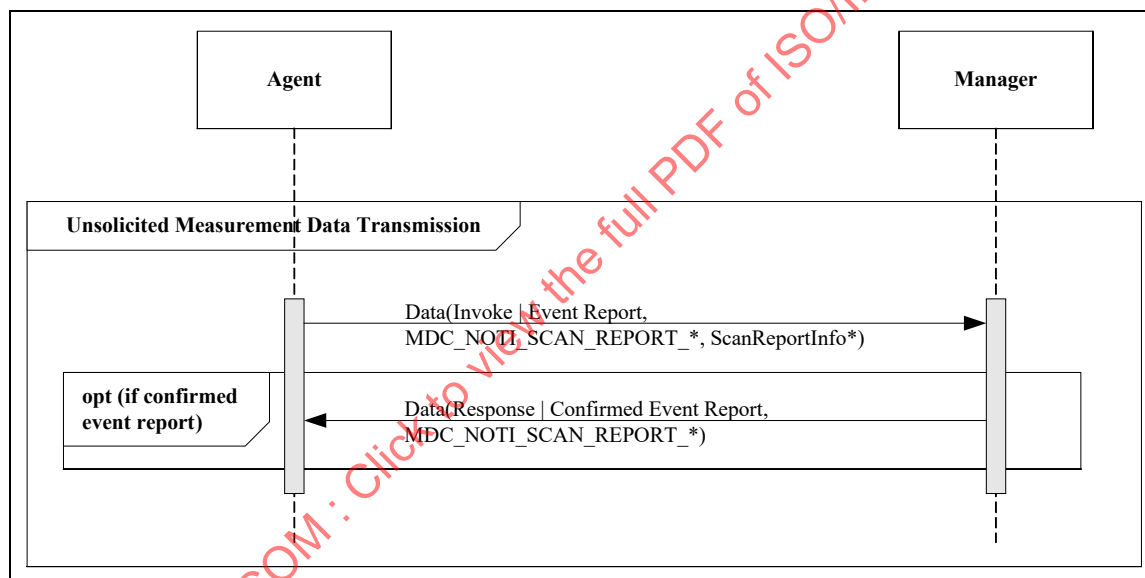


Figure 16—Agent-initiated measurement data transmission

8.9.3.3.3 Manager-initiated data request

In a manager-initiated measurement data request, the manager uses the ACTION service (see 7.3) to request measurement data transmission from the agent (see Figure 17). 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. Where possible, the agent should acquire data to fulfill the response; but where impractical, there is no requirement that the agent collect any data (e.g., inflate a blood pressure cuff). If an agent cannot provide immediate data, it may send the most recent observation or setting. 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*). If the agent does not have data available, it shall return an empty list of data. The capability to provide immediate data in response to a data request shall be indicated by setting the mss-msmt-manager-initiated-immediate bit of MetricSpecSmall.

Each outstanding manager-initiated data request shall be given a unique value for the data-req-id field that shall be between data-req-id-manager-initiated-min and data-req-id-manager-initiated-max. This value of data-req-id shall be returned in the data-req-id field of the DataResponse.

When the data request 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. To reduce the burden on the agent, it is highly recommended for the manager to minimize the list to just one handle per request.

A manager-initiated request shall be rejected if agent-initiated data transmission is pending, and a response of data-req-result-rejected shall be reported in the DataReqResult.

A manager-initiated data request response with dynamic attribute values shall update those dynamic attribute values in the manager.

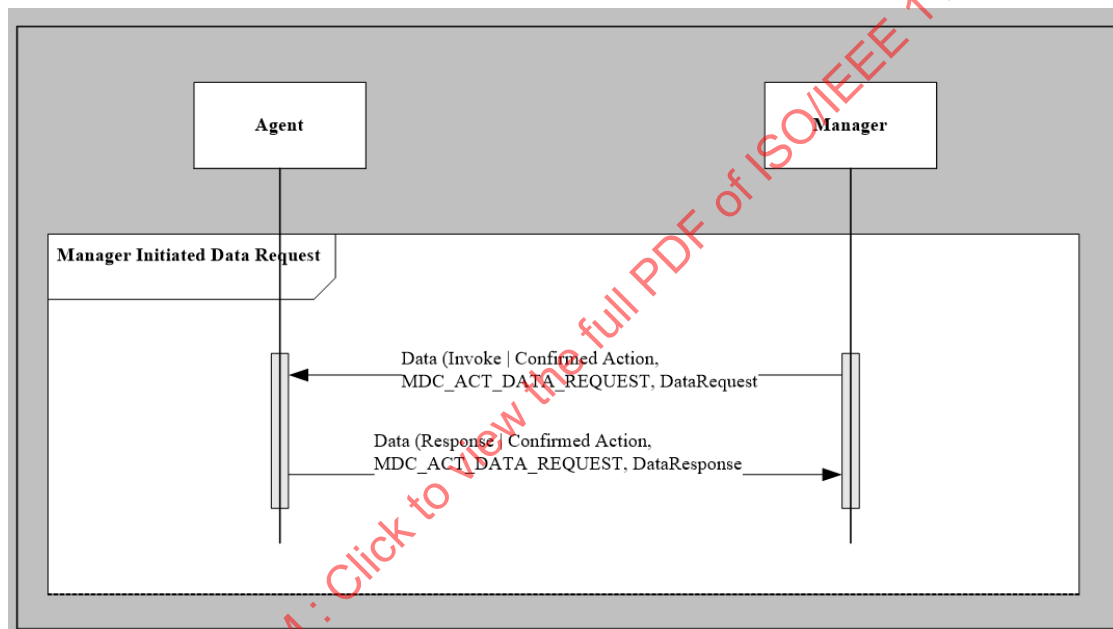


Figure 17 —Manager-initiated measurement data request

8.9.3.3.4 Scan report number management

The manager-initiated data request form of measurement transmission shall result in a response that has a 0 in the scan-report-no field of the ScanReportInfo* structure.

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. Note that there will be a separate scan-report-no for confirmed (data-req-id 0xF000) and unconfirmed (data-req-id 0xF001) scan event reports.

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 18 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 handle value is placed in the obj-handle field of the message (A.10.4) and is not present in the attribute-id-list of the request or the attribute-list of the response. The attribute-id-list shall be left empty to query for all attributes of the PM-store object. Alternatively, specific attributes of an object may be queried by listing the desired Attribute IDs found in Table 10. It is not required for an agent to support this capability. If this capability is not implemented, then the agent shall respond with an error (roer) message with an error-value of not-allowed-by-object.

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 10 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.

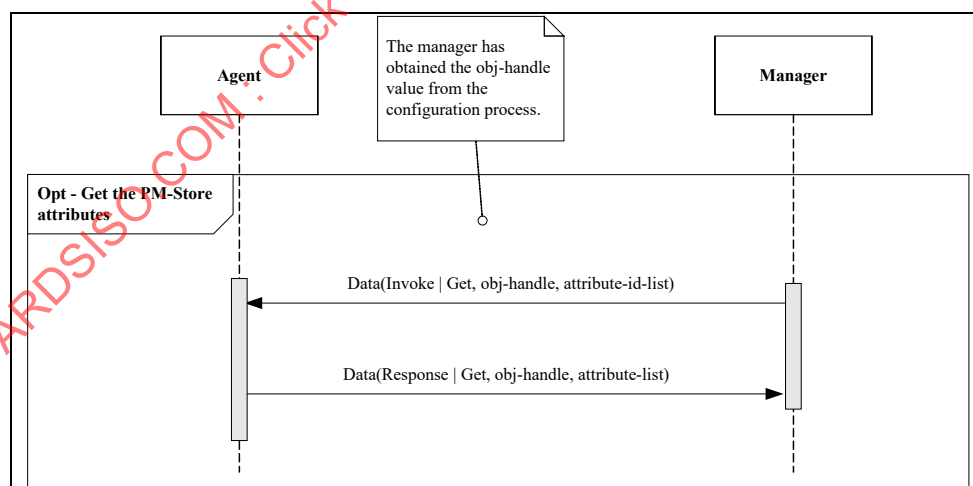


Figure 18—Retrieving a PM-store's attributes

- b) **Retrieving the ID list of PM-segments.** The manager retrieves information on the segments in a PM-store by sending an ACTION.Get-Segment-Id-List command to the specific PM-store (see Figure 19). The agent responds to the ACTION.Get-Segment-Id-List command with a list of the instance numbers.

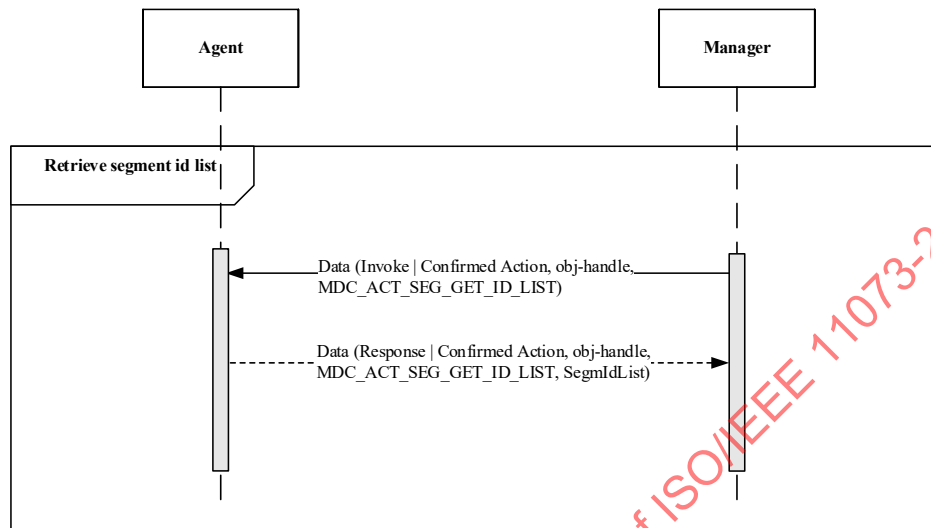


Figure 19—Retrieving PM-segment information using Get-Segment-Id-List

- c) **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 20) with a request to return information from all segments, a particular list of segments, or any segments within a given time range. 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. If there is no segment in any of these three cases, the agent responds with an empty list. The agent shall support the first selection criteria and may provide support for the second and third 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.

If the manager invokes the Get-Segment-Info method but the agent does not support the specified choice (list of segments or range of segments by time), then the agent shall respond with a roer DataAdu with an RoerErrorValue of “unsupported-choice”.

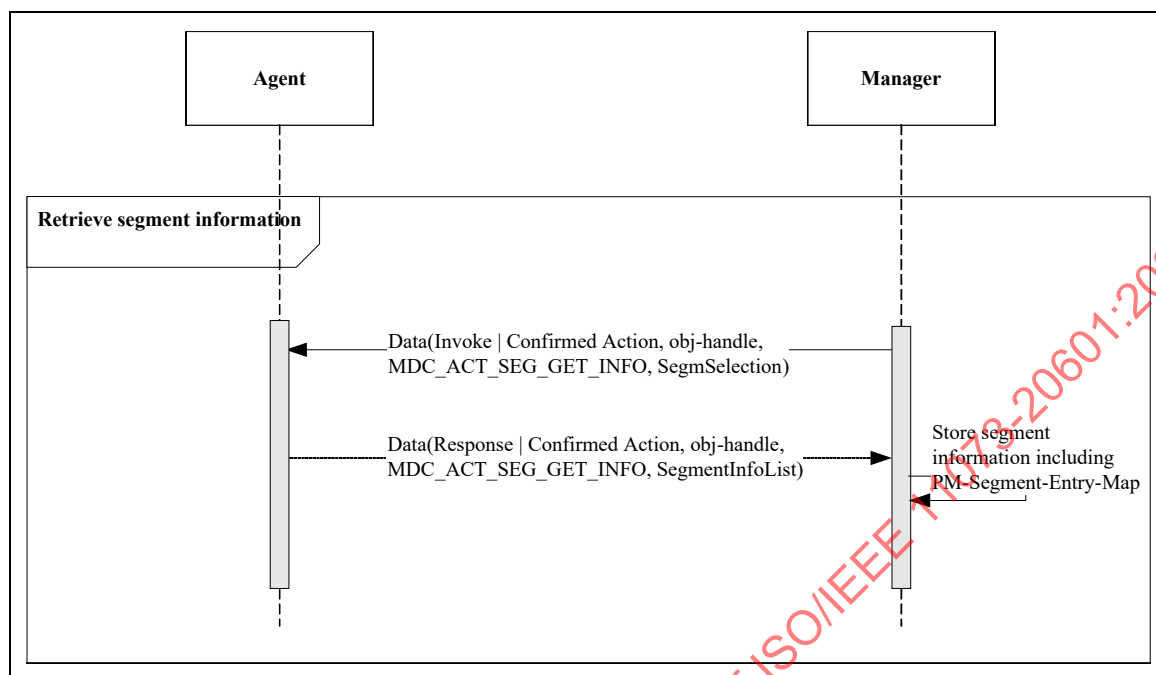


Figure 20—Retrieving PM-segment information using Get-Segment-Info

- d) **Transferring PM-segment content.** The manager retrieves specific PM-segments by using the Trig-Segment-Data-Xfer ACTION method to initiate the data transfer (see Figure 21). In the first step, the manager sends the ACTION method to the agent with the handle of the PM-store object to access. The argument to this ACTION method is the instance number of the segment to transfer.

The agent shall decide whether the request can be honored. It checks for a valid segment number, available segment data (e.g., the segment 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-Segment-Data-Xfer ACTION invoke message at any time. However, if the manager does send a Trig-Segment-Data-Xfer ACTION invoke message while a Clear-Segments ACTION invoke message is outstanding, the agent may generate a Trig-Segment-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 shall transfer all segment entries in FIFO order. 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.

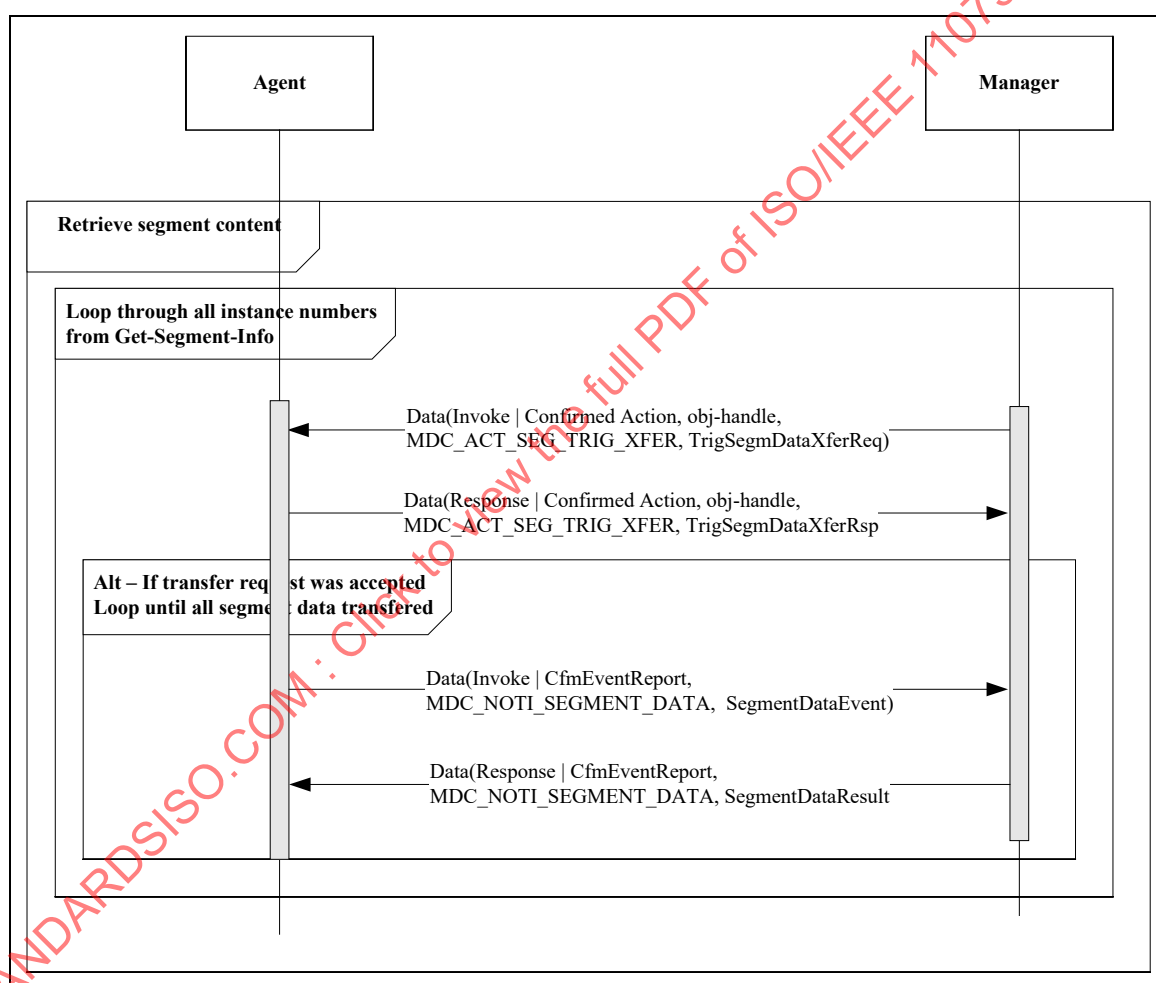


Figure 21—Retrieving PM-segment content

- e) **Clear a PM-segment.** The agent may support PM-segment clearing. The manager determines whether the agent supports any of the clearing functions by inspecting the `pmsec-clear-segm-all-sup`, `pmsec-clear-segm-by-list-sup`, and `pmsec-clear-segm-by-time-sup` flags in the `PM-Store-Capab` attribute.

