INTERNATIONAL STANDARD

ISO/IEEE 11073-20101

First edition
2004-12-15

Health informatics — Point-of-care medical device communication —

Part 20101: Application profiles — Base standard

Informatique de santé — Communication entre dispositifs médicaux sur le site des soins —
Partie 20101: Profils d'applications — Norme de base

Reference number
ISO/IEEE 11073-20101:2004(E)

ISO/IEEE 11073-20101:2004(E)

Health informatics — Point-of-care medical device communication —
Part 20101:
Application profiles — Base standard

Sponsor

IEEE 1073™ Standard Committee

of the

IEEE Engineering in Medicine and Biology Society

Approved 24 June 2004

IEEE-SA Standards Board

Abstract: The scope of this standard is upper layer [i.e., the International Organization for Standardization (ISO's) open systems interconnection (OSI) application, presentation layer, and session layer] services and protocols for information exchange under the ISO/IEEE 11073 standards for medical device communications (MDC). This standard is the base standard of the ISO/IEEE 11073-20000 medical device application profiles (MDAP), as harmonized through the Committee for European Normalization (CEN) and the ISO.

Keywords: abstract syntax, alarm, alert, communication, control, information model, medical device, object-oriented, point-of-care, POC, services

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

Institute of Electrical and Electronics Engineers
Standards Association
Manager, Standards Intellectual Property
445 Hoes Lane
Piscataway, NJ 08854
E-mail: stds.ipr@ieee.org
Web: www.ieee.org

Published 15 December 2004. Printed in the United States of America.

Print: ISBN 0-7381-4091-0 SH95257
PDF: ISBN 0-7381-4092-9 SS95257

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.

Use of an IEEE Standard is wholly voluntary. The IEEE disclaims liability for any personal injury, property or other damage, of any nature whatsoever, whether special, indirect, consequential, or compensatory, directly or indirectly resulting from the publication, use of, or reliance upon this, or any other IEEE Standard document.

The IEEE does not warrant or represent the accuracy or content of the material contained herein, and expressly disclaims any express or implied warranty, including any implied warranty of merchantability or fitness for a specific purpose, or that the use of the material contained herein is free from patent infringement. IEEE Standards documents are supplied "**AS IS**."

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. Every IEEE Standard is subjected to review at least every five years for revision or reaffirmation. When a document is more than five years old and has not been reaffirmed, 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 publishing and making this document available, the IEEE is not suggesting or rendering professional or other services for, or on behalf of, any person or entity. Nor is the IEEE undertaking to perform any duty owed by any other person or entity to another. Any person utilizing this, and any other IEEE Standards document, should rely upon the advice of a competent professional in determining the exercise of reasonable care in any given circumstances.

Interpretations: Occasionally questions may arise regarding the meaning of portions of standards as they relate to specific applications. When the need for interpretations is brought to the attention of IEEE, the Institute will initiate action to prepare appropriate responses. Since IEEE Standards represent a consensus of concerned interests, it is important to ensure that any interpretation has also received 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 interpretation requests except in those cases where the matter has previously received formal consideration. 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, explanation, or interpretation of the IEEE.

Comments for revision of IEEE Standards are welcome from any interested party, regardless of membership affiliation with IEEE. Suggestions for changes in documents should be in the form of a proposed change of text, together with appropriate supporting comments. Comments on standards and requests for interpretations should be addressed to:

Secretary, IEEE-SA Standards Board

445 Hoes Lane

Piscataway, NJ 08854 USA

NOTE — 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 with respect to the existence or validity of any patent rights in connection therewith. The IEEE shall not be responsible for identifying patents for which a license may be required by an IEEE standard or for conducting inquiries into the legal validity or scope of those patents that are brought to its attention.

Authorization to photocopy portions of any individual standard for internal or personal use is granted by the Institute of Electrical and Electronics Engineers, Inc., provided that the appropriate fee is paid to Copyright Clearance Center. To arrange for payment of licensing fee, 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.

ISO 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.

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

A pilot project between ISO and the IEEE has been formed to develop and maintain a group of ISO/IEEE standards in the field of medical devices as approved by Council resolution 43/2000. Under this pilot project, IEEE is responsible for the development and maintenance of these standards with participation and input from ISO member bodies.

Attention is drawn to the possibility that some of the elements of this International Standard may be the subject of patent rights. Neither ISO nor the IEEE shall be held responsible for identifying any or all such patent rights.

ISO/IEEE 11073-20101:2004(E) was prepared by IEEE 1073 Committee of the IEEE Engineering in Medicine and Biology Society.

IEEE Introduction

This introduction is not part of ISO/IEEE 11073-20101:2004(E), Health informatics — Point-of-care medical device communication — Part 20101: Application profiles — Base standard.

ISO/IEEE 11073 standards enable communication between medical devices and external computer systems. They provide automatic and detailed electronic data capture of patient vital signs information and device operational data. The primary goals are to:

— Provide real-time plug-and-play interoperability for patient-connected medical devices

— Facilitate the efficient exchange of vital signs and medical device data, acquired at the point-of-care, in all health care environments

"Real-time" means that data from multiple devices can be retrieved, time correlated, and displayed or processed in fractions of a second. "Plug-and-play" means that all the clinician has to do is make the connection — the systems automatically detect, configure, and communicate without any other human interaction.

"Efficient exchange of medical device data" means that information that is captured at the point-of-care (e.g., patient vital signs data) can be archived, retrieved, and processed by many different types of applications without extensive software and equipment support, and without needless loss of information. The standards are especially targeted at acute and continuing care devices, such as patient monitors, ventilators, infusion pumps, ECG devices, etc. They comprise a family of standards that can be layered together to provide connectivity optimized for the specific devices being interfaced.

Notice to users

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 with respect to the existence or validity of any patent rights in connection therewith. The IEEE shall not be responsible for identifying patents or patent applications for which a license may be required by to implement an IEEE standard or for conducting inquiries into the legal validity or scope of those patents that are brought to its attention.

Errata

Errata, if any, for this and all other standards can be accessed at the following URL: http://standards.ieee.org/reading/ieee/updates/errata/index.html. Users are encouraged to check this URL for errata periodically.

Interpretations

Current interpretations can be accessed at the following URL: http://standards.ieee.org/reading/ieee/interp/index.html.

Participants

At the time this standard was completed, the working group of the IEEE 1073 Standard Committee had the following membership:

Todd H. Cooper, *Chair*

Wolfgang Bleicher
Francis Cantraine
Todd H. Cooper
Michael Flötotto
Ken Fuchs
Kai Hassing
Gunther Hellmann
Ron Kirkham
Michael Krämer
Alberto Macerata
Simon Meij
Angelo Rossi Mori
Thomas Norgall
Thomas Penzel
Francesco Pinciroli
Rick Revello
Melvin Reynolds
Lief Rystrøm
Michael Spicer
Lars Steubesand
Andrew Sutton
Alpo Värri
Jan Wittenber
Paul Woolman
Christoph Zywietz

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

Thomas Canup
Michael Chilbert
Todd H. Cooper
James (Bob) Davis
George Economakos
Grace Esche
Kenneth Fuchs
Kai Hassing
Tom Kannally
Robert Kennelly
Randy Krohn
Yeou-Song Lee
Daniel Nowicki
Melvin Reynolds
Ricardo Ruiz-Fernández
Michael Spicer
M. Michael Shabot
Lars Steubesand
Jan Wittenber
Gin-Shu Young

When the IEEE-SA Standards Board approved this standard on 24 June 2004, it had the following membership:

Don Wright, *Chair*
Steve M. Mills, *Vice Chair*
Judith Gorman, *Secretary*

Chuck Adams
H. Stephen Berger
Mark D. Bowman
Joseph A. Bruder
Bob Davis
Roberto de Boisson
Julian Forster*
Arnold M. Greenspan
Mark S. Halpin
Raymond Hapeman
Richard J. Holleman
Richard H. Hulett
Lowell G. Johnson
Joseph L. Koepfinger*
Hermann Koch
Thomas J. McGean
Daleep C. Mohla
Paul Nikolich
T. W. Olsen
Ronald C. Petersen
Gary S. Robinson
Frank Stone
Malcolm V. Thaden
Doug Topping
Joe D. Watson

*Member Emeritus

Also included are the following nonvoting IEEE-SA Standards Board liaisons:

Satish K. Aggarwal, *NRC Representative*
Richard DeBlasio, *DOE Representative*
Alan Cookson, *NIST Representative*

Don Messina
IEEE Standards Project Editor

Contents

Health informatics — Point-of-care medical device communication —
Part 20101:
Application profiles — Base standard

1. Overview

This standard is divided into eight clauses, as follows:

- Clause 1 provides the scope of this standard.
- Clause 2 lists references to other standards that are useful in applying this standard.
- Clause 3 provides definitions and abbreviations.
- Clause 4 provides conventions.
- Clause 5 provides the rationale for this standard.
- Clause 6 provides a communication, i.e., protocol and service, model.
- Clause 7 provides an information, i.e., object, model.
- Clause 8 provides conformance requirements.

This standard also contains nine annexes, as follows:

- Annex A defines the specialized medical device encoding rules (MDER). (normative)
- Annex B describes the allocation of object identifiers. (normative)
- Annex C provides references to time synchronization protocols applied by this standard.
- Annex D includes state transition diagrams as part of the dynamic model.
- Annex E provides abstract syntax, which offers extensions to leveraged standards, such as minimal open systems interconnection (mOSI), that are specific to this standard. (normative)
- Annex F includes examples of a number of protocol data unit (PDU) examples.
- Annex G describes a specialization of Abstract Syntax Notation One (ASN.1).
- Annex H deals with compatibility of ASN.1 between the 1988/90 and 1994 versions.
- Annex I provides a bibliography of useful references.