The manager may clear a PM-segment at any time and uses the sequence shown in Figure 22. 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. If the agent supports this function, then it shall support clearing all segments (`pmsec-clear-segm-all-sup`), may support clearing a particular list of segments (`pmsec-clear-segm-by-list-sup`), and may support the time range selection criteria (`pmsec-clear-segm-by-time-sup`). The manager determines which capabilities are supported by inspecting the individual `PM-Store-Capab` attribute bits.

If the manager invokes the `Clear-Segments` method but the agent does not support this function at all (no `pmsec-clear-segm-*` flags in the `PM-Store-Capab` attribute being set), then the agent shall respond with an `roer DataAdu` with an `RoerErrorValue` of “no-such-action”. If the manager invokes the `Clear-Segments` method but the agent does not support the particular action (list of segments or range of segments), then the agent shall respond with a `roer DataAdu` with an `RoerErrorValue` of “not-allowed-by-object”.

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 `pmsec-clear-segm-remove` bit of the `PM-Store-Capab` attribute.

As described in 6.3.7.4, the `Clear-Segment` method may not clear all of the selected PM-segments. For verification purposes, a manager may issue `Get-Segment-Info` or `Get-Segment-Id-List` actions and `GET` requests to monitor the actual clearing and/or removal of any segments.

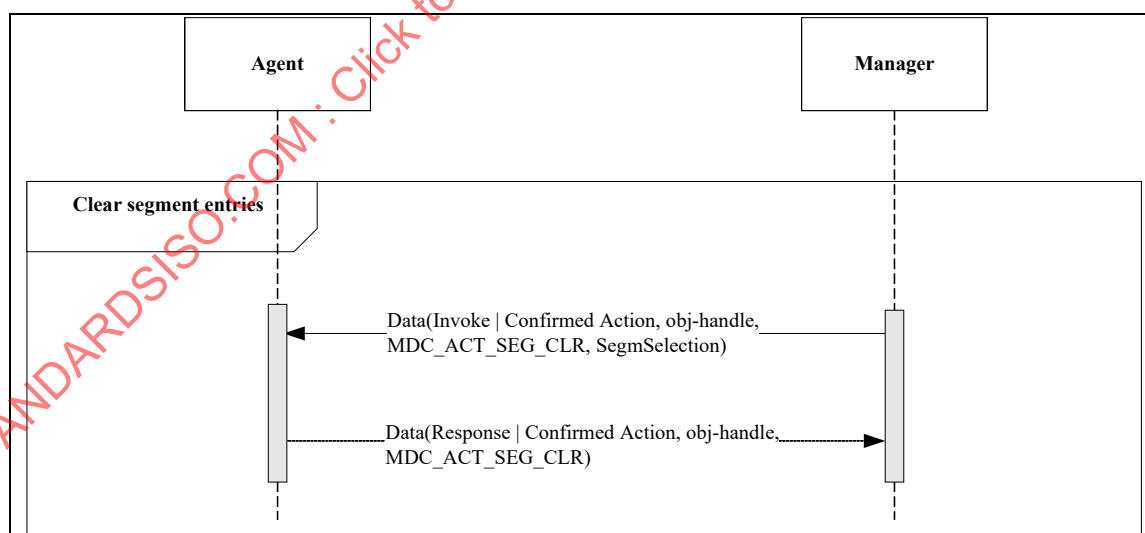


Figure 22—Clearing segment entries

8.9.3.5 Schedule-stored information retrieval

8.9.3.5.1 General

When an agent implements one or more schedule-store objects, the agent reports about the existence of the schedule-store object during the configuration phase. The manager uses this information to query the schedule-store object(s) of the agent. The interactions between the manager and agent when retrieving the information in the schedule-store(s) is described in 8.9.3.5.2.

8.9.3.5.2 Schedule-stored information retrieval

- a) **Retrieving the schedule-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 schedule-store objects in the agent. The manager may query each schedule-store to determine the number of schedule-segments that exist within the schedule-store. Figure 23 shows the sequence diagram of this operation. The manager sends a Get command to the agent requesting attribute information from a particular schedule-store. The manager uses the handle number to reference the desired schedule-store. The handle value is placed in the obj-handle field of the message (A.10.4) and is not present in the attribute-id list of the request or the attribute-list of the response. The attribute-id-list shall be left empty to query for all attributes of the schedule-store object. Alternatively, specific attributes of an object may be queried by listing the desired attribute IDs found in Table 22. It is not required for an agent to support this capability. If this capability is not implemented, then the agent shall respond with an error (roer) message with an error-value of not-allowed-by-object.

The manager can inspect the attributes to learn about how the store is configured. For instance, the Schedule-Store-Capab describes the capabilities of the store, and Number-Of-Schedule-Segments defines how many segments are present in the Schedule-store. See Table 22 for the full list of attributes and their definitions.

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

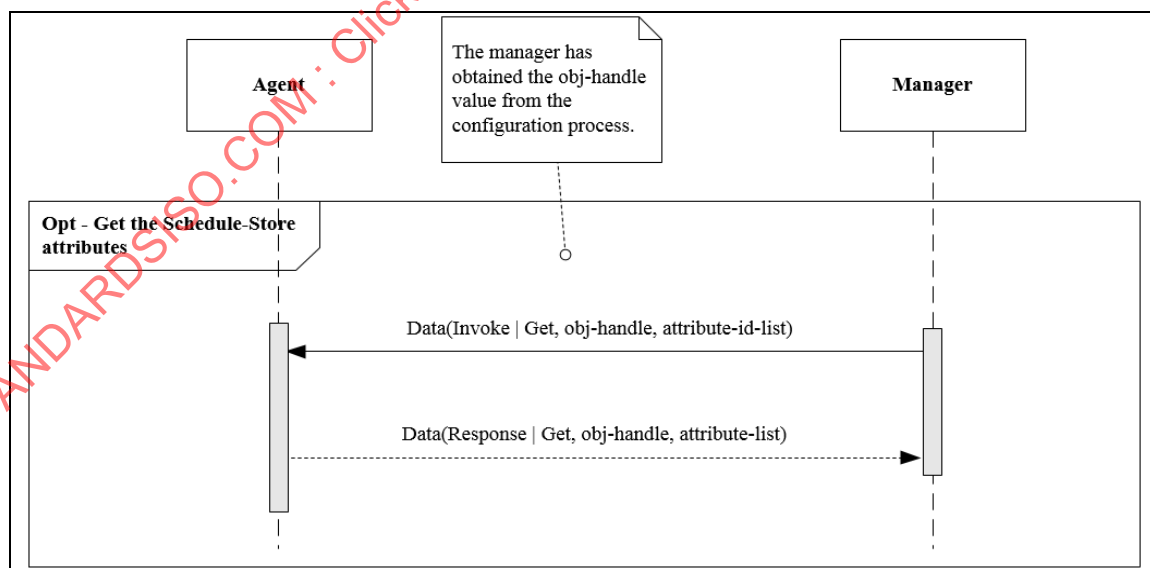


Figure 23—Retrieving attributes of a schedule-store

- b) **Retrieving the ID list of schedule-segments.** The manager retrieves information on the segments in a schedule-store by sending an ACTION.Get-Schedule-Segment-Id-List command to the specific schedule-store. The agent responds to the ACTION.Get-Schedule-Segment-Id-List command with a list of the instance numbers.
- c) **Retrieving the schedule-segment information.** The manager retrieves information on the segments in a schedule-store by sending an ACTION.Get-Schedule-Segment-Info command to the specific schedule-store (see Figure 24) with a request to return information from all segments or a particular list of segments. The agent responds to the ACTION.Get-Schedule-Segment-Info command with a list of segment numbers followed by the full attribute list for each of the segments. If there is no segment in any of these two cases, the agent responds with an empty list. The agent shall support the first selection criteria and may provide support for the second selection criteria. The manager is able to determine whether the agent provides support by inspecting the Schedule-Store-Capab attribute of the schedule-store information retrieved earlier.

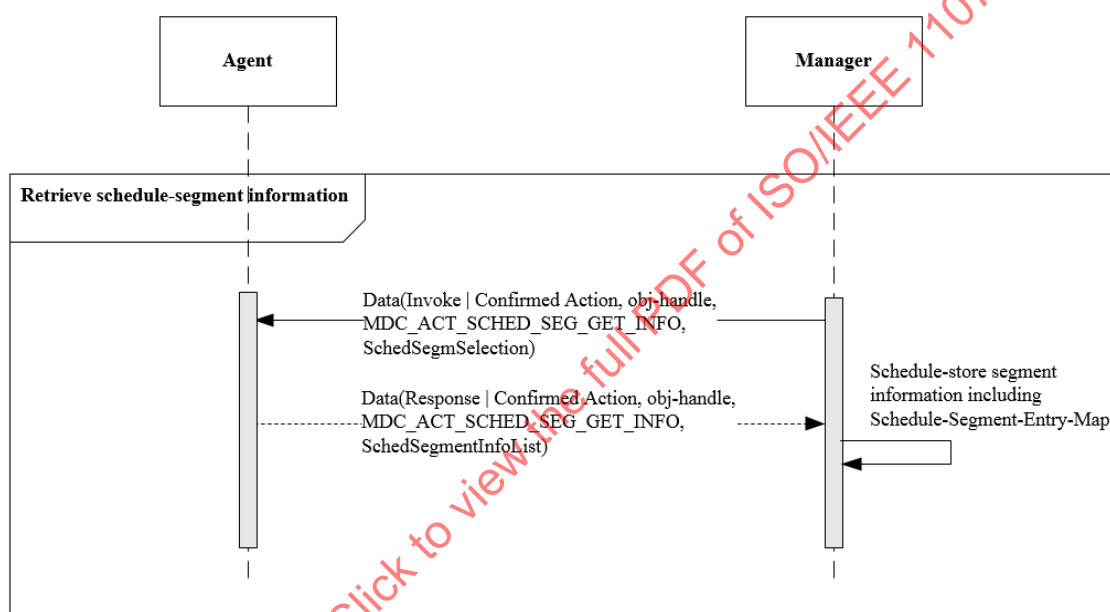


Figure 24—Retrieving schedule-segment information using Get-Schedule-Segment-Info

- d) If the manager invokes the Get-Schedule-Segment-Info method but the agent does not support the specified choice, then the agent shall respond with a roer DataApdu with a RoerErrorValue of “unsupported-choice”.
- e) **Transfer schedule-segment content.** The manager retrieves specific schedule-segments by using the Trig-Schedule-Segment-Data-Xfer ACTION method to initiate the data transfer (see Figure 25). In the first step, the manager sends the ACTION method to the agent with the handle of the schedule-store object to access. The argument to this ACTION method is the instance number of the segment to transfer.

The agent shall decide whether the request can be honored. It checks for a valid segment number, available segment data (e.g., the segment 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 txsr-successful response code to indicate that it has received the request and it can be honored.

The manager may send the Trig-Schedule-Segment-Data-Xfer ACTION invoke message at any time.

The agent shall send confirmed Schedule-Segment-Data-Event event reports until all entries in the schedule-segment are sent to the manager or the transfer is aborted by either the schsevtsta-agent-abort or schsevtsta-manager-abort bits described in A.11.8. The agent fills in the ScheduleSegmentDataEvent structure with information about the segment being sent. The agent shall transfer all segment entries in order, first entry first. The agent informs the manager of the schedule-store handle and uses the SchedSegmDataEventDescr to describe the segment number being transferred, the entry number of the first entry in the sched-segm-data-event-entries field, the number of entries in the message, and current status information. The agent shall always set any schsevtsta-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 schsevtsta-first-entry and/or schsevtsta-last-entry bits, respectively. If the agent wishes to abort the transfer, it shall set the schsevtsta-agent-abort bit to 1.

When transferring a segment, the agent uses the sched-segm-data-event-entries field to send all the entries. The agent shall start with the first entry, 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 schedule-segment Schedule-Segment-Entry-Map.

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

If the agent sets the schsevtsta-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 schsevtsta-manager-abort bit.

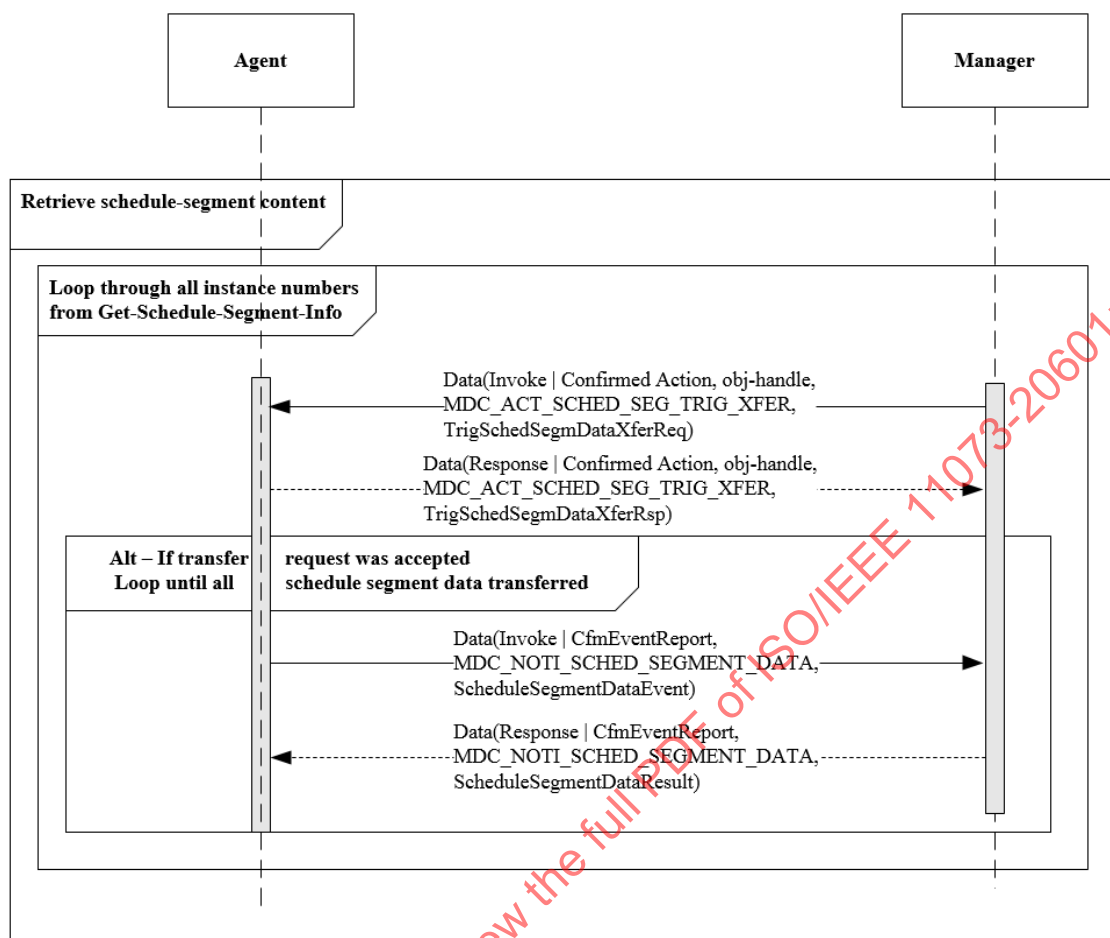


Figure 25—Retrieving schedule-segment content

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 [Transmission Control Protocol (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 by default unless another timeout applies (e.g., $TO_{clr-pms}$ overrides TO_{ca} as described in 8.9.5.6). 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:

- | | |
|--|---------------------------|
| — $TO_{cer-mds}$ (timeout for the MDS object) | MDS.Confirm-Timeout |
| — $TO_{cer-pms}$ (timeout for the PM-store object) | Segm.Confirm-Timeout |
| — $TO_{cer-sched}$ (timeout for the Schedule-store object) | SchedSegm.Confirm-Timeout |
| — $TO_{cer-scan}$ (timeout for the scanner object) | Scan.Confirm-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 four special-case timeouts that are also used in the personal health device protocol:

— $TO_{clr-pms}$: special timeout to clear the PM-store object	PMS.Clear-Timeout
— TO_{sp-mds} : special interservice timeout for the MDS object	3 s
— TO_{sp-pms} : special timeout for the PM-store segment transfer	Segm.Transfer-Timeout
— $TO_{sp-sched}$: special timeout for the Schedule-store segment transfer	SchedSegm.Transfer-Timeout

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

For TO_{sp-mds} (timeout: special interservice timeout for the MDS object), 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-mds} period. If the TO_{sp-mds} expires, the manager shall send an Association Abort message to the agent and transition back to the Unassociated state.