1.1 Scope

The scope of this standard is upper layer [i.e., the International Organization for Standardization's (ISO’s) open systems interconnection (OSI) application, presentation layer, and session layer] services and protocols for information exchange under the ISO/IEEE 11073 standards for medical device communications (MDC).

This standard is the base standard of the ISO/IEEE 11073-20000 medical device application profiles (MDAP), as harmonized through the Committee for European Normalization (CEN) and the ISO.

1.2 Purpose

The purpose of this standard is to define MDC upper layer application, i.e., ISO A-type profiles for interchange of data, which are defined by the medical device data language (MDDL) format, or ISO F-type profiles (ISO/IEEE 11073-10000 series).

1.3 Goals

The primary goal of MDAP standards is to support MDC upper layer data interchange, based on MDDL, among a wide range, by type and scale, of future and current devices for use in point-of-care (POC) settings in the acute care sections of hospitals.

1.4 Audience

The primary user of the MDAP standards is a software engineer who is creating a MDC system or attempting to establish an interface to one.

Because this family of standards is based largely upon international standardization profiles, familiarity with a range of related standards and technologies is useful if not necessary. The following are recommended as a minimum background:

a) ISO/IEEE 11073 architecture, especially IEEE Std 1073™,[1] ISO/IEEE 11073-10201, and lower layer standards (e.g., ISO/IEEE 11073-30200)
b) ISO’s OSI layered architecture, primarily the upper layers, i.e., application, presentation, and session
c) Systems management
d) Object-oriented analysis and design
e) Machine language theory

2. References

This standard shall be used in conjunction with the following publications. When the following standards are superceded by an approved revision, the revision shall apply.

IEEE Std 1073, IEEE Standard for Medical Device Communications—Overview and Framework.[2]

[1]Information on references can be found in Clause 2.
[2]IEEE publications are available from the Institute of Electrical and Electronics Engineers, Inc., 445 Hoes Lane, Piscataway, NJ 08854, USA (http://standards.ieee.org/).

ISO/IEC 8327-1, Information technology — Open systems interconnection —Connection-oriented session protocol — Part 1: Protocol specification.[3] (same as ITU-T Recommendation X.225)

ISO/IEC 8650-1, Information technology — Open systems interconnection — Connection-oriented protocol for the association control service element — Part 1: Protocol. (same as ITU-T Recommendation X.227)

ISO/IEC 8824-1, Information technology — Abstract Syntax Notation One (ASN.1) — Part 1: Specification of basic notation. (same as ITU-T Recommendation X.680)

ISO/IEC 8824-2, Information technology — Abstract Syntax Notation One (ASN.1) — Part 2: Information object specification. (same as ITU-T Recommendation X.681)

ISO/IEC 8825-1, Information technology — ASN.1 encoding rules — Part 1: Specification of Basic Encoding Rules (BER), Canonical Encoding Rules (CER) and Distinguished Encoding Rules (DER). (same as ITU-T Recommendation X.690)

ISO/IEC 9072-2, Information processing systems — Text communication — Remote operations — Part 2: Protocol specification.

ISO/IEC 9595, Information technology — Open systems interconnection — Common management information service definition.

ISO/IEC 9596-1, Information technology — Open systems interconnecton — Common Management Information Protocol — Part 1: Specification.

ISO/IEC 9899, Programming languages — C.

ISO/IEC ISP 11188-3, Information technology — International standardization profile — Common upper layer requirements — Part 3: Minimal OSI upper layer facilities.

ISO/IEEE 11073-10101, Health informatics — Point-of-care medical device communication — Part 10101: Nomenclature.[4]

ISO/IEEE 11073-10201, Health informatics — Point-of-care medical device communication — Part 10201: Domain information model (referred to hereinafter as “the DIM”).

ISO/IEEE 11073-30200, Health informatics — Point-of-care medical device communication — Part 30200: Transport profile — Cable connected.

ISO/IEEE 11073-30300, Health informatics — Point-of-care medical device communication — Part 30300: Transport profile — Infrared Wireless.

ITU-T Recommendation X.681, Information Technology—Abstract Syntax Notation One (ASN.1)—Information Object Specification. (same as ISO/IEC 8824-2)[5]

[3]ISO/IEC publications are available from the ISO Central Secretariat, Case Postale 56, 1 rue de Varembé, CH-1211, Genève 20, Switzerland/Suisse (http://www.iso.ch/). ISO/IEC publications are also available in the United States from Global Engineering Documents, 15 Inverness Way East, Englewood, CO 80112, USA (http://global.ihs.com/). Electronic copies are available in the United States from the American National Standards Institute, 25 West 43rd Street, 4th Floor, New York, NY 10036, USA (http://www.ansi.org/).

[4]ISO/IEEE publications are available from the ISO Central Secretariat, Case Postale 56, 1 rue de Varembé, CH-1211, Genève 20, Switzerland/Suisse (http://www.iso.ch/); in the United States from the Sales Department, American National Standards Institute, 25 West 43rd Street, 4th Floor, New York, NY 10036, USA (http://www.ansi.org/); and from the Institute of Electrical and Electronics Engineers, Inc., 445 Hoes Lane, Piscataway, NJ 08854, USA (http://standards.ieee.org/).

[5]ITU-T publications are available from the International Telecommunications Union, Place des Nations, CH-1211, Geneva 20, Switzerland/Suisse (http://www.itu.int/).

3. Definitions and abbreviations

3.1 Definitions

For the purposes of this standard, the following terms and definitions apply. IEEE 100, *The Authoritative Dictionary of IEEE Standards Terms and Definitions,* Seventh Edition [B3],[6] should be referenced for terms not defined in this clause.

3.1.1 abstract syntax: Specification of the structure of a data item without reference to or requirement for a specific implementation technology.

3.1.2 big endian: A byte order sequence where the most significant byte is sent first. For example, a 32 bit integer is communicated with the most significant byte (bits 24–31) first and the least significant byte (bits 0–7) last.

3.1.3 byte order: The sequence in which multibyte data primitives are communicated in a protocol data unit (PDU). For example, a 32 bit integer comprises 4 bytes. *See also:* **big endian**.

3.1.4 coalescing: The function of combining multiple presentation-layer protocol data units (PPDUs) into a single session-layer protocol data unit (SPDU) that is then communicated across a transport.

3.1.5 encoding rules: Specification for how data primitives used in an abstract syntax are converted to an implementation format. Generally synonymous with transfer syntax.

3.1.6 linked reply: A command response that requires more than one protocol data unit (PDU) to communicate information. For example, a single command to retrieve an entire log file may result in many linked response PDUs to provide the requested information.[7]

3.1.7 presentation data value (PDV): The union of the sets of values in all possible abstract syntaxes.

3.1.8 transfer syntax: Specification of the structure of data as they are communicated across a transport or physical medium.

3.2 Acronyms and abbreviations

This following acronyms and abbreviations supplement the acronyms and abbreviations of IEEE Std 1073:

AA	(session) abort accept (SPDU)
AARE	association response message
AARQ	association request message
AB	(session) abort (SPDU)
ABRT	abort (APDU)
AC	(session) accept (SPDU)
ACSE	association control service element
AE	application entity
AP	application process
APDU	application protocol data unit

[6]The numbers in brackets correspond to the numbers of the bibliography in Annex I.

[7]A *linked command* definition and service have been omitted from this version of the standard. This command requires multiple PDUs to convey the needed information. For example, a command to send a list of drug names to a device may require multiple linked PDUs. This command may be added to future versions of the standard if required by other profile and optional package standards.

API	application program interface
ARP	abnormal release provider (PPDU)
ARU	abnormal release user (PPDU)
ASN.1	Abstract Syntax Notation One
BCC	bedside communication controller
BER	basic encoding rules
CC	communication controller
CMDIP	Common Medical Device Information Protocol
CMIP	Common Management Information Protocol
CMISE	common management information service element
CMDISE	common medical device information service element
CN	(session) connect (SPDU)
CP	connect presentation (PPDU)
CPA	connect presentation accept (PPDU)
CPR	connect presentation reject (PPDU)
DCC	device communication controller
DICOM	Digital Imaging and Communcations in Medicine
DIF	device interface
DIM	domain information model (see ISO/IEEE 11073-10201.)
DN	(session) disconnect (SPDU)
DT	(session) data transfer (SPDU)
FN	(session) finish (SPDU)
FSM	finite state model or machine
LI	length indicator
LSB	least significant bit
MDAP	medical device application profiles (The acronym *MDAP* may be substituted for the ISO/IEEE 11073-20000 family of standards.)
MMDAP-DT	medical device application profiles normal data transfer (SPDU)
MMDAP-TD	medical device application profiles transmit data (PPDU)
MDAP-XT	medical device application profiles expedited data transfer (SPDU)
MDC	medical device communications or the nomenclature for such communication (ISO-IEEE 11073-10101)
MDCC	medical device communication controller
MDDL	medical device data language (The acronym *MDDL* may be substituted for the ISO/IEEE 11073-10000 family of standards.)
MDER	medical device encoding rules
MDIB	medical data information base
MDNF	medical device numeric format
MDS	medical device system
MDSE	medical device service element
MIB	management information base
mOSI	minimal open systems interconnection
MSB	most significant bit
MTU	maximum transfer unit
NBO	network byte order
OSI	open systems interconnection

PDU	protocol data unit (also referred to as a *message*; however, *PDU* connotes transfer through a transport profile, e.g., ISO/IEEE 11073-30200).
PDV	presentation data value
PER	packed encoding rules
PGI	parameter group identifier
PI	parameter identifier
POC	point of care or point-of-care
PPDU	presentation-layer protocol data unit
QoS	quality of service
RF	(session) refuse (SPDU)
ROER	remote operation error (APDU)
ROIV	remote operation invoke (APDU)
ROLIV	remote operation link invoke (APDU)
RORS	remote operation result (APDU)
ROSE	remote operation service element
SI	SPDU identifier
SNTP	Simple Network Time Protocol
SPDU	session-layer protocol data unit
SS	session service
TD	presentation data (PPDU)
UML	unified modeling language

4. Conventions

Various definitions use prefixed or suffixed symbols to denote specialized or optimized properties. An asterisk (*) is appended to some definitions to indicate specialization. For example, BER* refers to a version of basic encoding rules (BER) that has been significantly optimized for processing efficiency.