For TO_{sp-pms} (timeout: special segment transfer timeout of the PM-store object), after sending a Confirmed Action (MDC_ACT_SEG_TRIG_XFER) invoke message and receiving the response, the manager shall wait up to a TO_{sp-pms} period for a Confirmed Event Report (segm-evt-status=sevtsta-last-entry, segm-data-event-entries) invoke message. If the TO_{sp-pms} expires, the manager shall send an Association Abort message to the agent and transition back to the Unassociated state. The manager shall handle the Confirmed Action (MDC_ACT_SEG_TRIG_XFER) invoke message as it would any other action; it shall follow the timeout procedures described in 8.9.5.2 for Confirmed Actions.

For $TO_{sp-sched}$ (timeout: special segment transfer timeout of the Schedule-store object), after sending a Confirmed Action (MDC_ACT_SCHED_SEG_TRIG_XFER) invoke message and receiving the response, the manager shall wait up to a $TO_{sp-sched}$ period for a Confirmed Event Report (sched-segm-evt-status=schsevtsta-last-entry, sched-segm-data-event-entries) invoke message. If the $TO_{sp-sched}$ expires, the manager shall send an Association Abort message to the agent and transition back to the Unassociated state. The manager shall handle the Confirmed Action (MDC_ACT_SCHED_SEG_TRIG_XFER) invoke message as it would any other action; it shall follow the timeout procedures described in 8.9.5.2 for Confirmed Actions.

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 different Dev-Configuration-Id value in the dev-config-id field; 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 Request 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 [B21]. The MDER encoding rules are reproduced in Annex F along with additional optimizations specific to this standard. Further, for binary transmissions, network byte order (NBO) (big-endian encoding) shall be used. This standard also allows manager and agent to negotiate alternate encodings of PER [B23] and XER [B24].

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 Reference ID representation (i.e., MDC_...), but the nomenclature numeric codes shall be used during transmission regardless of which encoding rule was negotiated. 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 four types of clocks that an agent may implement: absolute time, base time with offset to local 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 3. All bit references in the subclauses are part of this attribute. If an agent has a clock of any type and this attribute is not present, the manager shall assume the default resolution of 0 (i.e., unknown). If an agent stores data (either in a PM-store object or as a “temporarily stored measurement”), it shall associate one and only one time stamp (Absolute-Time-Stamp, Base-Offset-Time-Stamp, Relative-Time-Stamp, or HiRes-Time-Stamp) with the data.

8.12.2 Absolute time

8.12.2.1 General

Agents with an internal 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.3.11) 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 Simple Network Time Protocol (SNTP)] is reported in the time-sync-protocol field using IDs such as MDC_TIME_SYNC_NTPV3. 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 to set the time with the Set-Time action. If the agent is aware that its notion of the time is inaccurate, it shall set the mds-time-mgr-set-time bit, in which case the manager shall invoke the Set-Time action command to set the absolute time on the agent. The Set-Time shall be sent within a $T_{O_{config}}$ time period after receiving the attribute from an MDS Get message. Upon receiving the Set-Time action, the agent shall clear the mds-time-mgr-set-time bit before sending the

Set-Time confirmation. If on further occasions the agent needs to have the time set, the agent shall set the mds-time-mgr-set-time bit and either wait for a GET request to be issued by the manager or send the updated attribute in a scan event report. Any agent that requests the manager to set its time by setting the mds-time-mgr-set-time bit shall not send any scan event reports until the manager sets the time. This restriction allows the agent to perform any adjustment in stored and/or temporarily stored measurements as necessary. In addition the restriction helps both the manager and agent to be working with a consistent time base when the data are received.

In some cases, 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, neither the mds-time-mgr-set-time nor the mds-time-capab-set-clock bits shall be set, and the manager shall not attempt to set the clock.

If the agent uses base offset time to report, the Set-Base-Offset-Time action shall be used instead of Set-Time action in the above description.

8.12.2.2 Comparable time

This standard utilizes a concept of “comparable time” to support all four usages of managing the time stamp of measurements. The key concepts of comparable time are as follows:

- When an agent reports time information, it shall confirm that all measurements reported as a set are from the same, unbroken timeline. For temporarily stored 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 RTC was set differently, then the agent shall 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. If the agent has no notion of the relationship between the two timelines before and after the time change, it shall report a Date-and-Time-Adjustment with a special value 0x7FFFFFFFFF to signify that there is an adjustment of unknown amount of time so that any receiver of these data is alerted and can take appropriate and agreed-upon action.
- 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 temporarily stored measurements and the Date-and-Time attribute is adjusted, the agent shall confirm 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 time stamps 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 time stamp 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). Such time adjustment shall be the cumulative adjustment between the time of the agent clock at the time of the observation and the time of the agent clock at the time of reporting each of the observations in that event report.

- If an agent collects PM-store measurements and the Date-and-Time attribute is adjusted, the agent shall confirm 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 an offset of –30 min.

8.12.3 Base time with offset

This standard includes base time with an offset in minutes to local time as a format for time stamps. The time displayed to the user, or local time, of the agent is represented as two components: one that is a base time and a second one that provides the offset in minutes from the base time to local time. This format provides the advantage over absolute time that it can maintain a contiguous time line and accommodate time changes, such as for daylight savings time (DST) or time zones. This is more efficient than using the mechanisms of comparable time and would be recommended for use in devices that store observations taken over extended periods.

Note, however, that if the base time is changed, then the time adjustment shall be indicated using the same mechanisms as for absolute time.

In this standard, base time representation follows the definition of NTP. Base time is represented as the number of seconds since 00:00 on 1 January 1900 as a 32-bit unsigned integer (without accounting for leap seconds in a similar fashion to NTP) and a 16-bit unsigned fraction of seconds field expressed as $x/65\,536$ s. A 16-bit signed integer field gives the offset from base time to local time in minutes.

The base time shall be set to be aligned with UTC and shall be set so that the offset to any local time can be accommodated by the maximum value of the offset field. If base offset time is used by the agent, then the bit `mds-time-state-bo-time-utc-aligned(14)` in `MdsTimeCapState` shall be set, and the offset carries a valid offset to local time. The `mds-time-state-bo-time-utc-aligned(14)` bit is superfluous in this version of the protocol as it is always set if there is base offset time. It has been retained for backward compatibility. In previous versions of the protocol, one could have a non-UTC-aligned base time. The option has been removed because it is not possible for a manager to validate such a time stamp.

If an agent is capable of applying DST changes appropriately so that each observation has DST applied correctly, then the bit `mds-time-dst-rules-enabled(15)` in `MdsTimeCapState` shall be set.

8.12.4 Relative time

Agents may implement a relative timer with time resolution down to 125 μ s [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 and/or a PM-store, agents shall confirm 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 receives 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 events were 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.

8.12.5 High-resolution relative time

Agents may implement an internal high-resolution timer with time resolution down to 1 μ s (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 promote 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 (e.g., scanners, scan periods, and scanner configuration)

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, it is highly recommended that additional specifications (e.g., timing, latencies, and system loading assumptions) are 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 30, Table 31, Table 32, Table 33, Table 34, Table 35, Table 36, Table 37, and Table 38. 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 30, Table 31, and Table 32 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 30 shows the general ICS.

For each implementation, one general ICS shall be provided.

IEEE Std 11073-20601-2019
 Health informatics—Personal health device communication
 Part 20601: Application profile—Optimized Exchange Protocol

Table 30—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 31 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 35)).	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 and profiles 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.		

9.4.2 Minimum requirements ICS

Table 31 shows the minimum requirements for conformance to this standard.

Table 31 —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 IEEE 11073-10101 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 32 shows the service support ICS.

Table 32—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 protocol data unit (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 33, Table 34, Table 35, Table 36, Table 37, and Table 38 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 33—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 IEEE Std 11073-10101.	—	Does the implementation use nomenclature codes from IEEE Std 11073-10101 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: 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 34 is a template only. For each object supported by the implementation, one row shall be filled out.

Table 34—Template for POC ICS

Index	Feature	Reference	Status	Support	Comment
POC- <i>n</i>	Object Description	The class of the object (e.g., numeric)	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 35 is a template only.

Table 35—Template for POC attribute ICS

Index	Feature	Reference	Status	Support	Comment
ATTR- <i>n-x</i>	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 36 is a template only. One table is provided for each object that supports special methods.

Table 36—Template for POC behavior ICS

Index	Feature	Reference	Status	Support	Comment
ACT- <i>n-x</i>	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 37 is a template only. One table is provided for each object that supports special object notifications.

Table 37—Template for medical object class (MOC) notification ICS

Index	Feature	Reference	Status	Support	Comment
NOTI- <i>n-x</i>	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 38 is a template only. One row of the table is to be used for each nomenclature element.

Table 38—Template for MOC nomenclature ICS

Index	Feature	Reference	Status	Support	Comment
NOME- <i>n</i>	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 aims to provide all data structures required to implement this standard.

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)

--
 -- Unless otherwise stated, all unused (reserved) bits in any of the BITS-* constructs shall be set to 0 by
 -- the sender and unless otherwise stated, shall be ignored by the receiver if they are set to 1.

--
 -- 8-bit bit string

BITS-8 ::= BIT STRING (SIZE(8))

--
 -- 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 IEEE Std 11073-10101. 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),          -- unspecified
    nom-part-obj(1),             -- object-oriented partition
    nom-part-metric(2),          -- metric (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-idco-extn(11),      -- implantable cardiac device observation extensions
    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 3, Table 6, Table 7, Table 8, Table 9, Table 10, Table 13, Table 14, Table 15, Table 16, and Table 18). 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 $+(2^{*}23 - 1) \rightarrow 0x007FFFFFFF]$
- NRes (not at this resolution) [exponent 0, mantissa $-(2^{*}23) \rightarrow 0x00800000]$
- + INFINITY [exponent 0, mantissa $+(2^{*}23 - 2) \rightarrow 0x007FFFFE]$
- – INFINITY [exponent 0, mantissa $-(2^{*}23 - 2) \rightarrow 0x00800002]$
- Reserved for future use [exponent 0, mantissa $-(2^{*}23 - 1) \rightarrow 0x00800001]$

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 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 two's-complement form.

Special values are assigned to express the following:

- NaN [exponent 0, mantissa $+(2^{11}-1) \rightarrow 0x07FF$]
- NRes [exponent 0, mantissa $-(2^{11}) \rightarrow 0x0800$]
- + INFINITY [exponent 0, mantissa $+(2^{11}-2) \rightarrow 0x07FE$]
- - INFINITY [exponent 0, mantissa $-(2^{11}-2) \rightarrow 0x0802$]
- Reserved for future use [exponent 0, mantissa $-(2^{11}-1) \rightarrow 0x0801$]

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  $\mu$ s (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  $\mu$ s 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 (e.g., 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 Base time with offset data type

The base time with offset data type specifies the time of day and includes a time offset field to specify the difference in minutes between the base time and the local time. Base time is encoded as the number seconds since midnight of 1st January 1900 specified as INT-U32 and the fraction of $x/65\,536$ s as INT-U16. The time offset field is specified as INT-I16. The base offset time data type is defined as follows:

```
--
BaseOffsetTime ::= SEQUENCE {
    bo-seconds    INT-U32,
    bo-fraction    INT-U16,
    bo-time-offset INT-I16
}
```

A.2.15 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)
}
-- the value notAvailable is not used in this standard
```

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)
    },
    component-id     PrivateOid,
    prod-spec        OCTET STRING
}
-- PRT-20 Participation Device
-- Serial Number
-- see note on GMDN below
-- string size shall be even
```

-- Note: The Global Medical Device Nomenclature (GMDN) is based on ISO 15225 [B14]
 -- and was developed under the auspices of CEN TC257 SC1.¹⁴

```
--
-- PowerStatus defines whether device is on battery or on mains. Upper bits define the charging
-- state.
```

¹⁴ More information can be found about this technical committee at <http://www.nkkn.net/gmdn/gmdnproject.htm>.

```
--
PowerStatus ::= BITS-16 {
    onMains(0),
    onBattery(1),
    chargingFull(8),
    chargingTrickle(9),
    chargingOff(10)
}

--
-- UdiSpec supports the encoding of unique identities such as the U.S. FDA UDI [B25].
--
UdiSpec ::= SEQUENCE OF UdiSpecification

UdiSpecification ::= SEQUENCE {
    udi-authority UdiAuthority,
    udi-issuer UdiIssuer,
    udi-label UdiLabel
}

UdiAuthority ::= ITU-OID;
-- Identifies the jurisdictive organization for the UDI, such as the U.S. FDA.
-- The OID for U.S. FDA is 2.16.840.1.113883.3.24

UdiIssuer ::= ITU-OID;
-- Organization that is charged with issuing UDIs for devices.
-- For example, the U.S. FDA issuers include
-- GS1(2.51), HIBCC(1.0.15961.13.10), and ICCBBA(2.16.840.1.113883.6.18).
-- An OID length of 0 should be used if the issuer is unknown or not recorded in the
-- device.

ITU-OID ::= OCTET STRING
-- a OID can be used as an unique ID for an organization
-- OID trees are maintained by registration authorities with ITU-T and ISO at the top;
-- HL7 v2.8 [B1] manages its own OID tree;
-- see http://oid-info.com/index.htm
-- note that HL7 v2.8 [B1] FHIR is using a different encoding, i.e. a URI ( a URI may
-- include an OID)

UdiLabel ::= OCTET STRING
-- full text label for the UDI—can be mapped to HL7 v2.8 [B1] udi PRT10
-- matches the human readable UDI string that is part of the labeling of the device

--
-- All measures about the battery are values with their dimensions. See the description
-- of Remaining-Battery-Time in Table 3 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),          -- indicates the value was measured during a
                                    -- calibration procedure
    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)
    msmt-value-exceed-boundaries(14), -- indicates that the measurement is outside threshold
                                    -- boundaries set in the CurrentLimits attribute.
    msmt-state-ann-inhibited(15)    -- indicates that threshold indication is disabled
                                    -- and should not cause an annunciation. If
                                    -- this bit is set, the bit 14 shall not be set.
}
```

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
    state          MeasurementStatus, -- Metric::Type attribute of the numeric object.
    unit-code      OID-Type,          -- from nom-part-dim dimensions nomenclature
    value          FLOAT-Type         -- partition
}
```

--
 -- 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 [B17] 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
}
```

-- The capability mask defines if the corresponding bit of an enumeration bit string in that object is
 -- supported.

```
CapabMaskSimp ::= BITS-32 {
    bit_0_supported(0),    -- bit is supported: 1 | bit is not supported: 0
    bit_1_supported(1),    -- bit is supported: 1 | bit is not supported: 0
    bit_2_supported(2),    -- bit is supported: 1 | bit is not supported: 0
    bit_3_supported(3),    -- bit is supported: 1 | bit is not supported: 0
    bit_4_supported(4),    -- bit is supported: 1 | bit is not supported: 0
}
```

IEEE Std 11073-20601-2019
 Health informatics—Personal health device communication
 Part 20601: Application profile—Optimized Exchange Protocol

```

    bit_5_supported(5),          -- bit is supported: 1 | bit is not supported: 0
    bit_6_supported(6),          -- bit is supported: 1 | bit is not supported: 0
    bit_7_supported(7),          -- bit is supported: 1 | bit is not supported: 0
    bit_8_supported(8),          -- bit is supported: 1 | bit is not supported: 0
    bit_9_supported(9),          -- bit is supported: 1 | bit is not supported: 0
    bit_10_supported(10),        -- bit is supported: 1 | bit is not supported: 0
    bit_11_supported(11),        -- bit is supported: 1 | bit is not supported: 0
    bit_12_supported(12),        -- bit is supported: 1 | bit is not supported: 0
    bit_13_supported(13),        -- bit is supported: 1 | bit is not supported: 0
    bit_14_supported(14),        -- bit is supported: 1 | bit is not supported: 0
    bit_15_supported(15),        -- bit is supported: 1 | bit is not supported: 0
    bit_16_supported(16),        -- bit is supported: 1 | bit is not supported: 0
    bit_17_supported(17),        -- bit is supported: 1 | bit is not supported: 0
    bit_18_supported(18),        -- bit is supported: 1 | bit is not supported: 0
    bit_19_supported(19),        -- bit is supported: 1 | bit is not supported: 0
    bit_20_supported(20),        -- bit is supported: 1 | bit is not supported: 0
    bit_21_supported(21),        -- bit is supported: 1 | bit is not supported: 0
    bit_22_supported(22),        -- bit is supported: 1 | bit is not supported: 0
    bit_23_supported(23),        -- bit is supported: 1 | bit is not supported: 0
    bit_24_supported(24),        -- bit is supported: 1 | bit is not supported: 0
    bit_25_supported(25),        -- bit is supported: 1 | bit is not supported: 0
    bit_26_supported(26),        -- bit is supported: 1 | bit is not supported: 0
    bit_27_supported(27),        -- bit is supported: 1 | bit is not supported: 0
    bit_28_supported(28),        -- bit is supported: 1 | bit is not supported: 0
    bit_29_supported(29),        -- bit is supported: 1 | bit is not supported: 0
    bit_30_supported(30),        -- bit is supported: 1 | bit is not supported: 0
    bit_31_supported(31),        -- bit is supported: 1 | bit is not supported: 0
  }

  CapabMaskBasic ::=BITS-16 {
    bit_0_supported(0),          -- bit is supported: 1 | bit is not supported: 0
    bit_1_supported(1),          -- bit is supported: 1 | bit is not supported: 0
    bit_2_supported(2),          -- bit is supported: 1 | bit is not supported: 0
    bit_3_supported(3),          -- bit is supported: 1 | bit is not supported: 0
    bit_4_supported(4),          -- bit is supported: 1 | bit is not supported: 0
    bit_5_supported(5),          -- bit is supported: 1 | bit is not supported: 0
    bit_6_supported(6),          -- bit is supported: 1 | bit is not supported: 0
    bit_7_supported(7),          -- bit is supported: 1 | bit is not supported: 0
    bit_8_supported(8),          -- bit is supported: 1 | bit is not supported: 0
    bit_9_supported(9),          -- bit is supported: 1 | bit is not supported: 0
    bit_10_supported(10),        -- bit is supported: 1 | bit is not supported: 0
    bit_11_supported(11),        -- bit is supported: 1 | bit is not supported: 0
    bit_12_supported(12),        -- bit is supported: 1 | bit is not supported: 0
    bit_13_supported(13),        -- bit is supported: 1 | bit is not supported: 0
    bit_14_supported(14),        -- bit is supported: 1 | bit is not supported: 0
    bit_15_supported(15),        -- bit is supported: 1 | bit is not supported: 0
  }

  --
  -- The state flag defines if the corresponding bit of an enumeration bit string in that object is a state or an
  -- event. For a bit that is a state, then the value is set to 1.
  --

  StateFlagSimp ::=BITS-32 {
    bit_0_state(0),             -- bit is state: 1 | bit is event: 0
    bit_1_state(1),             -- bit is state: 1 | bit is event: 0
  }

```

```

bit_2_state(2), -- bit is state: 1 | bit is event: 0
bit_3_state(3), -- bit is state: 1 | bit is event: 0
bit_4_state(4), -- bit is state: 1 | bit is event: 0
bit_5_state(5), -- bit is state: 1 | bit is event: 0
bit_6_state(6), -- bit is state: 1 | bit is event: 0
bit_7_state(7), -- bit is state: 1 | bit is event: 0
bit_8_state(8), -- bit is state: 1 | bit is event: 0
bit_9_state(9), -- bit is state: 1 | bit is event: 0
bit_10_state(10), -- bit is state: 1 | bit is event: 0
bit_11_state(11), -- bit is state: 1 | bit is event: 0
bit_12_state(12), -- bit is state: 1 | bit is event: 0
bit_13_state(13), -- bit is state: 1 | bit is event: 0
bit_14_state(14), -- bit is state: 1 | bit is event: 0
bit_15_state(15), -- bit is state: 1 | bit is event: 0
bit_16_state(16), -- bit is state: 1 | bit is event: 0
bit_17_state(17), -- bit is state: 1 | bit is event: 0
bit_18_state(18), -- bit is state: 1 | bit is event: 0
bit_19_state(19), -- bit is state: 1 | bit is event: 0
bit_20_state(20), -- bit is state: 1 | bit is event: 0
bit_21_state(21), -- bit is state: 1 | bit is event: 0
bit_22_state(22), -- bit is state: 1 | bit is event: 0
bit_23_state(23), -- bit is state: 1 | bit is event: 0
bit_24_state(24), -- bit is state: 1 | bit is event: 0
bit_25_state(25), -- bit is state: 1 | bit is event: 0
bit_26_state(26), -- bit is state: 1 | bit is event: 0
bit_27_state(27), -- bit is state: 1 | bit is event: 0
bit_28_state(28), -- bit is state: 1 | bit is event: 0
bit_29_state(29), -- bit is state: 1 | bit is event: 0
bit_30_state(30), -- bit is state: 1 | bit is event: 0
bit_31_state(31) -- bit is state: 1 | bit is event: 0
}

```

```

StateFlagBasic ::=BITS-16 {
    bit_0_state(0), -- bit is state: 1 | bit is event: 0
    bit_1_state(1), -- bit is state: 1 | bit is event: 0
    bit_2_state(2), -- bit is state: 1 | bit is event: 0
    bit_3_state(3), -- bit is state: 1 | bit is event: 0
    bit_4_state(4), -- bit is state: 1 | bit is event: 0
    bit_5_state(5), -- bit is state: 1 | bit is event: 0
    bit_6_state(6), -- bit is state: 1 | bit is event: 0
    bit_7_state(7), -- bit is state: 1 | bit is event: 0
    bit_8_state(8), -- bit is state: 1 | bit is event: 0
    bit_9_state(9), -- bit is state: 1 | bit is event: 0
    bit_10_state(10), -- bit is state: 1 | bit is event: 0
    bit_11_state(11), -- bit is state: 1 | bit is event: 0
    bit_12_state(12), -- bit is state: 1 | bit is event: 0
    bit_13_state(13), -- bit is state: 1 | bit is event: 0
    bit_14_state(14), -- bit is state: 1 | bit is event: 0
    bit_15_state(15) -- bit is state: 1 | bit is event: 0
}

```

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.3.11 PHD DM Status attributes

```
--
-- PHDDMStat allows generic device events to be recorded in order to track important events for the user
-- and troubleshooting information for manufacturers.
--
```

```
PHDDMStat ::= BITS-32 {
    device-status-undetermined (0),
    device-status-reset (1),
    -- reserved for future extension (2),
    -- reserved for future extension (3),
    -- reserved for future extension (4),
    device-status-error (5),
    device-status-error-mechanical (6),
    device-status-error-electronic (7),
    device-status-error-software (8),
```

```

device-status-error-battery (9),
-- reserved for future extension (10),
-- reserved for future extension (11),
-- reserved for future extension (12),
-- reserved for future extension (13),
-- reserved for future extension (14),
device-status-service (15),
device-status-service-time-sync-required (16)
device-status-service-calibration-required (17),
device-status-service-replenishment-required (18),
-- reserved for future extension (19),
-- reserved for future extension (20),
-- reserved for future extension (21),
-- reserved for future extension (22),
-- reserved for future extension (23),
-- reserved for future extension (24),
device-status-battery-low (25),
device-status-battery-depleted (26),
device-status-battery-replaced (27),
device-status-battery-interrupted (28)
-- reserved for future extension (29),
-- reserved for future extension (30),
-- reserved for future extension (31),
}

```

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
                                -- [B17], but not used. Thus, it shall be zero (0).
}

--
-- SetBOTimeInvoke selects the date and time to be set in base offset time format. If both the seconds and
-- fraction fields are set to 0x0, they are ignored in the action and only the offset is changed.
--
SetBOTimeInvoke ::= SEQUENCE {
    date-time      BaseOffsetTime
}

--
-- 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

```

IEEE Std 11073-20601-2019
 Health informatics—Personal health device communication
 Part 20601: Application profile—Optimized Exchange Protocol

```

-- PM-segments, e.g., from a previous
-- Get-Segment-Info or Get-Segment-Id-List
-- method call.
abs-time-range    [3] AbsTimeRange,
-- support of abs-time-range is optional, indicated in
-- the PM-Store-Capab attribute
bo-time-range     [4] BOTimeRange
-- support of bo-time-range is optional, indicated in
-- the PM-Store-Capab attribute
}

--
-- SegmIdList selects PM-segments by ID.
-- SegmIdList may be returned in response to the Get-Segment-Id-List method, containing a list of
-- the instance numbers of all the PM-segments of a PM-store.
SegmIdList ::= SEQUENCE OF InstNumber

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

--
-- BOTimeRange allows selection of PM-segments by time period specified as base offset time.
--
BOTimeRange ::= SEQUENCE {
    from-time      BaseOffsetTime,
    to-time        BaseOffsetTime
}

--
-- 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
}

--
-- SchedSegmSelection selects the Schedule-segments that are subject to the method.
SchedSegmSelection ::= CHOICE {
    all-sched-segments [1] INT-U16, -- if this type is chosen to select all schedule
    -- segments, the actual contents of the field is "do
    -- not care" and shall be zero
    sched-segm-id-list [2] SchedSegmIdList,
    -- using this list requires that the manager already
    -- know the Schedule-Instance-Number attributes
    -- of the Schedule-segments, e.g., from a previous
    -- Get-Schedule-Segment-Info or

```

-- Get-Schedule-Segment-Id-List method call.

```

}

--
-- SchedSegmIdList selects Schedule-segments by ID.
--
SchedSegmIdList ::= SEQUENCE OF InstNumber

--
-- ScheduleSegmentInfoList returns the object attributes (except the Fixed-Schedule-Segment-Data) of all
-- selected Schedule-segment object instances in response to the Get-Schedule-Segment-Info or
-- Get-Schedule-Segment-Id-List Schedule-store method.
-- This is required by the manager to retrieve the dynamic information about the schedule segments.
--
ScheduleSegmentInfoList ::= SEQUENCE OF ScheduleSegmentInfo

ScheduleSegmentInfo ::= SEQUENCE {
    sched-seg-inst-no      InstNumber,
    sched-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.
 unsupported-agent(3) -- this bit shall be used only by manager. It is used if
 -- manager does not support the agent's
 -- configuration(s), MDS, or other objects that are
 -- currently used by agent.
}

-- 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

IEEE Std 11073-20601-2019
 Health informatics—Personal health device communication
 Part 20601: Application profile—Optimized Exchange Protocol

```

-- 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 IEEE Std 11073-20601-2008
                           -- is supported
    protocol-version2(1),  -- This bit shall be set if IEEE Std 11073-20601a-
                           -- 2010 is supported
    protocol-version3(2),  -- This bit shall be set if IEEE Std 11073-20601-2014
                           -- is supported.
    protocol-version4(3),  -- This bit shall be set if IEEE Std 11073-20601-2019
                           -- 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 {
    nom-version1(0)       -- values reference a specific nomenclature standard
                           -- This bit shall be set if version 1 is supported
    nom-version2(1)       -- This bit shall be set if version 2 is supported
}

```

```

        nom-version3(2)          -- This bit shall be set if version 3 is supported
    }
    -- In terms of IEEE Std 11073-10101, nom-version1 corresponds to IEEE Std 11073-10101-2004,
    -- nom-version2 corresponds to IEEE Std 11073-10101a-2015, and nom-version3 corresponds to
    -- IEEE Std 11073-10101-2019.

    -- 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.
        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.

--
 DataA pdu ::= 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 DataA pdu 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 [B21] defines a number of RoerErrorValue values that are not
 -- defined in this standard. Therefore, for consistency, numbering of the RoerErrorValue in this standard
 -- skips any value already defined in that ISO/IEEE standard.

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),
- unsupported-choice is returned when an action is attempted on an object
- but the object does not support the choice that the action uses.
- 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
- unsupported-choice(27)
- invalid-choice is returned when an action is attempted on an object
- but the choice this action uses is neither defined in this standard nor understood or
- provided by the agent. 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
- invalid-choice(28)

}

```
-- At any point, if a DataAdu invoked action (roiv-*) requires the receiver to reject an
-- operation prior to attempting to execute it, the receiver shall send
-- back a RejectResult. The invokeID is used to determine which invocation resulted in a
-- rejection condition. The problem field shall be filled in with a value from the RorjProblem list
-- below.
```

```
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, the 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 5, Table 12, Table 17, and Table 19, 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 5, Table 12, Table 17, and Table 19, the corresponding
 -- event-reply-info shall be used. Accordingly event-reply-info will be empty, ConfigReportRsp,
 -- or SegmentDataResult.

```
EventReportResultSimple ::= SEQUENCE {
    obj-handle      HANDLE,
    current-time    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, the obj-handle shall either be 0 to represent the MDS object
 -- or a handle representing a PM-store object or a handle representing a Schedule-store object.
 -- The attribute-id-list shall be left empty if the purpose is to query for all attributes of the MDS or
 -- PM-store object or Schedule-store object.
 -- Alternatively, specific attributes of an object may be queried by listing the desired
 -- Attribute IDs found in Table 3 or Table 10 or Table 22.

```
GetArgumentSimple ::= SEQUENCE {
    obj-handle      HANDLE,
    attribute-id-list AttributeIdList
}
```

-- For GET responses defined in this standard, the 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 IEEE Std 11073-10101,
 -- 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, the 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, the 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 4 and Table 11, the corresponding action-info-args
 -- structures shall be used. Accordingly, action-info-args will be one of DataRequest,
 -- SetTimeInvoke, SetBOTimeInvoke, SegmSelection, TrigSegmDataXferReq or
 -- TrigSchedSegmDataXferReq.

```
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, the 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 4 and Table 11, the resulting action-info-args
 -- shall be used. Accordingly, action-info-args will be empty, DataResponse,
 -- SegmentInfoList, TrigSegmDataXferRsp or TrigSchedSegmDataXferRsp.

```
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. This value is the
 -- accumulated difference between internal clock and external reference source since last
 -- synchronization.

```
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,	-- if -- mds-time-capab-real-time-clock -- is set, then this indicates the -- 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. -- if mds-time-capab-bo-time -- is set, then this indicates the -- resolution of the agent's -- base time clock. -- 0 if unknown; otherwise, -- the number of 1/65536 s -- that elapse with each clock -- increment. The value of 0xFFFF -- is reserved to indicate an interval -- 1 s.
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 -- 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.
}		

- Only one of mds-time-capab-real-time-clock and mds-time-capab-bo-time shall be specified.
- Only one of mds-time-capab-sync-abs-time and mds-time-capab-sync-bo-time shall be specified.
- Only one of mds-time-state-abs-time-synced and mds-time-state-bo-time-synced shall be specified.