5. Rationale

The essential requirement for this standard is to provide a set of abstract and transfer (i.e., encoding rule) syntaxes that are optimized for use by application profiles and implementations of the domain information model (DIM).

5.1 Communication model

The following assumptions and requirements are for the base standard and profiles of it regarding the communication stack and related protocol definitions:

— While the communication stack should be based on already existing standards as far as possible, a primary focus for the protocol definition is on the overall efficiency of implementations (e.g., complexity, resource requirements, bandwidth requirements). It is necessary that even low-end devices be capable of implementing a communication stack based on this work.

— In order to reduce sending and receiving overhead, the headers added by each layer should, therefore, be short and have a fixed data structure without optional or variable size elements.

By avoiding optional components or variable length components in the PDU data type definitions, it is possible for sending devices to use canned messages (in other words, a message template can be filled in memory in which only the actual updated values must be copied). Also, for the receiver, this significantly reduces the complexity of the parser.

This requirement is also strongly related to the requirement for optimized encoding rules.

— The communication stack should be flexible so that other messaging profiles can be accommodated within the same framework.

The protocol stack is defined by PDU (data) type definitions and dynamic behavior only. The normative definition of application programming interfaces (APIs) is outside the scope of this standard; however, non-normative examples may be given to facilitate implementation and reuse.

It is desirable to define a transport-independent interface, although specific mappings of upper layer PDUs to services of an ISO/IEEE 11073-30000 transport profile is addressed through transport-specific sublayers as necessary.

General mechanisms at the transport interface to make provisions for information and behavior related to quality of service (QoS) may be considered.

It is assumed that no complicated session-layer protocol is needed for the communication between medical devices, especially as some error recovery mechanisms are actually defined by application objects defined in MDDL (e.g., scanner objects).

However, to support the standard association control service element (ACSE), a minimal set of ISO/OSI standard session services (SSs) is needed at least during association.

In addition, specific session-layer extensions need to be defined to optimize session-layer processing during normal communication (after association) that can coexist with the session-layer protocol defined in ISO/IEC 8327-1.

These optimizations relate, for example, to

— Simplification of the normal session-layer PDU (SPDU)

— Coalescing of application data in single SPDUs to reduce the message rate at the transport interface and below

5.2 Information model

The following assumptions and requirements regard the communication controller (CC) object model:

— Similar to the objects defined in the MDDL, objects defined in this model represent information that is shared between devices through the communication link. The objects defined here are part of the device medical data information base (MDIB) (directly or indirectly accessible), and they are basically information containers.

— The object model will focus on generalized concepts for the representation of communication interface capabilities (ranging, for example, from link speeds to general QoS-related parameters), interface configuration aspects and statistical data (e.g., for troubleshooting). The model should be independent of specific lower layer implementations, but may contain adaptations specific to IEEE lower layers.

— The medical device communication controller (MDCC) inherits from medical device data language (MDDL) CC, as defined in MDDL.1 (IEEE Std 1073.3.1™ [B5]); definitions may be replicated for clarity and reference convenience in this standard.

— Unified modeling language (UML) is to be used as a notation.

Object attributes and behavior will be defined in a notation that is consistent with the MDDL standard, in particular

- Static views: inheritance, containment, attribution
- Dynamic views
  - The device connection state machine with all message interchanges
  - Object-specific dynamic behavior, e.g., for scanner objects defined in MDDL

Dynamic modeling is necessary to define the actual interactions between communicating medical devices.

In addition, information objects pertaining to management information need to be defined to facilitate support, for example, of configuration, access, performance, and fault tolerance information as defined in ISO/IEEE 11073-30200.

6. Communication model

This clause is intended to include service and protocol definitions.

6.1 General

Figure 1 shows the upper layer communication stack, i.e., the layered set of protocol and service components.

The figure shows the components of the communication stack, as follows:

- ACSE is the ISO/OSI standard for association control.

  A standard association mechanism provides the flexibility to adopt future requirements, e.g., additional format profiles in the upper layers (such as picture formats and Simple Network Time Protocol [SNTP]), different encoding mechanisms.

  It also provides a safe and secure compatibility check and some limited means for option negotiation (e.g., to make sure that both devices use compatible nomenclature versions).
- Common medical device information service element (CMDISE) is the object management service, in principle, a lightweight version of the ISO/OSI common management information service element (CMISE).
- The remote operation service element (ROSE) provides basic services used by the CMDISE (invoke an operation, return the result of an operation, return an error, reject an operation). To comply with the definition of optimized encoding rules, a modified version of the ROSE is needed to work with the CMDISE.
- The session layer and presentation layer produce a minimized overhead only.
- Medical device service element (MDSE) is the overall set of these elements. Encapsulation permits transparency and implementation flexibility in that applications do not have to know about the internal composition of the MDSE, and implementers can choose to integrate specific elements in various ways as long as the interfaces to the application processes and transport system result in a conforming implementation.

Components of the MDIB are defined normatively in the DIM and management information base (MIB) elements documents and are summarized briefly as follows:

- Medical device system (MDS): the highest level containment object representing the device overall.
- CC: a general object from which specializations are defined, as follows:
  - Device CC (DCC): a specialization representing the device CC agent.

  - Bedside CC (BCC): a specialization representing a DCC manager (i.e., host CC).
- Device interface (DIF): an abstraction representing a transport service access point.
- MIB element: an abstraction representing performance, status, or other related information. Specialized MIB elements are particular to a given DIF configuration or implementation.

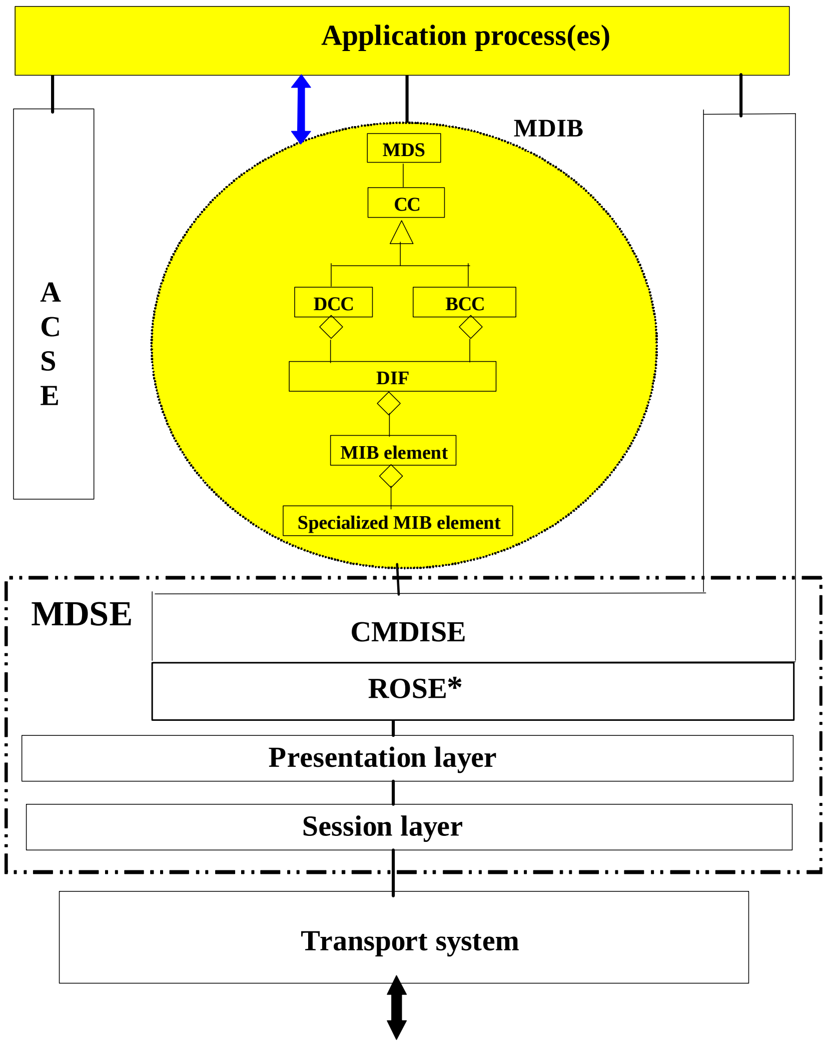


Figure 1—MDC stack

An application message, such as an event report message from a scanner object as defined in MDDL, flows through this communication stack as shown in Figure 2.