MdsTimeCapState ::= BITS-16 {

- | | |
|---|--|
| mds-time-capab-real-time-clock(0), | -- device supports an internal RTC
-- following absolute time |
| mds-time-capab-set-clock(1), | -- device supports Set-Time Action
-- or Set-Base-OffsetTime Action |
| mds-time-capab-relative-time(2), | -- device supports RelativeTime |
| mds-time-capab-high-res-relative-time(3), | -- device supports |

```

    mds-time-capab-sync-abs-time(4),
    mds-time-capab-sync-rel-time(5),
    mds-time-capab-sync-hi-res-relative-time(6),
    mds-time-capab-bo-time(7),
    mds-time-state-abs-time-synced(8),
    mds-time-state-rel-time-synced(9),
    mds-time-state-hi-res-relative-time-synced(10),
    mds-time-mgr-set-time(11),
    mds-time-capab-sync-bo-time(12),
    mds-time-state-bo-time-synced(13),
    mds-time-state-bo-time-UTC-aligned(14)
    mds-time-dst-rules-enabled(15)

-- HighResRelativeTime
-- device syncs AbsoluteTime
-- device syncs RelativeTime
-- device syncs HiResRelativeTime
-- device supports BaseOffsetTime
-- AbsoluteTime is synced
-- RelativeTime is synced
-- HiResRelativeTime is synced
-- manager shall set the time
-- device syncs BaseOffsetTime
-- base time is synced
-- base time is aligned to UTC
-- device supports and applies DST
-- rules
}

-- *****
-- 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-cu(),
-- 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 [B17]. It defines availability, periodicity, and category of the measurement.
-- The setting of bits 0 to 5 is primarily informational and shall be set if the condition is true but
-- a manager cannot assume that if they are set the behavior will be observed.
-- 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-msmt-manager-initiated-immediate(7) -- the agent can provide an immediate
                                   -- measurement in response to a data request. If not
                                   -- set, then the agent should return the most recent
                                   -- measurement or setting
    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 [B17], but enhanced by
-- value ms-struct::ms-struct-compound-fix. For IEEE Std 11073-20601-2014, ms-struct-compound
```


-- and ms-struct-compound-fix shall only be used for numeric objects. Additional structures would
 -- need to be introduced in RT-SA and enumeration objects to allow the use of compound structures.
 -- If compound is used, the agent shall send no more than ms-comp-no observed values.
 --
 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) -- [B17] 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 object attribute value
 -- changes (Attribute Change Sets). 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 the 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      DataReqId,
    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-confirmed(61440)   -- 0xF000
    -- data-req-id-agent-initiated-confirmed shall be used in agent-initiated measurement
    -- data transmission via confirmed event report.
    --
    data-req-id-agent-initiated-unconfirmed(61441) -- 0xF001
}

```

IEEE Std 11073-20601-2019
Health informatics—Personal health device communication
Part 20601: Application profile—Optimized Exchange Protocol

-- data-req-id-agent-initiated-unconfirmed shall be used in agent-initiated measurement
-- data transmission via unconfirmed event report.
--
-- Values between 0xF002 and 0xFFFF, inclusive, are reserved.

}

--

-- The value used for person-id is vendor determined (e.g., 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 DataReqId,
 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 5)

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.
                                     -- The scan-event-report-no of the
                                     -- returned ScanReportInfo shall be
                                     -- set to zero.
    data-req-mode     DataReqMode,   -- Defines the mode by setting one
                                     -- bit.
    data-req-obj-handle-list  HANDLEList -- contains the handles of objects
                                     -- that the manager is requesting.
  }

  -- All unassigned "DataReqMode" bit values are reserved for future expansion and shall be set to zero.
  DataReqMode ::= BITS-16 {
    data-req-scope-handle(6)         -- Shall be set in this version.
  }

  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.
                                                  -- Shall currently be set only
                                                  -- to 0 or 1
  }

  -- All unassigned " DataReqModeFlags " bit values are reserved for future expansion and
  -- shall be set to zero.
  DataReqModeFlags ::= BITS-16 {
    data-req-supp-init-manager(8),    -- this field is used in the association to flag
    data-req-supp-person-id(11),      -- data request capabilities
    data-req-supp-init-agent(15)      -- supports manager-init.
                                     -- used only with agent-initiated data flows
                                     -- agent uses agent-initiated data requests/flows
  }

```

```

}

-- DataResponse is returned as a result of an MDS-Data-Request (see Table 4). The event-type and
-- event-info fields are filled in using the same parameters as found in MDS object events.
-- See Table 5 for the permitted 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 {
    data-req-result    DataReqResult,
    event-type         OID-Type, -- event-type shall be 0 and event-info.length = 0
                                -- if data-req-result is not data-req-result-no-error.
                                -- Otherwise value taken 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),
    data-req-result-rejected(2) -- pending agent-initiated data exists

    -- The following error codes are returned when the manager request contains
    -- a scope that is not supported by the agent.
    data-req-result-no-init-manager-support(6), -- manager-init is not supported by agent

    -- The following error codes are returned when the manager request contains
    -- unknown values in the supporting fields
    data-req-result-unknown-handle(10),
    data-req-result-request-overflow(11), -- manager has tried to make more requests
                                         -- than supported
    data-req-result-unsupp-scope(14), -- too many scope bits set
}

```

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

-- Alert-Op-State attribute defines the current limit alert state.

-- Note: The bits refer to the limit alarm only, not to the global alert state of the metric.

--

CurLimAlStat ::= BITS-16 {
 lim-alert-off(0), -- if this bit is set, all alerts (both high and low) are off
 lim-low-off(1), -- low-limit violation detection is off
 lim-high-off(2) -- high-limit violation detection is off
}

-- Current-Limits attribute

--

CurLimAlVal ::= SEQUENCE {
 lower FLOAT-Type,
 upper FLOAT-Type
}

-- Alert-Op-Text-String attribute assigns individual labels to upper and lower alarm limit.

--

AlertOpTextString ::= SEQUENCE {
 lower-text OCTET STRING,
 upper-text OCTET STRING
}

-- Measurement-Confidence-95 attribute defines the lower and upper bounds for a range within which the
-- manufacture is 95% confident the actual measurement value resides.

--

-- Note: The unit for the lower and upper bounds is the same as for the measurement.

--

MeasurementConfidence95 ::= SEQUENCE {
 lower-bound SFLOAT-type
 upper-bound SFLOAT-type
}

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 {
 pm-sc-var-no-of-segm(0), -- indicates that the number of PM-segments
 -- contained in this PM-store is dynamic and may
 -- change
 pm-sc-segm-id-list-select(3), -- PM-segments in the SegmSelection data type can
 -- be selected by defining a list of segment IDs
}

IEEE Std 11073-20601-2019
 Health informatics—Personal health device communication
 Part 20601: Application profile—Optimized Exchange Protocol

```

    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 or
                                -- bo-time-range depending upon which time
                                -- mode the device supports
    pmsc-clear-segm-by-list-sup(7), -- clearing a list of segments is supported
    pmsc-clear-segm-by-time-sup(8), -- clearing segments by abs-time-range or
                                -- bo-time-range is supported depending
                                -- upon which time mode the device supports
    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-clear-segm-all-sup(10),  -- clearing all segments is supported
    pmsc-multi-person(12)         -- The PM-store supports PM-segment for more
                                -- than one person
    pmsc-get-segm-id-list-sup(14), -- The Get-Segment-Id-List method is supported.
  }

  --
  -- 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 are 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
                                                -- (SegmentEntryHeader)
    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.
  --
  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)
  }

```

```

seg-elem-hdr-bo-time(3)      -- entry preceded by base offset time
                             -- (data type BaseOffsetTime)
                             -- option (0) and option (3) are mutually exclusive
}

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 object-oriented
                                -- 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 agent shall transfer all segment entries in FIFO order.
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 stop the segment
    -- transfer and respond with an error (roer) code
    -- of protocol-violation).
    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)
}

```

A.11.9 Schedule store and schedule-segment related data types

```

--
-- The instance number list a list of InstNumbers..

InstNumberList ::= SEQUENCE OF InstNumber

--
-- The Schedule-Store-Capab attribute defines specific static capabilities and properties of the Schedule-
-- store object instance.
-- The default value of this attribute is 0 (no bits set).

```

-- All unassigned "SchedStoreCapab" bit values are reserved for future expansion and shall be set to zero.

--

```
SchedStoreCapab ::= BITS-32 {
    schedsc-var-no-of-segm(0),          -- indicates that the number of Schedule-segments
                                        -- contained in this Schedule-store is dynamic and
                                        -- may change
    schedsc-segm-id-list-select(3),     -- Schedule-segments in the SchedSegmSelection
                                        -- data type can be selected by defining a list of
                                        -- segment IDs
    schedsc-epi-seg-entries(4),        -- Schedule-segments may contain
                                        -- episodic/aperiodic entries and therefore have
                                        -- to contain explicit time stamp information
    schedsc-peri-seg-entries(5),       -- Schedule-segments may contain periodic
                                        -- entries and therefore the Schedule
                                        -- segment or Schedule-store shall support the
                                        -- Schedule-Sample-Period attribute
    schedsc-multi-person(12)           -- The Schedule-store supports Schedule-segment for
                                        -- more than one person
    schedsc-get-segm-info-sup(13),      -- The Get-Schedule-Segment-Info method is
                                        -- supported.
    schedsc-get-segm-id-list-sup(14),   -- The Get-Schedule-Segment-Id-List method is
                                        -- supported.
}
```

}

--

-- All entries in the segment shall follow the format defined by this attribute. First, the mandatory header shall follow the description in sched-segm-entry-header. This allows each entry in the segment to be preceded by a header (e.g., periodicity, start time, and duration information) that is applicable to all elements in an entry. Next, the elements shall follow the format and order described in sched-segm-entry-elem-list. An element typically represents a setting. For each element, the stored data are defined in the form of an attribute value map, in the same way as metric objects.

--

```
ScheduleSegmentEntryMap ::= SEQUENCE {
    sched-segm-entry-header SchedSegmEntryHeader, -- defines start time type in front
                                                    -- of each entry
                                                    -- (SchedSegmEntryHeader)
    sched-segm-entry-elem-list SchedSegmEntryElemList
}
```

--

-- The following format defines data items that are in front of each schedule-segment entry. The conditional header allows definition of data items that are common to all elements in the entry. If the Schedule-Segment-Entry-Interval is used, then the header shall not be used. The schedule segment entry event report shall begin with data from the first element.

```
SchedSegmEntryHeader ::= BITS-16 {
    sched-seg-hdr-absolute-time(0),      -- entry preceded by absolute time
                                        -- (data type AbsoluteTime)
    sched-seg-hdr-relative-time(1),      -- entry preceded by relative time
                                        -- (data type RelativeTime)
    sched-seg-hdr-hires-relative-time(2), -- entry preceded by high-resolution relative
                                        -- time (data type HighResRelativeTime)
    sched-seg-hdr-bo-time(3)             -- entry preceded by base offset time
                                        -- (data type BaseOffsetTime)
}
```

```

-- option (0) and option (3) are mutually
-- exclusive
}
--
-- All unassigned " SchedSegmEntryHeader " bit values are reserved for future expansion and shall be set
-- to zero.
-- If the Schedule-segment is periodic, the SchedSegmElemStartTime shall be relative time or
-- high-resolution relative time relative to the Schedule-Segment Reference Time attribute.
--

SchedSegmEntryElemList ::= SEQUENCE OF SchedSegmEntryElem

--
-- SchedSegmEntryElem shall reference a metric object instance in the agent configuration
-- using its handle value. This referenced object shall exist in the agent configuration.
--
SchedSegmEntryElem ::= SEQUENCE {
    handle          HANDLE,          -- handle of referenced object
    attr-val-map    AttrValMap      -- attribute value map describing the data
}

--
-- Request to start the transfer of the specified segment
--
TrigSchedSegmDataXferReq ::= SEQUENCE {
    sched-seg-inst-no InstNumber
}

TrigSchedSegmDataXferRsp ::= SEQUENCE {
    Sched-seg-inst-no      InstNumber,
    trig-sched-segm-xfer-rsp TrigSchedSegmXferRsp
}

--
-- All unassigned " TrigSchedSegmXferRsp " values are reserved for future expansion and shall not be
-- used.
--
TrigSchedSegmXferRsp ::= INT-U16 {
    tschsxr-successful(0),          -- Agent will start transfer of schedule segment
    tschsxr-fail-no-such-segment(1), -- schedule-segment ID not found
    tschsxr-fail-segm-try-later(2), -- the requested schedule-segment is opened by
                                     -- the user and is locked
    tschsxr-fail-segm-empty(3),     -- the schedule segment being requested is empty
    tschsxr-fail-not-otherwise-specified(512)
}

-- ScheduleSegmentDataEvent
--
-- The agent shall transfer all segment entries in order, first entry first (FIFO).
--

ScheduleSegmentDataEvent ::= SEQUENCE {
    sched-segm-data-event-descr SchedSegmDataEventDescr,
    sched-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.

$$\}$$

```
ScheduleSegmentDataResult ::= SEQUENCE {
    sched-segm-data-event-descr SchedSegmDataEventDescr
}
```

—

-- The Schedule Segment Data Event Descriptor defines which entries of the Schedule Segment Data are communicated in the Event message.

—

```
SchedSegmDataEventDescr ::= SEQUENCE {
    sched-segm-instance      InstNumber,      -- instance number of segment being
                                         -- transferred
    sched-segm-evt-entry-index  INT-U32,      -- array index of the first entry in this event
    sched-segm-evt-entry-count  INT-U32,      -- count of entries in this event
    sched-segm-evt-status      SchedSegmEvtStatus
}
```

}

—

-- All unassigned " SchedSegmEvtStatus " bit values are reserved for future expansion and shall be set to
-- zero.

—