The scanner retrieves data from the MDIB and wraps it in a scanner event information field.

After that, each layer (or stack element) copies the data portion and wraps it in its own PDU message, typically by adding some layer-specific header data.

The receiving system reverses this process: Each layer strips the wrapper and header information and passes the result to the next level.

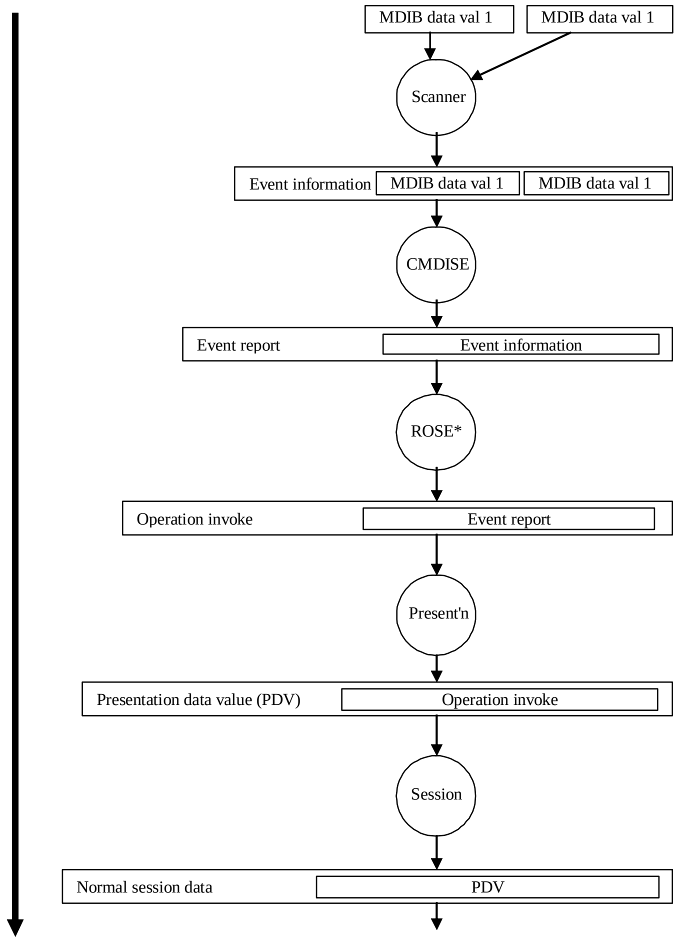


Figure 2—Data flow through the communication stack

6.2 ACSE protocol

6.2.1 General

For association control, it is assumed that the standard ACSE as defined in ISO/IEC 8650-1 is used.

In addition to the standard ACSE, it is necessary to define the set of application-specific user information fields as well as the minimum (and mandatory) set of optional elements in the standard ACSE PDUs.

6.2.2 ACSE services

Table 1 shows the services provided by ACSE.

The services are mapped to messages, i.e., application PDUs (APDUs). For the A-ASSOCIATE services, for example, there are two messages: the association request message (AARQ) and the association response message (AARE).

Table 1—ACSE service summary

Service	Type
A-ASSOCIATE	Confirmed
A-RELEASE	Confirmed
A-ABORT	Nonconfirmed
A-P-ABORT	Provider-initiated

Each service (and thus the resulting APDU) has a number of data fields or parameters. Table 2 and Table 3 show the actual parameters of the AARQ and AARE service calls that are defined in ACSE. The fields are indicated as M for mandatory, O for optional, or U for user optional.

Table 2—AARQ APDU fields

Field name	Presence
Protocol version	O
Application context name	M
Calling application process (AP) title	U
Calling application entity (AE) qualifier	U
Calling AP invocation-identifier	U
Calling AE invocation-identifier	U
Called AP title	U
Called AE qualifier	U
Called AP invocation-identifier	U
Called AE invocation-identifier	U
Implementation information	O
User information	U

Table 3—AARE APDU fields

Field name	Presence
Protocol version	O
Application context name	M
Responding AP title	U
Responding AE qualifier	U
Responding AP invocation-identifier	U
Responding AE invocation-identifier	U

Table 3—AARE APDU fields *(continued)*

Field name	Presence
Result	M
Result source—diagnostic	M
Implementation information	O
User information	U

As can be seen, the majority of the fields are optional. Only a very small set is mandatory.

The ACSE in the interoperability profile is only a vehicle for a standardized connection setup. Additional information, supplied in the user information field, is defined in this standard to facilitate medical device interoperability.

6.2.3 ACSE ASN.1 message definition

Refer to Annex E for detailed definition.

For true interoperability, ACSE messages shall be BER encoded. Furthermore, they shall be wrapped in the corresponding presentation-layer PDU (PPDU) (CP: Connect Presentation, CPA: Connect Presentation Accept) and SPDU (CN: Session Connect, AC: Session Accept), as defined in Annex E.

6.2.4 ACSE user information fields

The user-specific (i.e., application-specific) information fields in the ACSE message definition for use in the inter-operability communication stack are defined in Annex E.

For startup, only minimal information needs to be supplied in the ACSE PDUs. After the association phase, all other necessary data for application and compatibility checking can be supplied in native CMDISE services using definitions from MDDL.

6.3 Session-layer protocol

6.3.1 General

6.3.2 Session-layer services

The session-layer protocol defines a set of services used for connection and data transfer. Important services to be considered here are

— Session connect

— Session accept

— Session data transfer

A number of additional services and PDUs may be relevant. This needs to be further investigated, especially regarding the mapping between ACSE, PPDUs, and SPDUs.

6.3.3 Session-layer message definitions

For all needed standard SSs as well as for the optimized session-layer extension, the PDU structure (session header) will be defined.

SPDUs are constructed from simpler elements in the form shown in Figure 3.

SPDUs	SI	LI	Parameter field	User information field
PGI units	PGI	LI	Parameter field	
PI units	PI	LI	Parameter field	

LEGEND

LI: length indicator (length 0-254: one octet; otherwise, 3 octets with first being 255)

PGI: parameter group identifier (defines a group of session-layer parameters)

PI: parameter identifier (defines a single session-layer parameter)

SI: SPDU identifier (unique identifier that defines the session-layer message type)

Figure 3—SPDU format

The parameter fields are constructed from PGI and PI units with certain rules (in accordance with ISO/IEC 8327-1).

An example of a session-layer message is given in 6.3.3.1.

6.3.3.1 Session connect (CN) SPDU

The format and contents of the CN SPDU are as follows:

```
0D XX             SI=13 (CN), LI = length in octets
   05 06             PGI=05(connect/accept item), LI=06
13 01 00          PI=19 (options), LI=1, value = 0
   16 01 03          PI=22 (version), LI=1, value = 3
   14 02 00 02       PI=20 (user requirements), LI=2, value:
                       select full duplex functional unit
C1 YY             PGI=193 (user data), LI= length of user data
```

6.4 Presentation-layer protocol

6.4.1 General

In general, the presentation-layer protocol allows negotiating the abstract syntax (e.g., MDDL over CMDISE ASN.1) and transfer syntax (i.e., optimized encoding rules, e.g., MDER) between systems.

As for the session-layer protocol, some limited standard services are necessary to support the ACSE.

The major added value of the presentation layer is the syntax negotiation during association. Also, it allows the definition of multiplex points (presentation context identifiers) in the application, which in turn allows the communication of data in different formats within the same association [e.g., Digital Imaging and Communications in Medicine (DICOM) images]. In a given association, it is possible to support multiple presentation contexts simultaneously.

For the normal communication between medical devices (after association), the processing and messaging requirements for the presentation layer should be as small as possible.

6.4.2 Presentation-layer services

The presentation layer provides, for example, the following services:

— Connect presentation
— Connect presentation accept
— Presentation data

6.4.3 Presentation-layer messages

Refer to Annex E for definition of the following PDUs:

— Connect presentation (CP) PPDU
— Connect presentation accept (CPA) PPDU
— Connect presentation reject (CPR) PPDU
— Abnormal release provider (ARP) PPDU
— Abnormal release user (ARU) PPDU
— Presentation data (TD) PPDU

6.5 ROSE protocol

6.5.1 General

The ROSE is used by the Common Medical Device Information Protocol (CMDIP). It provides a linkage between invoke messages and result messages (i.e., requests and responses) by means of invoke identifier fields. It also contains a field to distinguish the various remote operations (here: CMDISE services).

The ROSE protocol uses the same abstract syntax that is negotiated for the CMDIP and the data structures from ISO/IEEE 11073-10201. As a result, modifications to the ISO/OSI ROSE are needed to comply with the restrictions of optimized ASN.1 encoding rules.

6.5.2 ROSE services

The ROSE protocol is a relatively simple protocol defining the following services:

— Remote operation invoke
— Remote operation result
— Remote operation error
— Remote operation reject

6.5.3 ROSE message definitions

Refer to Annex E for ROSE PDU definitions.