```
SchedSegmEvtStatus ::= BITS-16 {
    schsevtsta-first-entry(0),           -- this event contains the first segment entry
    schsevtsta-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)
    schsevtsta-agent-abort(4),          -- transfer aborted by agent (manager shall reply
                                         -- with the same status)
    schsevtsta-manager-confirm(8),      -- set in reply if segment was received correctly (if
                                         -- not set in reply, agent shall stop the segment
                                         -- transfer and respond with an error (roer) code
                                         -- of protocol-violation).
    schsevtsta-manager-abort(12)        -- sent in reply by manager (agent shall stop sending
                                         -- messages)
```

$$\}$$

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 need to 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

$$M = (50.0 - (-45.0)) / (190 - 0) = 0.5$$

$$B = 50.0 - (0.5 \times 190) = -45.0$$

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

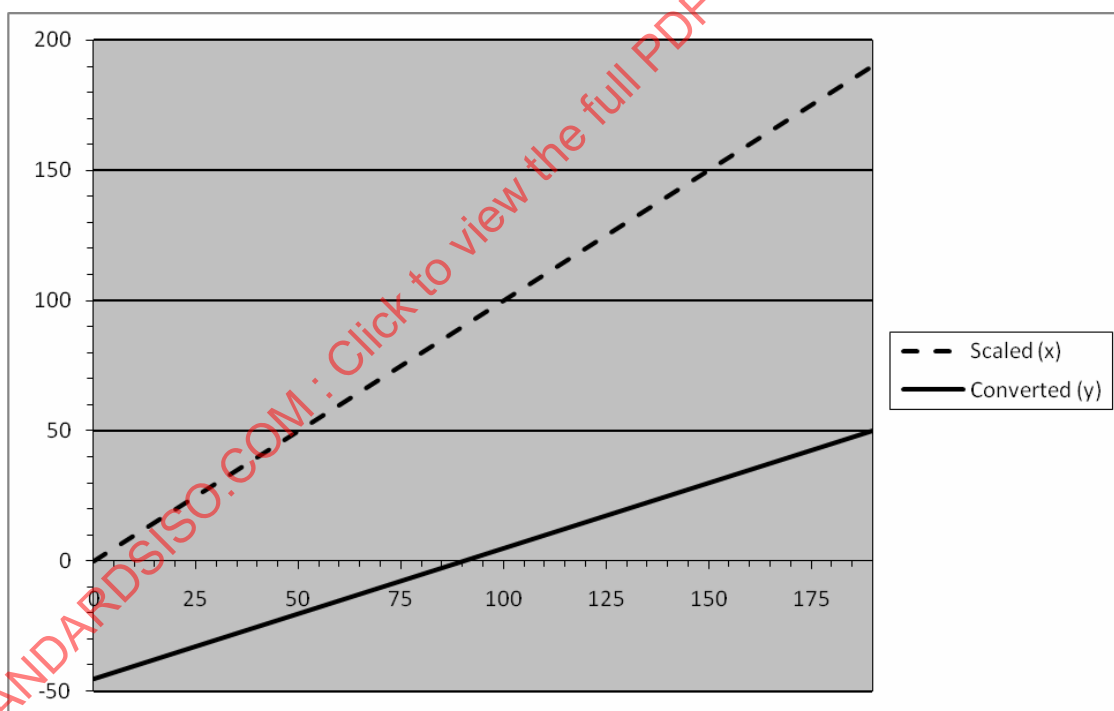


Figure B.1—Graphical view of conversion

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 schedule segment 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 (PM) store object hierarchy

C.2.1 General

A PM 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.

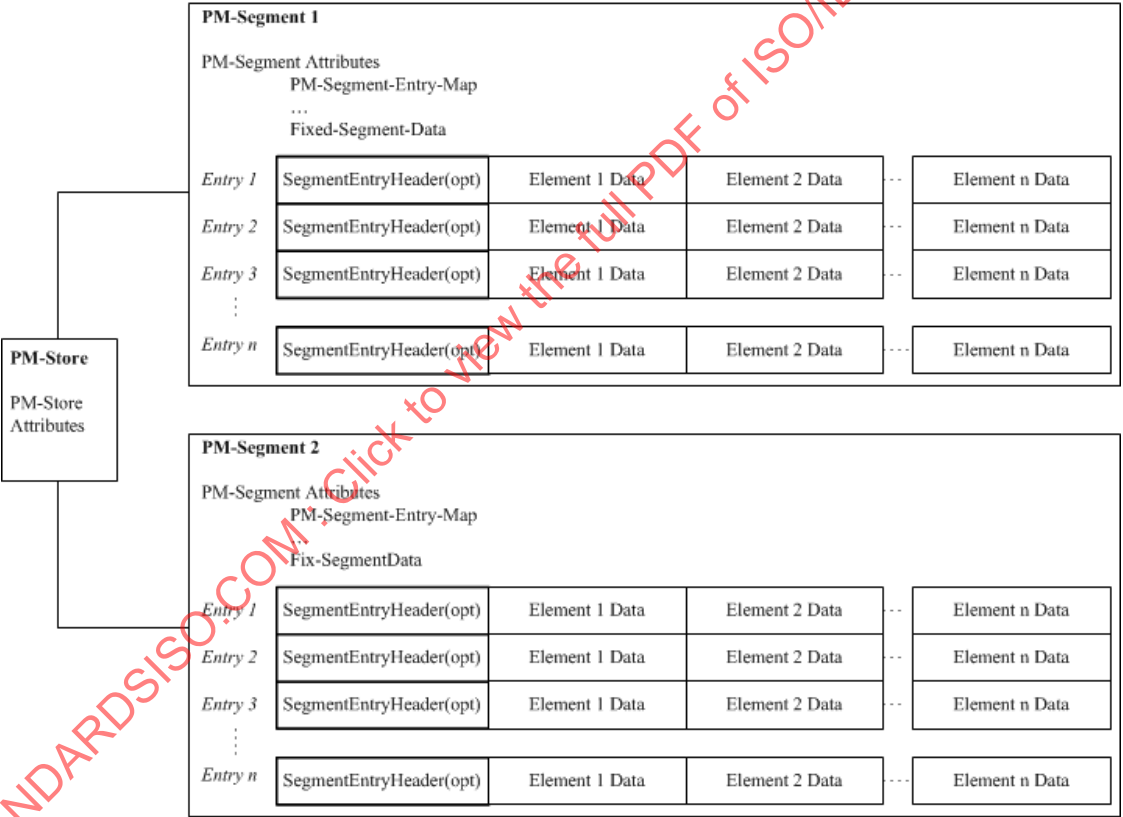


Figure C.1—PM-store with 2 PM-segments, fixed-segment-data within the segments

C.2.2 PM-store object

Support for the PM-store object is optional. Only agents that wish to store PMs 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 (e.g., data stored in this segment is from 12:00 to 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).

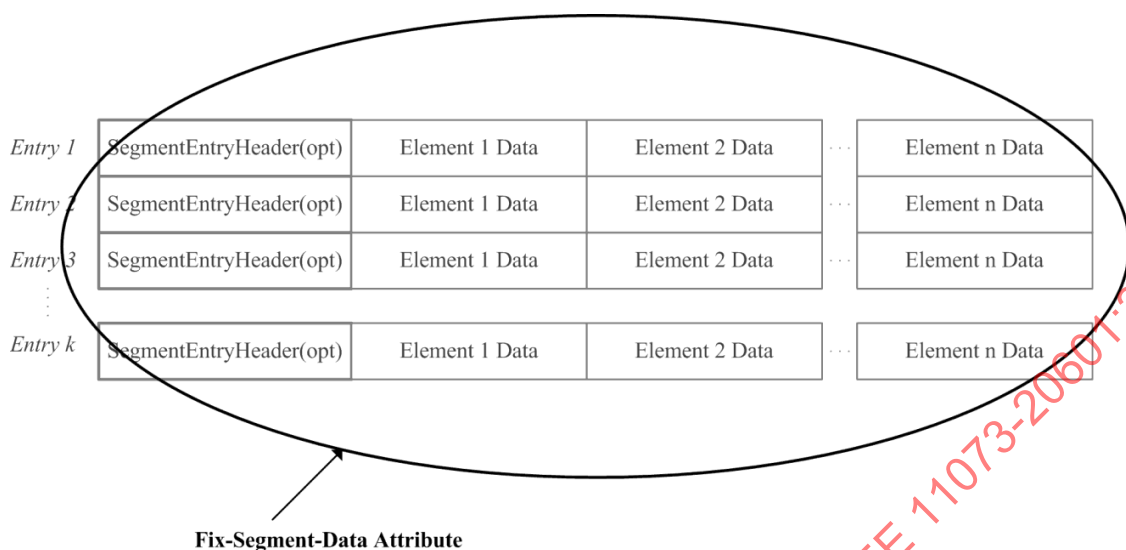


Figure C.3—Fixed-segment-data attribute containing actual stored data

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.

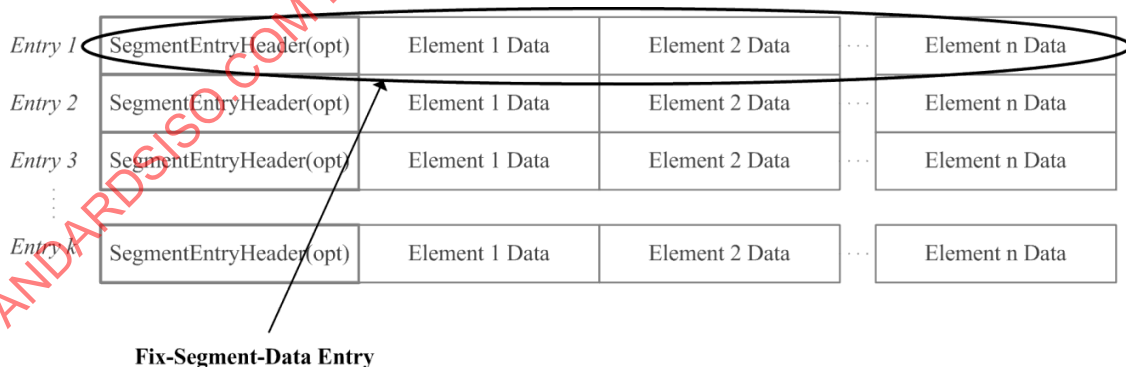


Figure C.4—Entry (array element in fixed-segment-data)

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.

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

Fix-Segment-Data 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 is 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 is 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 maintains 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 are 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 supports 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 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 could 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 could 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

The intended audience for Table E.1 is primarily users who maintain the standards and seek for consistent use of state numbers in Table E.3 and Table E.4.

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	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 Configuring Waiting GetMDS	Y	
9	Connected Associated Configuring Sending GetMDS		Y
10	Connected Associated Configuring Waiting SetTime	Y	
11	Connected Associated Configuring Sending SetTime		Y
12	Connected Associated Operating	Y	Y
13	Connected Disassociating	Y	Y

E.2 Events

The intended audience for Table E.2 is primarily users who maintain the standards and seek for consistent use of state numbers in Table E.3 and Table E.4.

All the events of the agent and manager are defined in Table E.2.

Event table 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 (API)
- Rx A PDU that has arrived on the input data stream

Table E.2—Events

Event Number	Event
1	IND Transport connection
2	IND Transport disconnect
3	IND Timeout and maximum retry not reached
4	IND Timeout and maximum retry reached
5	REQ Assoc
6	REQ Assoc Rel
7	REQ Assoc Abort
8	Rx aarq (*)
9	REQ acceptable and known configuration
10	REQ acceptable and unknown configuration
11	REQ unacceptable configuration
12	Rx aare (*)
13	Rx aare(accepted)
14	Rx aare(accepted-unknown-config)
15	Rx aare(rejected-*)
16	Rx rlrq
17	Rx rlre
18	Rx abrt
19	Rx apdu(*) Any APDU received that is not explicitly covered for this state (e.g., corrupt, unknown, unexpected)
20	REQ ConfigEventReport available
21	Rx roiv-*
22	Rx roiv-cmip-get, handle = 0
23	Rx roiv-* but not (roiv-cmip-get, handle = 0)
24	Rx roiv-confirmed-event-report
25	Rx roiv-* but not (roiv-confirmed-event-report)
26	Rx (rors-*, roer-*, or rorj-*)
27	Rx rors-cmip-confirmed-event-report (unsupported-config) and additional configurations available
28	Rx rors-cmip-confirmed-event-report (unsupported-config) and no additional configurations available
29	Rx rors-cmip-confirmed-event-report (accepted-config)
30	Rx (rors-*, roer-*, or rorj-*), but not Rx: rors-cmip-confirmed-event-report
31	REQ Unsupported-configuration asserted
32	REQ Supported-configuration asserted
33	IND App: ConfigEventReport available
34	REQ roiv-cmip-confirmed-*
35	Rx roiv-cmip-confirmed-action (set time)
36	Rx roiv-cmip-confirmed-action (but not set time)
37	Rx rors-cmip-confirmed-action (set time)
38	Reserved for deprecated event. No longer in use.
39	REQ roiv-cmip-get, handle=0
40	REQ roiv-*, but not roiv-cmip-get, handle=0
41	Rx rors-cmip-get, handle=0
42	IND Timeout Rx roiv-cmip-get, handle = 0
43	IND Timeout Rx roiv-cmip-confirmed-action (set time)
44	Rx roiv-confirmed-event-report containing a configuration
45	Rx roiv-* but not roiv-confirmed-event-report containing a configuration

E.3 Agent state table

The agent state machine shall be implemented as described in Table E.3.

Agent state table 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 sent on the output data stream
Signal ID – x.y	state.event where x is given in Table E.1 and Table E.2

All timeout values indicate the length of time to wait prior to asserting an “IND Timeout” condition.

Table E.3—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
2.5	Connected Unassociated	REQ Assoc	Connected Associating	Timeout=TO _{assoc} , 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>	Can be used to synchronize state on both sides.	Tx abrt(reason undefined)
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(reason undefined)
2.16	Connected Unassociated	Rx rlrq	Connected Unassociated <no state transition>	Should not happen.	Tx abrt(reason undefined)
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

(Table continues)

Table E.3—Agent state table (*continued*)

Signal ID	Initial state	Event/input stream	Next state	Semantic behaviors/notes	Tx stream (output)/event output
2.19	Connected Unassociated	Rx apdu(*) Any APDU not covered in 2.* (e.g., corrupt, unknown, unexpected)	Connected Unassociated <no state transition>	Should not happen.	Tx abrt(reason undefined)
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>	Timeout=TO _{assoc} , increment retry count.	Tx aarq
3.4	Connected Associating	IND Timeout and maximum retry reached	Connected Unassociated	None.	Tx abrt(reason response-timeout)
3.6	Connected Associating	REQ AssocRel	Connected Unassociated	None.	Tx abrt(reason undefined)
3.7	Connected Associating	REQ AssocAbort	Connected Unassociated	None.	Tx abrt reason defined by application
3.8	Connected Associating	Rx aarq(*)	Connected Unassociated	Agent-agent association.	Tx aare(rejected-permanent)
3.13	Connected Associating	Rx aare(accepted)	Connected Associated Configuring Waiting GetMDS	This causes a direct transition to Configuring/Waiting GetMDS 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
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(reason undefined)
3.17	Connected Associating	Rx rlre	Connected Unassociated	Should not happen.	Tx abrt(reason undefined)
3.18	Connected Associating	Rx abrt	Connected Unassociated	None.	None
3.19	Connected Associating	Rx apdu(*) Any APDU not covered in 3.* (e.g., corrupt, unknown, unexpected)	Connected Unassociated	Should not happen.	Tx abrt(reason undefined)

(Table continues)

Table E.3—Agent state table (*continued*)

Signal ID	Initial state	Event/input stream	Next state	Semantic behaviors/notes	Tx stream (output)/event output
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(reason response- timeout)
4.6	Connected Associated Configuring Sending Config	REQ AssocRel (*)	Connected Disassociating	Software requests association release. Timeout=TO _{release} .	Tx rlrq(*)
4.7	Connected Associated Configuring Sending Config	REQ AssocAbort	Connected Unassociated	Software abort.	Tx abrt reason defined by application
4.8	Connected Associated Configuring Sending Config	Rx aarq(*)	Connected Unassociated	Should not happen.	Tx abrt(reason undefined)
4.12	Connected Associated Configuring Sending Config	Rx aare	Connected Unassociated	Should not happen.	Tx abrt(reason undefined)
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(reason undefined)
4.18	Connected Associated Configuring Sending Config	Rx abrt	Connected Unassociated	None.	None
4.19	Connected Associated Configuring Sending Config	Rx apdu(*) Any APDU not covered in 4.* (e.g., corrupt, unknown, unexpected)	Connected Unassociated	Should not happen.	TX abrt(reason undefined)
4.22	Connected Associated Configuring Sending Config	Rx roiv- cmip-get, handle=0	Connected Associated Configuring Sending Config <no state transition>	Not allowed until Configuring Waiting GetMDS state is reached.	Tx roer (no- such-object- instance)
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)

(Table continues)

Table E.3—Agent state table (*continued*)

Signal ID	Initial state	Event/input stream	Next state	Semantic behaviors/notes	Tx stream (output)/event output
4.26	Connected Associated Configuring Sending Config	Rx (rors-*, roer, or rorj)	Connected unassociated	Should not happen.	Tx abrt(reason undefined)
4.32	Connected Associated Configuring Sending Config	REQ Send(Config Report)	Connected Associated Configuring Waiting Approval	The agent has a configuration that it has not yet tried with the manager. Timeout=TO _{config} .	Tx EventReport(C onfigReport)
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(reason configuration- timeout)
5.6	Connected Associated Configuring Waiting Approval	REQ AssocRel(*)	Connected Disassociating	Software request association release. Timeout=TO _{release} .	Tx rlrq(*)
5.7	Connected Associated Configuring Waiting Approval	REQ AssocAbort	Connected Unassociated	Software abort.	Tx abrt reason defined by application
5.8	Connected Associated Configuring Waiting Approval	Rx aarq(*)	Connected Unassociated	Should not happen.	Tx abrt(reason undefined)
5.12	Connected Associated Configuring Waiting Approval	Rx aare(*)	Connected Unassociated	Should not happen.	Tx abrt(reason undefined)
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(reason undefined)
5.18	Connected Associated Configuring Waiting Approval	Rx abrt	Connected Unassociated	None.	None
5.19	Connected Associated Configuring Waiting Approval	Rx apdu(*) Any APDU not covered in 5.* (e.g., corrupt, unknown, unexpected)	Connected Unassociated	Should not happen.	TX abrt(reason undefined)

(Table continues)

Table E.3—Agent state table (*continued*)

Signal ID	Initial state	Event/input stream	Next state	Semantic behaviors/notes	Tx stream (output)/event output
5.22	Connected Associated Configuring Waiting Approval	Rx roiv-cmip-get, handle=0	Connected Associated Configuring Waiting Approval <no state transition>	Not allowed until Configuring Waiting GetMDS state is reached.	Tx roer (no-such-object-instance)
5.23	Connected Associated Configuring Waiting Approval	Rx roiv-* but not (roiv-cmip-get, handle=0)	Connected Associated Configuring Waiting Approval <no state transition>	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) and additional configurations available	Connected Associated Configuring Sending Config	Manager has rejected the configuration and additional configurations are available.	None
5.28	Connected Associated Configuring Waiting Approval	RX rors-cmip-confirmed-event-report (unsupported -config) and no additional configurations available	Connected Disassociating	Manager has rejected the configuration and additional configurations are not available.	TX rlrq(reason no more-configs)
5.29	Connected Associated Configuring Waiting Approval	Rx rors-cmip-confirmed-event-report (accepted-config)	Connected Associated Configuring Waiting GetMDS	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(reason undefined)
5.35	Connected Associated Configuring Waiting Approval	Rx roiv-cmip-confirmed-action (set time)	Connected Associated Configuring Waiting Approval <no state transition>	Not allowed until Configuring Waiting SetTime state is reached.	Tx roer (no-such-object-instance)
5.36	Connected Associated Configuring Waiting Approval	Rx roiv-cmip-confirmed-action (but not set time)	Connected Associated Configuring Waiting Approval <no state transition>	Not allowed until Operating state is reached.	Tx roer (no-such-object-instance)

(Table continues)

Table E.3—Agent state table (*continued*)

Signal ID	Initial state	Event/input stream	Next state	Semantic behaviors/notes	Tx stream (output)/event output
8.2	Connected Associated Configuring Waiting GetMDS	IND Transport Disconnect	Disconnected	None.	None
8.4	Connected Associated Configuring Waiting GetMDS	IND Timeout	Connected Unassociated	No reply. The roiv-cmip-get, handle=0 is not received. The manager shall send “roiv-cmip-get, handle=0” within a TO_{get} period after entering state 9.	Tx abrt(reason response-timeout)
8.6	Connected Associated Configuring Waiting GetMDS	REQ AssocRel(*)	Connected Disassociating	Software request association release. Timeout= $TO_{release}$.	Tx rlrq(*)
8.7	Connected Associated Configuring Waiting GetMDS	REQ AssocAbort	Connected Unassociated	Software abort.	Tx abrt reason defined by application
8.16	Connected Associated Configuring Waiting GetMDS	Rx rlrq	Connected Unassociated	None.	Tx rlre(normal)
8.17	Connected Associated Configuring Waiting GetMDS	Rx rlre	Connected Unassociated	Should not happen.	Tx abrt(reason undefined)
8.18	Connected Associated Configuring Waiting GetMDS	Rx abrt	Connected Unassociated	None.	None
8.19	Connected Associated Configuring Waiting GetMDS	Rx apdu(*) Any APDU not covered in 8 * (e.g., corrupt, unknown, unexpected)	Connected Unassociated	Should not happen.	TX abrt(reason undefined)
8.22	Connected Associated Configuring Waiting GetMDS	Rx roiv-cmip-get, handle=0	Connected Associated Configuring Waiting SetTime (if mds-time-mgr-set-time bit is set) Or Connected Associated Operating (if mds-time-mgr-set-time bit is not set)	Manager probes MDS. See 6.3.2.6.1.	rors-cmip-get, handle=0

(Table continues)

Table E.3—Agent state table (*continued*)

Signal ID	Initial state	Event/input stream	Next state	Semantic behaviors/notes	Tx stream (output)/event output
8.23	Connected Associated Configuring Waiting GetMDS	Rx roiv-* but not (roiv-cmip-get, handle=0)	Connected Associated Configuring Waiting GetMDS <no state transition>	Not allowed until Configuring Waiting SetTime state or Operating state is reached.	Tx roer (no-such-object-instance)
10.2	Connected Associated Configuring Waiting SetTime	IND Transport Disconnect	Disconnected	None.	None
10.4	Connected Associated Configuring Waiting SetTime	IND Timeout	Connected Unassociated	No reply. The Set-Time (or Set-Base-Offset-Time) action command is not received. If the mds-time-mgr-set-time bit is set, the manager shall issue Set-Time (or Set-Base-Offset-Time) action within a TO_{set} period after receiving rors-cmip-get, handle=0.	Tx abrt(reason response-timeout)
10.6	Connected Associated Configuring Waiting SetTime	REQ AssocRel(*)	Connected Disassociating	Software request association release. Timeout= $TO_{release}$.	Tx rlrq(*)
10.7	Connected Associated Configuring Waiting SetTime	REQ AssocAbort	Connected Unassociated	Software abort.	Tx abrt reason defined by application
10.16	Connected Associated Configuring Waiting SetTime	Rx rlrq	Connected Unassociated	None.	Tx rlre(normal)
10.17	Connected Associated Configuring Waiting SetTime	Rx rlre	Connected Unassociated	Should not happen.	Tx abrt(reason undefined)
10.18	Connected Associated Configuring Waiting SetTime	Rx abrt	Connected Unassociated	None.	None
10.19	Connected Associated Configuring Waiting SetTime	Rx apdu(*) Any APDU not covered in 10.* (e.g., corrupt, unknown, unexpected)	Connected Unassociated	Should not happen.	TX abrt(reason undefined)
10.35	Connected Associated Configuring Waiting SetTime	Rx roiv-cmip-confirmed-action (set time)	Connected Associated Operating	Manager sets the local time on agent. See 8.8.3 and 8.12.2.1 for more details.	rors-cmip-confirmed-action (set time)

(Table continues)

Table E.3—Agent state table (*continued*)

Signal ID	Initial state	Event/input stream	Next state	Semantic behaviors/notes	Tx stream (output)/event output
10.36	Connected Associated Configuring Waiting SetTime	Rx roiv-cmip-confirmed-action (but not set time)	Connected Associated Configuring Waiting SetTime<no state transition>	Not allowed until Operating state is reached.	Tx roer (no-such-object-instance)
12.2	Connected Associated Operating	IND Transport Disconnect	Disconnected	None.	None
12.4	Connected Associated Operating	IND Timeout	Connected Unassociated	No reply.	Tx abrt(reason-response-timeout)
12.6	Connected Associated Operating	REQ AssocRel	Connected Disassociating	None. Timeout=TO _{release} .	Tx rlrq(normal) ¹⁵
12.7	Connected Associated Operating	REQ AssocAbort	Connected Unassociated	None.	Tx abrt reason defined by application
12.8	Connected Associated Operating	Rx aarq(*)	Connected Unassociated	Should not happen.	Tx abrt(reason undefined)
12.12	Connected Associated Operating	Rx aare(*)	Connected Unassociated	Should not happen.	Tx abrt(reason undefined)
12.16	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)
12.17	Connected Associated Operating	Rx rlre	Connected Unassociated	Should not happen.	Tx abrt(reason undefined)
12.18	Connected Associated Operating	Rx abrt	Connected Unassociated	None.	None
12.19	Connected Associated Operating	Rx apdu(*) Any APDU not covered in 12.* (e.g., corrupt, unknown, unexpected)	Connected Unassociated	Should not happen.	Tx abrt(reason undefined)
12.21	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)
12.26	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 ¹⁶

(Table continues)

¹⁵ An AssocRel should not be sent until all outstanding invoke-ids are retired.¹⁶ 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—Agent state table (*continued*)

Signal ID	Initial state	Event/input stream	Next state	Semantic behaviors/notes	Tx stream (output)/event output
13.2	Connected Disassociating	IND Transport Disconnect	Disconnected	None.	None
13.4	Connected Disassociating	IND Timeout	Connected Unassociated	No reply to rlrq.	Tx abrt(reason response- timeout)
13.6	Connected Disassociating	REQ AssocRel	Connected Disassociating	Already disassociating. Ignore.	None
13.7	Connected Disassociating	REQ AssocAbort	Connected Unassociated	Abort the graceful disassociation process.	Tx abrt reason defined by application
13.8	Connected Disassociating	Rx aarq	Connected Unassociated	Should not happen.	Tx abrt(reason undefined)
13.12	Connected Disassociating	Rx aare(*)	Connected Unassociated	Should not happen.	Tx abrt(reason undefined)
13.16	Connected Disassociating	Rx rlrq	Connected Disassociating <no state transition>	Both sides releasing connection. Respond and wait for own rlre.	Tx rlre(normal)