6.6 CMDISE protocol (CMDIP)

6.6.1 General

6.6.2 CMDISE services

The following basic services may be provided by MDAPs, depending on scalability (i.e., minimum, basic, and extended profiles have different requirements):

- Retrieve object attribute value
- Modify object attribute value
- Invoke object defined functions
- Create and delete object instances
- Report events that occurred within an object

Parameters for each of the services and the related results are defined in the MDDL. Table 4 gives an example for the event report service:

Table 4—Event report service parameters

Parameter	Description
Invoke identifier	Unique identifier (e.g., a sequence number) assigned to a specific instance of the service so that it can be distinguished from other service invocations that a service provider may have in progress.
Mode	Confirmed or unconfirmed; confirmed mode requires a response
Object class	Identifies class of the object that generates the event (values defined in nomenclature or dictionary)
Object instance	Identifies instance of the object that generates the event
Event time	Time the event was generated
Event type	Identifies the type of the event (values defined in nomenclature or dictionary)
Event information	(Optional) additional information about the event, as defined by the event type parameter; the event information is defined by the object that generates the event

6.6.3 CMDIP message definitions

ASN.1 data type definitions for all CMDIP services are defined in Annex E.

Note that some CMDISE service parameters defined in MDDL may actually map to ROSE. In particular the invoke identifier and mode parameters are really defined in the ROSE, as explained in 6.5.

Refer to Annex E for PDU examples.

6.6.4 SNTP

Refer to Annex C.

7. Information model

7.1 Object model

This subclause is intended to include object class definitions, which in general include the following classes.

- — CC, e.g., DCC, BCC
- — MIB

Refer to the DIM for definition detail.

7.2 Format model

7.2.1 Syntax

7.2.1.1 Transfer syntax

Transfer syntax are defined in Annex A.

Mappings to ISO ASN.1 may be found in Annex G.

7.2.1.2 Abstract syntax

Abstract syntax is defined in Annex E.

7.2.2 Compatibility

In general, it is desirable to maintain syntactical backwards-compatibility, especially for transfer syntax, although such compatibility is not always possible. This subclause is intended to identify significant compatibility issues and resolution to facilitate implementation. Refer to Table 5.

Table 5—Compatibility cases

Case	Issue	Resolution
1	Compatibility of 1988/90 and 1994 versions of ISO ASN.1 and related encoding rules (ISO/IEC 8824 series, ISO/IEC 8825 series)	
1.1	ANY DEFINED BY changed; refer to Annex H for background information.	1. ABSTRACT SYNTAX: Modules shall follow guidelines in ISO/IEC 8824-1; see Annex H. 2. TRANSFER SYNTAX: MDER for ANY DEFINED BY or instance-of type shall be identical; see Annex A.

8. Conformance

This clause is intended to define conformance criteria.

8.1 Scope

To maximize implementation flexibility and interoperability, implementation conformance scope in this base standard is limited to protocols. However, application profiles may define service definitions, as appropriate.

8.2 Object identifier administration

This standard allocates object identifier administration authority to other standards, especially MDDL. Administration authority requires referencing of the MDAP arc and then specification of the extensions allocated by the extensive standard.

8.3 MDAP subset conformance

MDAP profiles shall specify the extent of conformance with this standard by exception categories, as follows:

a) *No exceptions:* The subset takes no exceptions to this standard.

b) *Some exceptions:* The subset takes some exceptions to this standard. In this case only the specific exceptions require specification. In the event that this standard defines a table for conformance specification, the subset standard shall replicate the table and fill in the specific cases of conformance for all cases, whether conforming or not.

8.4 Implementation conformance

Device implementers may not specify conformance with this standard, but must specify conformance with a profile that references this standard. As noted in 8.1, implementation conformance shall be limited to protocols, although informative, definitions of APIs are appropriate to facilitate reuse of implementation components.

Annex A

(normative)

Medical device encoding rules (MDER)

A.1 General

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 to 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 so-called *canned* messages in which only dynamically changing data need to be included in relatively high-frequency messages, particularly waves.

A.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 ISO/IEEE 11073-10201 where all the data definitions are formalized using ASN.1).

In order to support the requirement for encoding and decoding performance and support of canned messages, 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, packed encoding rules [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, 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 medical objects only, so 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, so other general standard encoding rules (e.g., BER, PER) can be used as well, as negotiated in the specific upper layers communication profile.

Table A.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.

Refer to Annex H for further details on specialization of ASN.1 types in MDER.

Table A.1—Supported ASN.1 data types

ASN.1 type	Status	Comments
INTEGER	R	Size constraints must 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 (–127..128) 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 must 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.

A.3 Byte order

Refer to Figure A.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:

— Representation in diagrams uses the NBO format shown in Figure A.1.

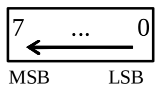
— No alignment is used in MDER. In other words, additional bytes are not added to byte strings, e.g., to obtain lengths that are divisible by two or four. However, variable length data items, i.e., strings, should have an even length for performance reasons. For example, because most data elements are 16 bit, they will not be misaligned if strings are even length.

— MDAP communicants are restricted to using the NBO (big endian) convention.

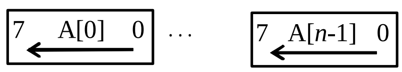
— The association protocol shall use ISO BER to provide for universal interoperability during negotiation of MDER conventions. All other PDUs exchanged in the life cycle of device-host communication will be based in MDER, e.g., CMIP* and ROSE* PDUs. The suffixed asterisk (*) indicates that MDER is used as an optimization of the ISO protocol, which is based typically in 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 A.1, the structure ABCD would be ordered DCBA. In this case, if big endian is the negotiated protocol, then a little endian machine would have 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 may be facilitated by non-normative definitions in this and related standards.

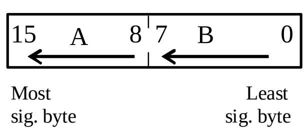
- NBO
  - One byte bit string, i.e., octet
    - Bit sequence: in order of least significant bit (LSB) to most significant bit (MSB), e.g. 0, ... , 7 or 24, ... , 31; bit ordering is representing in diagrams by the following notation, ←——— , in which the arrow tip represents the last bit transferred:



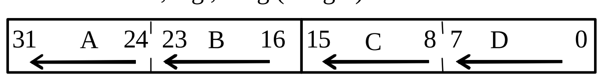
  - Multibyte string
    - Unstructured: an array of octets (i.e., an octet string)
      - Bit sequence: for each byte, as defined for octet
      - Byte sequence: generically numbered from [0] to [*n*-1], e.g., A[0] to A[*n*-1], where <*n*> = length in octets.



    - Structured: a multibyte ordering of bits, typically in multiples of two (e.g., a short integer is 16 bits, a long integer is 32 bits); floating point numbers in general are multiples of 16 bits, although in this standard, only a 32 bit FLOAT is defined. Two generic examples are given (ABCD refers to byte order):
      - 16 bit structure, e.g., short (integer)
        - Bit sequence: each byte transferred as defined above for octet
        - Byte sequence: transferred in order of most significant byte to least significant byte
        - For signed integers, typically the MSB of the most signficiant byte is the sign (s) bit.



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



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

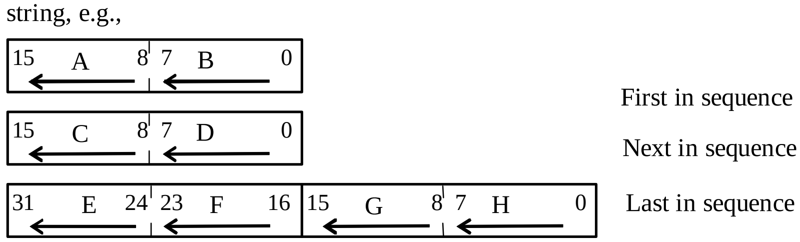


Figure A.1—Binary string [re]presentation conventions—NBO

A.4 Encodings

A.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 64K bytes (16 bits), which should be sufficient for communication.

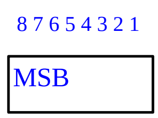
Simple types are defined because of size constraints and have fixed length. SEQUENCE types having fixed length are supported provided there are no OPTIONAL syntax components. If this is not tolerable, standard encoding rules must be defined for use in the standard profile.

A.4.2 INTEGER

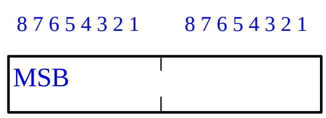
The encoding of an integer value is primitive, and the contents octets represent the value using a two-complement binary representation.

For the size-constrained integer values supported by MDER, Figure A.2 defines octet encodings.[8]

- 8 bit types INT-U8, INT-I8



- 16 bit types INT-U16, INT-I16



- 32 bit types INT-U32, INT-I32

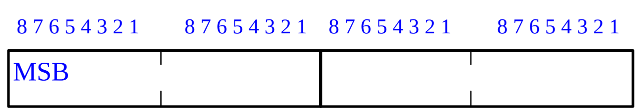


Figure A.2—Integer encodings

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

[8]To promote C programming language standardization for these integer data types, ISO/IEC 9899 definitions should be used.

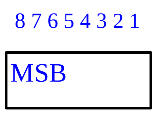
A.4.3 BIT STRING

The encoding of a bit string value is primitive, and the contents octets simply represent the bits set in the bit string.

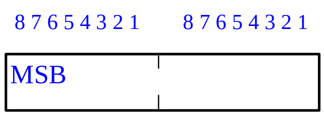
Bit 0 in the encoding is represented by the MSB, bit 1 is represented by the next bit in the octet, etc.

For the size constrained bit string values supported by MDER, Figure A.3 defines octet encodings.

- 8-bit types BITS-8



- 16-bit types BITS-16



- 32-bit types BITS-32

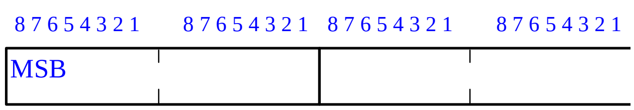


Figure A.3—BIT STRING encodings

Example:

A definition

```
state ::= BITS-16 {open(0), locked(1) }
```

can be 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).

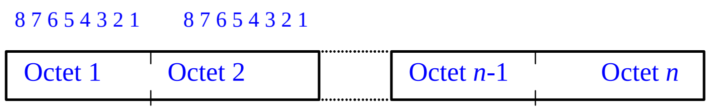
A.4.4 OCTET STRING

The encoding of an octet string value is primitive, and the contents octets simply represent the elements of the string. The octets themselves use an encoding inherent to the definition of the type of the string.

Dependent on this type, the octets may contain printable characters (in the case of 16 bit character sets, a character uses 2 octets in the encoding), or it may contain a larger area of encapsulated binary data.

In the encoding, MDER distinguishes between variable length OCTET STRING and size-constrained OCTET STRING as shown in Figure A.4:

- Fixed (size-constrained): OCTET STRING (SIZE(*n*))



- Variable-length OCTET STRING types

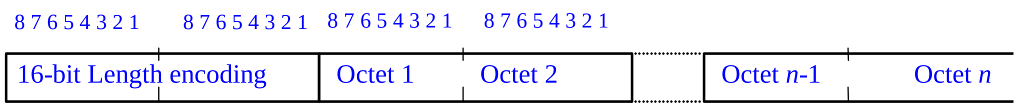


Figure A.4—OCTET STRING encodings

An OCTET STRING with a fixed (i.e., size-constrained) is encoded with the corresponding set of contents octets only.

Variable-length OCTET STRING types are encoded with a 16 bit length field (unsigned integer, two-complement), followed by the defined number of contents 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:

```
typedefunsigned char fixed_size_label[12];

typedef struct {
   unsigned short length;
   unsigned chardata[1];/* this is a placeholder for an appropriately
      sized array */
   } variable_label;
```

A.4.5 SEQUENCE

The encoding of a sequence value is constructed, and the contents octets represent the encoded values of the elements of the SEQUENCE type, without any further encoded data. No gaps (e.g., for alignment) are added.

The component values must appear in the order of their definition in the SEQUENCE type.

Example:

The following definitions

```
IdentType ::= SEQUENCE {
   id      INT-U16,
   instanceINT-U16
   }
```

can be mapped to C language type representations as follows:

```
typedef struct {
   unsigned shortid,
   unsigned shortinstance
   } IdentType;
```

and has the MDER encoding in Figure A.5:

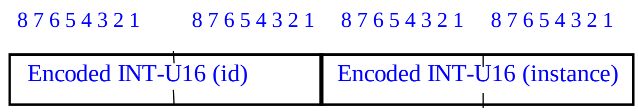


Figure A.5—Sample encoding of a SEQUENCE

A.4.6 SEQUENCE OF

The encoding of a value of the SEQUENCE OF type is constructed, and the contents octets represent the encoded values of the elements of the SEQUENCE OF type, preceded with a count field specifying the number of elements and a length field specifying the complete length of the data structure (without count and length themselves).

The encoding must preserve the order of the component values. See Figure A.6.

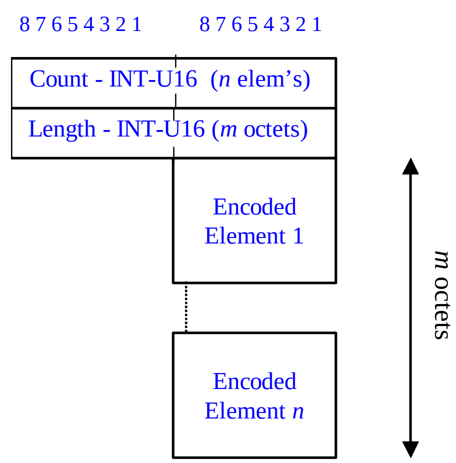


Figure A.6—Encoding of SEQUENCE OF

Count and length fields with contents 0 indicate an empty list data structure. Such a value combination is allowed.

Example:

The following definitions

```
Array1 ::= SEQUENCE OF Entry
```

can be mapped to a C language type representation as follows:

```
typedef struct {
    unsigned shortcount;
    unsigned shortlength;
    Entry  data[1]; /* placeholder for sufficient number of entries */
} Array1;
```

A.4.7 CHOICE

The encoding of a choice value is constructed, and the contents octets represent the encoded values of the chosen alternative, preceded with a tag field specifying the selected alternative and a length field specifying the length of the encoding of the selected alternative. See Figure A.7.

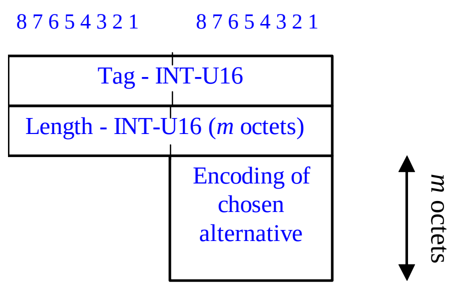


Figure A.7—Encoding of CHOICE

Example:

The following definitions

```
ChoiceType ::= CHOICE {
    one OneType,
    two TwoType
}
```

can be mapped to a C language type representation as follows:

```
typedef struct {
    unsigned shortchoice_id;
    unsigned shortlength;
    union {
        OneTypeone;
        TwoTypetwo;
        } data;
} ChoiceType;

#define one_type_chosen1
#define two_type_chosen2
```

The rules for tag values are defined as follows:

— Tags may be 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 would be 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 should be sequential, but may be 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
    }
```

A.4.8 ANY DEFINED BY and instance-of

The encoding of a value for the ANY DEFINED BY type (ASN.1 1988/90) or the instance-of type (ASN.1 1994) is constructed, and the contents octets represent the encoded values of the selected value, preceded with a length field specifying the length of the encoding of the selected value. See Figure A.8.

The types are used to represent embedded syntaxes using a registered object identifier. Refer to Annex H for compatibility cases.

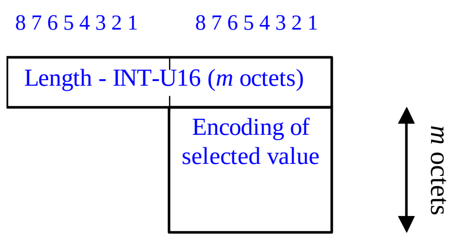


Figure A.8—Encoding of ANY DEFINED BY (instance-of)

Example:

The following definitions

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

can be mapped to a C language type representation as follows:

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

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

In the preceding mapping, the type-id field is a context-free object identifier. An application has to use the identifier 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.

A.5 Floating point data structure

The restricted subset of ASN.1 that can be mapped with MDER does not contain the ASN.1 data type FLOAT.

Instead, a generic data type FLOAT-Type is defined in ISO/IEEE 11073-10201 for floating point numbers.

A 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 magnitude. See Figure A.9.

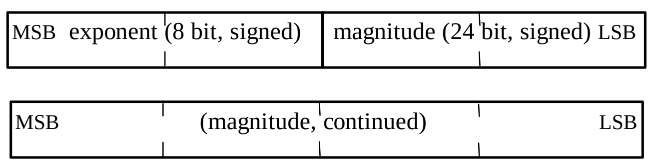


Figure A.9—MDNF encoding

The number represented is (magnitude)*(10**exponent).[9] Both the exponent and magnitude are in two-complement form. The magnitude is not necessarily normalized.

There are four special values that can be represented as shown in Table A.2.

Table A.2—MDNF special values

Special value	Magnitude
NaN (not a number)	+(2**23–1)
NRes (not at this resolution)	–(2**23)
+ INFINITY	+(2**23–2)
– INFINITY	–(2**23–2)

[9]The double asterisk (**) is used to represent the exponent operation.

The exponent is not important in these cases. This leaves the following ranges for normal number representation:

- –128 ≤ exponent ≤ 127
- –2((2**23)–3) ≤ magnitude ≤ +((2**23)–3)
- NaN = +((2**23)–1)
- NRes = –2(2**23)
- ± INFINITY = ± ((2**23)–2)

Definitions of the number of the valid digits for the presentation on a display are as follows:

- If the exponent < 0, then the integer value of the exponent shows the number of valid digits after the point. See the examples in Table A.3.

Table A.3—Examples when exponent < 0

Exponent	Magnitude	Value
–3	32 000	32.000
–1	320	32.0

- If the exponent ≥ 0, then the number of valid digits after the point is zero. See the examples in Table A.4.

Table A.4—Examples when exponent ≥ 0

Exponent	Magnitude	Value
1	320	3200
2	32	3200

Annex B

(normative)

Allocation of identifiers

B.1 Introduction

This annex is provided for standard writers and implementers of the ISO/IEEE 11073-20000 family of standards as the guideline for allocation of object identifiers for ISO/IEEE 11073 standards (refer to ISO/IEC 8824-2 for definition and use of object identifiers).

B.2 Allocation framework

For brevity, object identifiers assigned in this annex use indented, tabular form, in which each indent corresponds with a branch. Refer to Figure B.1.

The root for the object identifiers in this document is

```
iso(1)
|
•—member-body(2)
    |
    •—US(840)
        |
        •—ieee1073(10004)
            |
            •—mdap(2)
                |
                •—modules(0)
                |
                •—version1(1)            MDAP Version 1
                    |
                    •—mdap-0(0)          Base standard
```

Figure B.1—Object identifier assignments—root path

The arcs below the mdap-0 root are as shown in Figure B.2.[10]