13.17	Connected Disassociating	Rx rlre	Connected Unassociated	Release process completed, exit to unassociated.	None
13.18	Connected Disassociating	Rx abrt	Connected Unassociated	None.	None
13.19	Connected Disassociating	Rx apdu(*) Any APDU not covered in 13.* (e.g., corrupt, unknown, unexpected)	Connected Unassociated	Should not happen.	Tx abrt(reason undefined)
13.21	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
13.26	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(reason undefined)

E.4 Manager state table

The manager state machine shall be implemented as described in Table E.4.

Manager state table 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

Table E.4—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>	Can be used to synchronize state on both sides.	Tx abrt (reason undefined)
2.8	Connected Unassociated	Rx aarq (*)	Connected Associating	Associate request.	None
2.12	Connected Unassociated	Rx aare(*)	Connected Unassociated <no state transition>	Should not happen.	Tx abrt(reason undefined)
2.16	Connected Unassociated	Rx rlrq	Connected Unassociated <no state transition>	Should not happen.	Tx abrt(reason undefined)
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 apdu(*) Any APDU not covered in 2.* (e.g., corrupt, unknown, unexpected)	Connected Unassociated <no state transition>	Should not happen.	Tx abrt(reason undefined)

(Table continues)

Table E.4—Manager state table (*continued*)

Signal ID	Initial state	Event/input stream	Next state	Semantic behaviors/notes	Tx stream (output)/event generated
3.2	Connected Associating	IND Transport disconnect	Disconnected	None.	None
3.6	Connected Associating	REQ AssocRel	Connected Unassociated	None.	Tx abrt(reason undefined)
3.7	Connected Associating	REQ AssocAbort	Connected Unassociated	None.	Tx abrt reason defined by application
3.8	Connected Associating	Rx aarq(*)	Connected Unassociated	Should not happen.	Tx abrt(reason undefined)
3.9	Connected Associating	REQ acceptable and known configuration	Connected Associated Configuring Sending GetMDS	The agent and configuration are known to the manager.	Tx aare(accepted)
3.10	Connected Associating	REQ acceptable with unknown configuration	Connected Associated Configuring Waiting	The manager the association request, but does not have valid configuration information for the agent. Timeout=TO _{config} .	Tx aare(accepted-unknown-config)
3.11	Connected Associating	REQ unacceptable configuration	Connected Unassociated	The manager determines that the association is unacceptable.	Tx aare(reject-*)
3.12	Connected Associating	Rx aare(*)	Connected Unassociated	Should not happen.	Tx abrt(reason undefined)
3.16	Connected Associating	Rx rlrq	Connected Unassociated	Should not happen. The manager has received a request to release the association, but it has not yet established an association.	Tx abrt(reason undefined)
3.17	Connected Associating	Rx rlre	Connected Unassociated	Should not happen.	Tx abrt(reason undefined)
3.18	Connected Associating	Rx abrt	Connected Unassociated	None.	None
3.19	Connected Associating	Rx apdu(*) Any APDU not covered in 3.* (e.g., corrupt, unknown, unexpected)	Connected Unassociated	Should not happen.	Tx abrt(reason undefined)
6.2	Connected Associated Configuring Waiting	IND Transport disconnect	Disconnected	None.	None
6.4	Connected Associated Configuring Waiting	IND Timeout	Connected Unassociated	No reply. Config Report not received.	Tx abrt(reason configuration-timeout)
6.6	Connected Associated Configuring Waiting	REQ AssocRel	Connected Disassociating	None. Timeout=TO _{release} .	Tx rlrq(normal)

(Table continues)

Table E.4—Manager state table (*continued*)

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 reason defined by application
6.8	Connected Associated Configuring Waiting	Rx aarq(*)	Connected Unassociated	Should not happen.	Tx abrt(reason undefined)
6.12	Connected Associated Configuring Waiting	Rx aare(*)	Connected Unassociated	Should not happen.	Tx abrt(reason undefined)
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(reason undefined)
6.18	Connected Associated Configuring Waiting	Rx abrt	Connected Unassociated	None.	None
6.19	Connected Associated Configuring Waiting	Rx apdu(*) Any APDU not covered in 6.* (e.g., corrupt, unknown, unexpected)	Connected Unassociated	Should not happen.	Tx abrt(reason undefined)
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. This is expected to be deprecated in future.	None ¹⁷
6.44	Connected Associated Configuring Waiting	Rx roiv-confirmed-event-report containing a configuration	Connected Associated Configuring Checking Config	Event report containing configuration from agent. Timeout=TO _{config} .	None
6.45	Connected Associated Configuring Waiting	Rx roiv-* but not roiv-confirmed-event-report containing a configuration	Connected Unassociated	Not allowed.	Tx abrt(reason undefined)
7.2	Connected Associated Configuring Checking Config	IND Transport Disconnect	Disconnected	None.	None

(Table continues)

¹⁷ 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.4—Manager state table (*continued*)

Signal ID	Initial state	Event/input stream	Next state	Semantic behaviors/notes	Tx stream (output)/event generated
7.4	Connected Associated Configuring Checking Config	IND Timeout	Connected Unassociated	No reply.	Tx abrt(reason response-timeout)
7.6	Connected Associated Configuring Checking Config	REQ AssocRel	Connected Disassociating	None.	Tx rlrq(normal)
7.7	Connected Associated Configuring Checking Config	REQ AssocAbort	Connected Unassociated	None.	Tx abrt(reason defined by application)
7.8	Connected Associated Configuring Checking Config	Rx aarq(*)	Connected Unassociated	Should not happen.	Tx abrt(reason undefined)
7.12	Connected Associated Configuring Checking Config	Rx aare(*)	Connected Unassociated	Should not happen.	Tx abrt(reason undefined)
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(reason undefined) ¹⁸
7.18	Connected Associated Configuring Checking Config	Rx abrt	Connected Unassociated	None.	None
7.19	Connected Associated Configuring Checking Config	Rx apdu(*) Any APDU not covered in 7.* (e.g., corrupt, unknown, unexpected)	Connected Unassociated	Should not happen.	Tx abrt(reason undefined)
7.24	Connected Associated Configuring Checking Config	Rx roiv-confirmed-event-report	Connected Unassociated	The agent is sending event reports before a configuration is agreed to.	Tx abrt(reason undefined)
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 abrt(reason undefined)

(Table continues)

¹⁸ 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.4—Manager state table (*continued*)

Signal ID	Initial state	Event/input stream	Next state	Semantic behaviors/notes	Tx stream (output)/event generated
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. This is expected to be deprecated in future.	None
7.31	Connected Associated Configuring Checking Config	REQ agent supplied unsupported configuration	Connected Associated Configuring Waiting	Timeout=TO _{config} .	Tx rors-cmip- configuration- event(unsupport ed-config)
7.32	Connected Associated Configuring Checking Config	REQ agent supplied supported configuration	Connected Associated Configuring Sending GetMDS	None.	Tx rors-cmip- configuration- event(supporte d-config)
9.2	Connected Associated Configuring Sending GetMDS	IND Transport disconnect	Disconnected	None.	None
9.4	Connected Associated Configuring Sending GetMDS	IND Timeout	Connected Unassociated	No reply. The rors-cmip-get, handle=0 is not received.	Tx abrt(reason response- timeout)
9.6	Connected Associated Configuring Sending GetMDS	REQ AssocRel	Connected Disassociating	None. Timeout=TO _{release} .	Tx rlrq(normal)
9.7	Connected Associated Configuring Sending GetMDS	REQ AssocAbort	Connected Unassociated	None.	Tx abrt reason defined by application
9.8	Connected Associated Configuring Sending GetMDS	Rx aarg(*)	Connected Unassociated	Should not happen.	Tx abrt(reason undefined)
9.12	Connected Associated Configuring Sending GetMDS	Rx aare(*)	Connected Unassociated	Should not happen.	Tx abrt(reason undefined)
9.16	Connected Associated Configuring Sending GetMDS	Rx rlrq	Connected Unassociated	The manager has received a request to release the association.	Tx rlre(normal)
9.17	Connected Associated Configuring Sending GetMDS	Rx rlre	Connected Unassociated	Should not happen.	Tx abrt(reason undefined)
9.18	Connected Associated Configuring Sending GetMDS	Rx abrt	Connected Unassociated	None.	None

(Table continues)

Table E.4—Manager state table (*continued*)

Signal ID	Initial state	Event/input stream	Next state	Semantic behaviors/notes	Tx stream (output)/event generated
9.19	Connected Associated Configuring Sending GetMDS	Rx apdu(*) Any APDU not covered in 9.* (e.g., corrupt, unknown, unexpected)	Connected Unassociated	Should not happen.	Tx abrt(reason undefined)
9.39	Connected Associated Configuring Sending GetMDS	REQ roiv-cmip-get, handle=0	Connected Associated Configuring Sending GetMDS <no state transition>	On entering the Configuring Sending GetMDS substate, the manager shall issue a GET service command with handle=0 to the agent in order to retrieve the values of all implemented MDS object attributes. See 6.3.2.6.1. The manager shall send “roiv-cmip-get, handle=0” within a TO_{set} period after entering state 9.	Tx roiv-cmip-get, handle=0
9.40	Connected Associated Configuring Sending GetMDS	REQ roiv-*, but not roiv-cmip-get, handle=0	Connected Associated Configuring Sending GetMDS <no state transition>	Not allowed until Configuring Waiting SetTime state or Operating state is reached.	None.
9.41	Connected Associated Configuring Sending GetMDS	Rx rors-cmip-get, handle=0	Connected Associated Configuring Sending SetTime (if mds-time-mgr-set-time bit is set) Or Connected Associated Operating (if mds-time-mgr-set-time bit is not set)	Depending on the value of mds-time-mgr-set-time bit, manager decides whether to set the local time on agent. If the mds-time-mgr-set-time bit is set, the manager shall issue Set-Time (or Set-Base-Offset-Time) action within a TO_{ca} period after receiving rors-cmip-get, handle=0.	Tx roiv-cmip-confirmed-action (set time), if mds-time-mgr-set-time bit is set Or None, if mds-time-mgr-set-time bit is not set
11.2	Connected Associated Configuring Sending SetTime	IND Transport disconnect	Disconnected	None.	None
11.4	Connected Associated Configuring Sending SetTime	IND Timeout	Connected Unassociated	No reply. The rors-cmip-get, handle=0 is not received.	Tx abrt(reason response-timeout)
11.6	Connected Associated Configuring Sending SetTime	REQ AssocRel	Connected Disassociating	None. Timeout= $TO_{release}$.	Tx rlrq(normal)

(Table continues)

Table E.4—Manager state table (*continued*)

Signal ID	Initial state	Event/input stream	Next state	Semantic behaviors/notes	Tx stream (output)/event generated
11.7	Connected Associated Configuring Sending SetTime	REQ AssocAbort	Connected Unassociated	None.	Tx abrt reason defined by application
11.8	Connected Associated Configuring Sending SetTime	Rx aarq(*)	Connected Unassociated	Should not happen.	Tx abrt(reason undefined)
11.12	Connected Associated Configuring Sending SetTime	Rx aare(*)	Connected Unassociated	Should not happen.	Tx abrt(reason undefined)
11.16	Connected Associated Configuring Sending SetTime	Rx rlrq	Connected Unassociated	The manager has received a request to release the association.	Tx rlre(normal)
11.17	Connected Associated Configuring Sending SetTime	Rx rlre	Connected Unassociated	Should not happen.	Tx abrt(reason undefined)
11.18	Connected Associated Configuring Sending SetTime	Rx abrt	Connected Unassociated	None.	None
11.19	Connected Associated Configuring Sending SetTime	Rx apdu(*) Any APDU not covered in 11.* (e.g., corrupt, unknown, unexpected)	Connected Unassociated	Should not happen.	Tx abrt(reason undefined)
11.37	Connected Associated Configuring Sending SetTime	Rx rors-cmip-confirmed-action (set time)	Connected Associated Operating	Manager sets the local time on agent. See 8.8.3 and 8.12.2.1 for more details.	None.
11.38	Connected Associated Configuring Sending SetTime	Rx (rors-*, roer-*, or rorj-*), but not Rx: rors-cmip-confirmed-action (set time)	Connected Associated Configuring Waiting SetTime<no state transition>	Not allowed until Operating state is reached.	None.
12.2	Connected Associated Operating	IND Transport Disconnect	Disconnected	None.	None
12.4	Connected Associated Operating	IND Timeout	Connected Unassociated	No reply.	Tx abrt(reason response-timeout)
12.6	Connected Associated Operating	REQ AssocRel	Connected Disassociating	None.	Tx rlrq(normal) ¹⁹
12.7	Connected Associated Operating	REQ AssocAbort	Connected Unassociated	None.	Tx abrt reason defined by application

(Table continues)

¹⁹ An AssocRel should not be sent until all outstanding invoke-ids are retired.

Table E.4—Manager state table (*continued*)

Signal ID	Initial state	Event/input stream	Next state	Semantic behaviors/notes	Tx stream (output)/event generated
12.8	Connected Associated Operating	Rx aarq(*)	Connected Unassociated	Should not happen.	Tx abrt(reason undefined)
12.12	Connected Associated Operating	Rx aare(*)	Connected Unassociated	Should not happen.	Tx abrt(reason undefined)
12.16	Connected Associated Operating	Rx rlrq	Connected Unassociated	None.	Tx rlre(normal)
12.17	Connected Associated Operating	Rx rlre	Connected Unassociated	Should not happen.	Tx abrt(reason undefined)
12.18	Connected Associated Operating	Rx abrt	Connected Unassociated	None.	None
12.19	Connected Associated Operating	Rx apdu(*) Any APDU not covered in 12.* (e.g., corrupt, unknown, unexpected)	Connected Unassociated	Should not happen.	Tx abrt(reason undefined)
12.26	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 ²⁰
12.44	Connected Associated Operating	Rx roiv-confirmed-event-report containing a configuration	Connected Unassociated	Not allowed.	Tx abrt(reason undefined)
12.45	Connected Associated Operating	Rx roiv-* but not roiv-confirmed-event-report containing a configuration	Connected Associated Operating <no state transition>	Normal processing of messages. This is the normal Operating state.	Tx (rors-*, or roer, or rorj)
13.2	Connected Disassociating	IND Transport Disconnect	Disconnected	None.	None
13.4	Connected Disassociating	IND Timeout	Connected Unassociated	No reply.	Tx abrt(reason response-timeout)
13.6	Connected Disassociating	REQ AssocRel	Connected Disassociating <no state transition>	Already disassociating. Ignore.	None
13.7	Connected Disassociating	REQ AssocAbort	Connected Unassociated	Abort the graceful disassociation process.	Tx abrt reason defined by application
13.8	Connected Disassociating	Rx aarq(*)	Connected Unassociated	Should not happen.	Tx abrt(reason undefined)

(Table continues)

²⁰ 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.4—Manager state table (*continued*)

Signal ID	Initial state	Event/input stream	Next state	Semantic behaviors/notes	Tx stream (output)/event generated
13.12	Connected Disassociating	Rx aare(*)	Connected Unassociated	Should not happen.	Tx abrt(reason undefined)
13.16	Connected Disassociating	Rx rlrq	Connected Disassociating <no state transition>	Both sides releasing connection. Wait for own rlre.	Tx rlre(normal)
13.17	Connected Disassociating	Rx rlre	Connected Unassociated	Release process completed. Exit to unassociated.	None
13.18	Connected Disassociating	Rx abrt	Connected Unassociated	None.	None
13.19	Connected Disassociating	Rx apdu(*) Any APDU not covered in 13.* (e.g., corrupt, unknown, unexpected)	Connected Unassociated	Should not happen.	Tx abrt(reason undefined)
13.21	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
13.26	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(reason undefined) ²¹

NOTE—If an abort is sent by an agent, then the following step is highly recommended: the agent should include a delay sufficient for any outstanding confirmation messages from the manager to clear before a subsequent association request is sent.

²¹ 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 [B21], 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 and 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.

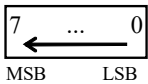
F.3 Byte order

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.

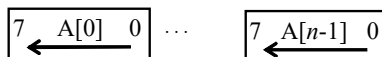
- NBO

- 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:



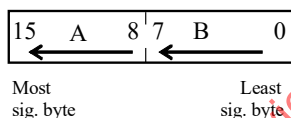
- Multibyte string

- 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 $\langle n \rangle$ = length in octets.

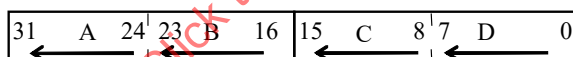


- Structured: a multibyte ordering of bits, typically in multiples of two (e.g., a short integer is 16 bits, a long integer is 32 bits); floating point numbers in general are multiples of 16 bits, although in this standard, only a 32 bit FLOAT is defined. Two generic examples are given (ABCD refers to byte order):

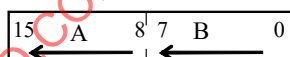
- 16 bit structure, e.g., short (integer)
 - Bit sequence: each byte transferred as defined above for octet
 - Byte sequence: transferred in order of most significant byte to least significant byte
 - For signed integers, typically the MSB of the most significant byte is the sign (s) bit.



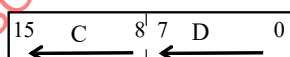
- 32 bit structure, e.g., long (integer)



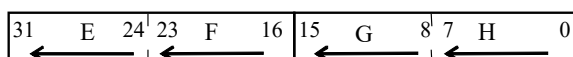
- By convention, multistructure compositions are shown in order of appearance in a serialized string, e.g.,



First in sequence



Next in sequence



Last in sequence

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 (65 536) 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.²²

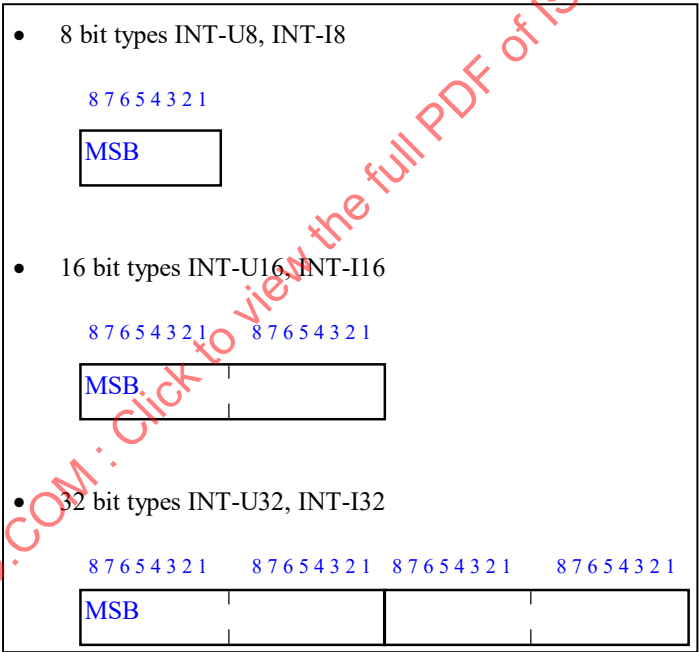


Figure F.2—Integer encodings

The octets contain the twos-complement representation of the encoded integer value.

²² To promote C programming language standardization for these integer data types, ISO/IEC 9899 [B16] 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.

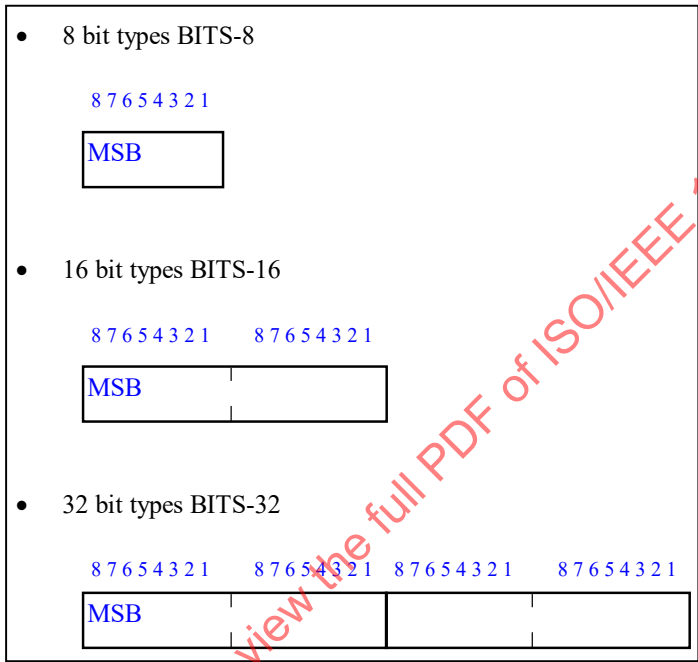


Figure F.3—BIT STRING encodings

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:

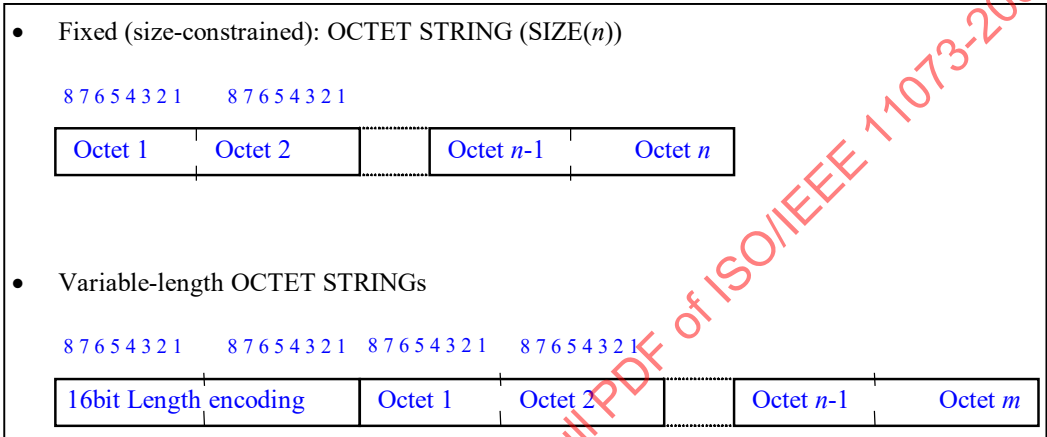


Figure F.4—OCTET STRING encodings

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.

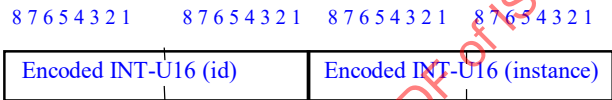


Figure F.5—Sample encoding of a SEQUENCE

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 the sum of the length of all the *n* encoded elements. 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.

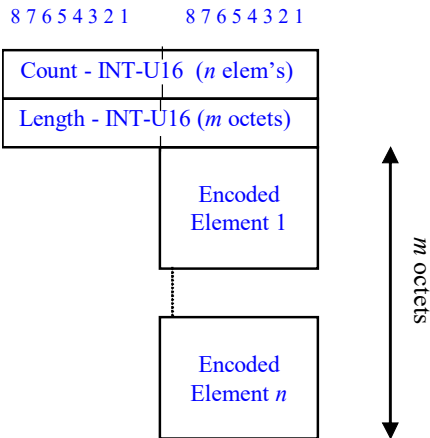


Figure F.6—Encoding of SEQUENCE OF

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.

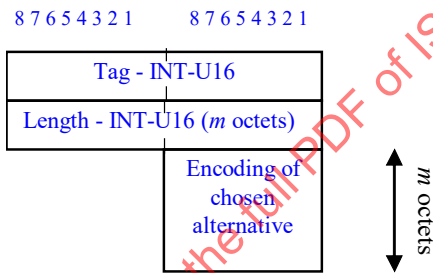


Figure F.7—Encoding of CHOICE

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.
- The abstract syntax for implicit tags does not include an explicit choice number and, therefore, requires a rule for assigning choice_id field values. For implicit tags, choice_id field values shall start with the value 1 and are sequential in order of the abstract syntax choices. In the example above, the choice_id field values for one_type_chosen and two_type_chosen fields are 1 and 2, respectively.
- The abstract syntax for explicit tags includes an explicit choice number, which is mapped directly to the choice_id field in the encoding rule just defined. In this case, choices are sequential, but disjoint, depending on the application, as in the following example:

```
choice-type ::= CHOICE {
    one           [1] OneType,    -- defines tag value 1 in MDER
    four          [4] FourType   -- defines tag value 4 in MDER
}
```

F.4.8 ANY DEFINED BY and instance-of

The ANY DEFINED BY type (ASN.1 1988/90) or the instance-of type (ASN.1 1994) is encoded by a header of a length field to specify the number of octets in the encoding of the selected value that follows. See Figure F.8.

The type specified refers to embedded syntaxes that are specified using a registered OID. Refer to Annex H of ISO/IEEE 11073-20101:2004 [B21] for compatibility cases.

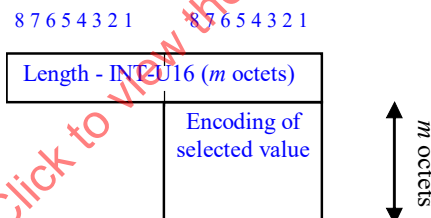


Figure F.8—Encoding of ANY DEFINED BY (instance-of)

Example:

The following definitions

```
TestType ::= SEQUENCE {
    type-id          OIDType,
    value            ANY DEFINED BY type-id
}
```

are mapped to a C language type representation as follows:

```
typedef struct {
    OIDType          type-id,
    unsigned short   any_length;
    char             any_data;    /* placeholder for encoded data type */
} TestType;
```

This example shows the byte encoding of the SEQUENCE containing a context-sensitive OID and the value of an ANY DEFINED BY.

In the preceding mapping, the type-id field is a context-free OID. An application uses the ID field to cast the any_data field to the right data type. The character data type for the any_data field is essentially meaningless and provides the address of the field only. Note that length can be 0, which means the any_data field does not exist.

The instance-of type encodes the ASN.1 TYPE-IDENTIFIER construct and is identical to the ANY DEFINED BY encoding for the purpose of backwards-compatibility.

F.5 Floating point numbers

The restricted subset of ASN.1 for MDER does not contain the ASN.1 data type FLOAT, and instead this standard defines its own floating point types, FLOAT-Type and SFLOAT-Type.

F.6 Floating point data structure—FLOAT-Type

The FLOAT-Type is mapped as a 32-bit structure, formatted according to the medical device numeric format (MDNF).

MDNF is a 32-bit word comprising a signed 8-bit integer exponent followed by a signed 24-bit integer mantissa. See Figure F.9.

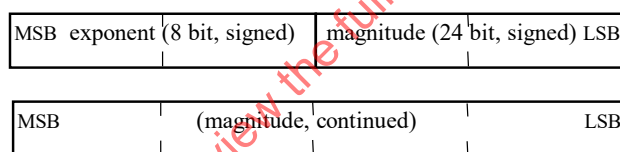


Figure F.9—MDNF encoding

The number represented is $(\text{mantissa}) \times (10^{\text{exponent}})$.²³ Both the exponent and mantissa are in two's-complement form. The mantissa and exponent are adjusted to denote the precision as described in F.8.

The special values that are represented and displayed are in Table F.2.

Table F.2—MDNF special values

Special value	Mantissa	Bit value
NaN (not a number)	$+(2^{**23} - 1)$	0x007FFFFF
NRes (not at this resolution)	$-(2^{**23})$	0x00800000
+ INFINITY	$+(2^{**23} - 2)$	0x007FFFFE
- INFINITY	$-(2^{**23} - 2)$	0x00800002
Reserved for future use	$-(2^{**23} - 1)$	0x00800001

²³ The double asterisk (**) is used to represent the exponent operation.