```
Arc                          Definition
StandardSpecificExt(0)       mOSI extensions (see E.1)
|
 •—modules(0)                    Syntactical and contextual extensions
|   |
|   •—abstractSyntax(1)              Abstract syntax  (see E.3)
|   |   |
|   |   •—nomen16(1)                     Nomenclature -- 16 bit context-sensitive
|   |   |
|   |   •—nomen32(2)                     Nomenclature -- 32 bit context-free
|   |
|   •—transferSyntax(2)              Transfer syntax  (MDER)
|   |   |
|   |   •—mderBigEndian(1)               Big endian
|   |   |
|   |   •—mderLittleEndian(2)            Little endian
|   |
|   •—applicationContext(3)          Application context (see Annex D)
|       |
|       •—testMode(0)                    Test operation
|       |
|       •—normalMode(1)                  Normal operation
|
•—asn1Modules(2)             MDAP-specific ASN.1 modules  (see E.2)
    |
    •—aAssociateUserInfo(0)      Association user information
    |
    •—remoteOps(1)               ROSE optimizations
    |
    •—commonMgmt(2)              CMIP optimizations
    |
    •—mdapExt(3)                 MDAP extensions
    |
    •—mib(4)                     MIB definitions
```

NOTE—Additional branches as defined in IEEE Std 1073 are omitted for brevity.

Figure B.2—Object identifier allocations—standard specific extensions

B.3 Derivation examples

This clause shows derivation of several object identifiers using the definition of Figure B.1.

Example 1) Presentation context: defined as an abstract-transfer syntax pair, for example,

— Abstract syntax: `nomen16` *(device not otherwise specified [NOS])*

```
iso(1) member-body(2) US(840) ieee1073(10004) mdap(2) version1(1)
mdap-0(0) standardSpecificExt(0) modules(0) abstractSyntax(1)  1
```

[10]Notes in text, tables, and figures are given for information only and do not contain requirements needed to implement this standard.

— Transfer syntax: `mderBigEndian`

```
iso(1) member-body(2) US(840) ieee1073(10004) mdap(2) version1(1)
mdap-0(0) standardSpecificExt(0) modules(0) transferSyntax(2) 1
```

Example 2) Application context, for example,

— Application context: DCC normal operation mode

```
iso(1) member-body(2) US(840) ieee1073(10004) mdap(2) version1(1)
mdap-0(0) standardSpecificExt(0) modules(0) applicationContext(3) dcc(2) 1
```

Annex C

(informative)

Time synchronization

C.1 Purpose

This annex is reserved for specification detail concerning time synchronization packages specified by application profiles referencing this standard.

C.2 Scope

This standard does not presently define a method for time synchronization between medical devices in order to permit implementation in lower layers (e.g., ISO/IEEE 11073-30200, ISO/IEEE 11073-30300) as well as object-oriented implementation in application-layer-based format profiles (e.g., the DIM).

C.3 Specification

To facilitate interoperability, application profiles applying SNTP shall be consistent with relevant definitions in ISO/IEEE 11073-30200 concerning protocols and the DIM concerning the related clock object.

Annex D

(informative)

Dynamic model

General definitions may be found in the DIM. This annex includes additional information.

Refer to the tables in this annex for a tabular representation of the finite state model (FSM). Table D.1 maps the FSM for a DCC agent-only, and Table D.2 maps a BCC manager-only. In these tables, “-only” implies that each side does not provide symmetrical client-server capability.

Profiles leveraging this standard should revise Table D.1 and Table D.2 as appropriate for equivalent visualizations (i.e., state diagrams), clarifications, or extensions; and they should take these into account in implementation conformance statements.

Table D.1 shows the state transition table of the device agent system (e.g., infusion pump). Events in *italics* are external events (e.g., generated by the manager system); other events are internal events. Example: If the agent is in the **Disconnected** state and receives a *connect event*, then the agent sends a boot notification and changes to the **Unassociated** state.

Empty fields in Table D.1 mean that the event does not cause any actions or state changes.

Table D.2 shows the state transition table of the DCC host, i.e., BCC manager (e.g., infusion pump host).

Empty fields in Table D.2 mean that the event does not cause any actions or state changes. Some states that are in the state diagram are left from the table for editorial purposes (transitions should be obvious).

NOTE—A specific *hot start* behavior that may allow the **Configuring** state to be skipped is not yet defined and may be added to these tables at a later point.[11]

[11]Notes in text, tables, and figures are given for information only and do not contain requirements needed to implement this standard.

Table D.1—DCC Agent-only state

Event	**State**						
	Disconnected	**Unassociated**	**Associating**	**Associated**	**Configuring**	**Configured**	**Operating**
connect event (e.g., sensed physical connection)	**Unassociated**						
server boot notification		if assoc. allowed {send assoc. req **Associating**}					
association accept			send assoc accept **Associated**				
association reject			send assoc reject **Unassociated**				
MDS Create Event				if (server needs it) send MDS create notification create CTXT scanner **Configuring**			
Context Scanner Create Response					rx obj. create ERs create other scanners send MDS state chg. **Configured**		
MDS State Change (to operating)[a]						**Operating**	
Reconfiguration (e.g., create/delete events)							update configuration **Operating**
termination intention				**Terminating**	**Terminating**	**Terminating**	**Terminating**

Table D.1—DCC Agent-only state *(continued)*

Event	State						
	Disconnected	**Unassociated**	**Associating**	**Associated**	**Configuring**	**Configured**	**Operating**
association release request				send release response reset configuration **Unassociated**	send release response reset configuration **Unassociated**	send release response reset configuration **Unassociated**	send release response reset configuration **Unassociated**
association abort			reset configuration **Unassociated**	reset configuration **Unassociated**	reset configuration **Unassociated**	reset configuration **Unassociated**	reset configuration **Unassociated**
disconnect event		reset configuration **Disconnected**	reset configuration **Disconnected**	reset configuration **Disconnected**	reset configuration **Disconnected**	reset configuration **Disconnected**	reset configuration **Disconnected**
recoverable error					send MDS error if state change { send MDS state **new state** }else **Configuring**	send MDS error if state change { send MDS state **new state** }else **Configured**	send MDS error if state change { send MDS state **new state** }else **Operating**
MDS State Change					check if transition allowed confirm state change **new state**	check if transition allowed confirm state change **new state**	check if transition allowed confirm state change **new state**
unrecoverable error			reset configuration send assoc. abort (if possible) **Unassociated**	reset configuration send assoc. abort (if possible) **Unassociated**	reset configuration send assoc. abort (if possible) **Unassociated**	reset configuration send assoc. abort (if possible) **Unassociated**	reset configuration send assoc. abort (if possible) **Unassociated**

[a] In certain fault conditions, the end of the configuring state might not be detected.

Table D.2—BCC manager-only state

Event	State						
	Disconnected	**Unassociated**	**Associating**	**Associated**	**Configuring**	**Configured**	**Operating**
connect event (e.g., sensed physical connection)	**Unassociated**						
association request		check if assoc. allowed **Associating**					
association accept			send assoc accept **Associated**				
association reject			send assoc reject **Unassociated**				
configure				send MDS create notification **Configuring**			
Context Scanner Create Command					init. CTXT scanner send obj. create ERs send MDS state chg. **Configured**		
operation						init. all episodic data **Operating**	
Reconfiguration (e.g., new/modified scanners)							update configuration **Operating**
termination intention				**Terminating**	**Terminating**	**Terminating**	**Terminating**

Table D.2—BCC manager-only state *(continued)*

Event	State						
	Disconnected	**Unassociated**	**Associating**	**Associated**	**Configuring**	**Configured**	**Operating**
association release request				send release response reset configuration **Unassociated**	send release response reset configuration **Unassociated**	send release response reset configuration **Unassociated**	send release response reset configuration **Unassociated**
association abort			reset configuration **Unassociated**	reset configuration **Unassociated**	reset configuration **Unassociated**	reset configuration **Unassociated**	reset configuration **Unassociated**
disconnect event		reset configuration **Disconnected**	reset configuration **Disconnected**	reset configuration **Disconnected**	reset configuration **Disconnected**	reset configuration **Disconnected**	reset configuration **Disconnected**
recoverable error					send MDS error if state change { send MDS state **new state** }else **Configuring**	send MDS error if state change { send MDS state **new state** }else **Configured**	send MDS error if state change { send MDS state **new state** }else **Operating**
MDS State Change					check if transition allowed confirm state change **new state**	check if transition allowed confirm state change **new state**	check if transition allowed confirm state change **new state**
unrecoverable error			reset configuration send assoc. abort (if possible) **Unassociated**	reset configuration send assoc. abort (if possible) **Unassociated**	reset configuration send assoc. abort (if possible) **Unassociated**	reset configuration send assoc. abort (if possible) **Unassociated**	reset configuration send assoc. abort (if possible) **Unassociated**

Annex E

(normative)

Abstract syntax

This annex defines several specializations of abstract syntax, specifically

- — mOSI extensions related to the session layer and presentation layer (see E.1)
- — ASN.1 modules related to application remote operation and common management information services and protocols (e.g., ROSE*, CMIP*) (see E.2)
- — abstract and transfer syntax extensions related to MDDL and MDER (see E.3)
- — MIB proxy definitions (see E.4)

Assumptions for specifications and examples provided in this annex include the following:

a) ISO syntaxes are replicated and hex[adecimal] annotations (dumps) of octet strings provided. Although these definitions are more complex to specify literally in detail, they facilitate implementation by giving a more literal compilation of the mappings between abstract and transfer syntaxes. ISO BER and MDAP MDER encodings are assumed based on presentation context, specifically ISO ACSE/BER and ISO/IEEE 11073 MDDL/MDER.

b) For consistency with respect to application layer definitions, the session-layer and presentation-layer extensions (in E.1.1 and E.1.2) may be defined in separate ASN.1 modules.

E.1 mOSI extensions

MDAP standard-specific extensions pertain to mOSI session-layer and presentation-layer adaptations that have been optimized for use in medical devices.

E.1.1 Session layer

A full specification of mOSI requirements for session-layer facilities can be found in ISO/IEC ISP 11188-3.

The session layer supports the kernel and duplex functional units only. The basic concatenation protocol mechanism is supported; the maximum allowed size of SS user data must be larger than 512 bytes (e.g., no support for expedited data, segmenting).

Basic concatenation means that the session layer concatenates only a single Category 0 SPDU with a single Class 2 SPDU (unlike extended concatenation, which can contain multiple Category 2 SPDUs).

In this standard, a list of supported SPDUs is defined using a textual notation that specifies the SPDU contents. The described elements of the SPDUs are mandatory. Note that the SPDUs typically contain session-layer user data, which would be appended to the messages listed, but are not shown in this subclause.

The following abbreviations are used (in accordance with ISO/IEC 8327-1):

- — LI: length indicator (length 0–254: one octet, otherwise 3 octets with first being 255)
- — PGI: parameter group identifier (defines a group of session-layer parameters)
- — PI: parameter identifier (defines a single session-layer parameter)
- — SI: SPDU identifier (unique identifier that defines the session-layer message type)

E.1.1.1 Session connect (CN) SPDU

The format and contents of the CN SPDU are as follows:

```
0D XX                   SI=13 (CN), LI = length in octets
   05 08                PGI=05(connect/accept item), LI=08
   13 01 00             PI=19 (options), LI=1, value = 0
   16 01 02             PI=22 (version), LI=1, value = 2
   80 00                PI=128 (MDAP Session Extensions), LI=0
   14 02 00 02          PI=20 (user requirements), LI=2,
                        value: select full duplex functional unit
C1 YY                   PGI=193 (user data), LI= length of user data
```

This SPDU normally contains the connect presentation message (see E.1.2.1), which in turn contains the ACSE association request (see E.1.3.1).

E.1.1.2 Session accept (AC) SPDU

The format and contents of the AC SPDU are as follows:

```
0E XX                   SI=14 (AC) LI = length in octets
   05 08                PGI=05(connect/accept item), LI=06
      13 01 00          PI=19 (options), LI=1, value = 0
      16 01 02          PI=22 (version), LI=1, value = 2
      80 00             PI=128 (MDAP Session Extensions), LI=0
      14 02 00 02       PI=20 (user requirements), LI=2,
                        value: confirms selection of full duplex FU
   C1 YY                PGI=193 (user data), LI= length of user data
```

The AC SPDU normally contains the connect presentation response (see E.1.2.2), which in turn contains the ACSE association response (see E.1.3.1).

E.1.1.3 Session refuse (RF) SPDU

The format and contents of the RF SPDU are as follows:

```
0C 03                   SI=12 (RF), LI = length in octets (fixed: 3)
   32 01 00             PI=50 (reason), length 1, reason not specified
```

The RF SPDU is used as a reply in case of unsupported session options or corrupt session header information (in MDAP, only this single refuse SPDU is used).

E.1.1.4 Session finish (FN) SPDU

The format and contents of the FN SPDU are as follows:

```
09 XX                   SI=9 (FN), LI = length in octets
   C1 YY                PGI=193 (user data), LI = length of udata
```

The FN SPDU usually contains a normal presentation data value (PDV) that in turn contains the ACSE association release request (see E.1.3.1). See Figure F.3 for an example.

E.1.1.5 Session disconnect (DN) SPDU

The format and contents of the DN SPDU are as follows:

```
0A XX                  SI=10 (DN), LI = length in octets
   C1 YY               PGI=193 (user data), LI = length of udata
```

The DN SPDU usually contains a normal PDV that in turn contains the ACSE association release response (see E.1.3.1). See Figure F.4 for an example.

E.1.1.6 Session data transfer (DT) SPDU

The format and contents of the DT SPDU are as follows:

```
01 00                  SI=1 LI=0 GIVE TOKEN (GT) SPDU
01 00                  SI=1 LI=0 DATA TRANSFER (DT) SPDU
```

Data transfer is a Category 2 SPDU that must be preceded by a Category 0 SPDU (the token *give*, which has the same value for the SI field). The LI fields indicate that no parameters (i.e., PGI fields, PI fields) are present. The user information is simply appended to this message.

The DT SPDU contains either

a) A TD PPDU, or

b) An MDAP-TD PPDU.

E.1.1.7 Session abort (AB) SPDU

There are two basic forms of the AB SPDU. The first is when no user data are included with the message (see 8.3.9.3 in ISO/IEC 8327-1):

```
19 03                  SI=25 LI = length in octets
   11 01 09            PI=17 (transport disconnect),LI=1, value=3 (bit 1
                       = transport release; bit 4 = no reason (i.e., no
                       ARP/ARU))
```

The second form is used when user information is provided:

```
19 XX                  SI=25 LI = length in octets
   11 01 03            PI=17 (transport disconnect),LI=1, value=3 (bit 2
                       = user abort; bit 1 = transport release)
   C1 YY               PGI=193 (user data), LI= length of data
```

The AB SPDU of this second form may contain one of the following:

a) An abnormal release provide (ARP) PPDU with no association user data

b) An abnormal release use (ARU) PPDU that contains an abort (ABRT) APDU

c) Empty user data, in which case the length would be null (i.e., 0xC1 0x00)

Because there are no MDDL abort user data defined in this standard, the information contained in the ARP or ARU PPDU is of minimal value to the AB SPDU recipient. As a result, the first session-only form is the preferred AB SPDU format.

Whichever form is used (i.e., with or without user data), the receipt of an AB SPDU (SI = 25) will be sufficient to terminate the session.

E.1.1.8 Session abort accept (AA) SPDU

The AA SPDU is not used.

E.1.1.9 MDAP session-layer extensions

MDAP defines additional session-layer services. The purpose of the session-layer extension is to facilitate simple and efficient demultiplexing of standard (mOSI) and nonstandard (MDAP) PPDUs. Here, demultiplexing is possible by using a single byte; and more importantly BER encoding, which would require a length field with variable size, can be avoided on the presentation header.

The MDAP session-layer extension also provides coalescing provisions. Similar to extended concatenation in the standard session layer, multiple PPDUs can be concatenated in a single SPDU. This is especially useful to reduce the number of messages transiting the lower layers (and thus requiring processing resources).

E.1.1.9.1 MDAP session connect/accept items

Use of the MDAP session-layer extensions is negotiated during session connection establishment.[12] Two additional PI fields are defined in the connect/accept item:

```
1:   80 00           PI=128 (enable MDAP Session Extensions), LI=0
2:   81 01 xx        PI=129 (enable MDAP coalescing), LI=1, value = xx with
                        0000 0001b = 32ms period
                        0000 0010b = 64ms period
                        0000 0100b = 128ms period
                        0000 1000b = 256ms period
                        etc.
```

A CN SPDU example is as follows:

```
0D XX                SI=13 (CN), LI = length in octets
   05 08             PGI=05(connect/accept item), LI=08
   13 01 00          PI=19 (options), LI=1, value = 0
   16 01 03          PI=22 (version), LI=1, value = 3
   80 00             PI=128 (MDAP Session Extensions), LI=0
   14 02 00 02       PI=20 (user requirements), LI=2, value
                     select full duplex functional unit
   C1 YY             PGI=193 (user data), LI= len
```

E.1.1.9.2 MDAP data transfer SPDUs

The MDAP session-layer extension defines additional SI fields for normal and expedited data transfer. The formats are as follows:

— MDAP normal data transfer (MDAP-DT) SPDU

```
E1 00                SI=E1h LI = 0
```

— MDAP expedited data transfer (MDAP-XT) SPDU

```
E2 00                SI=E2h LI = 0
```

[12]The method of negotiation is specific to the application profile used; however, in general a manager system may indicate support for coalescence. If the agent system also indicates support in its association request response, then coalescence is enabled. The manager and agent independently specify the flush period that will be used. See E.1.1.9.3.

These messages contain a MDAP presentation message and are passed to the MDAP presentation-layer extension instead of the normal presentation layer. No PGI fields or PI fields are defined; hence the LI field is 0. These are defined as Category 1 SPDUs that do not require the token *give* (just like typed data in normal session).

User data are simply appended to these messages and contain an MDAP PDV (MDAP-PPDU), which is assumed to be a CMIP*/ROSE* PDU.

E.1.1.9.3 MDAP session coalescing

For reducing the total number of MDAP messages, the MDAP session layer supports coalescing, in which multiple MDAP-DT PPDUs are combined and sent in a single SPDU. An SPDU buffer is sent (passed to the lower layers) if the buffer size would exceed the network maximum transfer unit (MTU) with the next MDAP-DT part or after a certain flush time period, which can be set at session connect time.

In the case that multiple MDAP-DT PPDUs are combined, length fields must be added to the components so that the session layer can demultiplex the individual parts. The LI field is used for this, indicating the overall length of the SPDU. In accordance to ISO/IEC 8327-1, a 3-octet-long LI field with the first octet being 0xFF is used to encode lengths within the 255–65535 range. The MDAP session-layer extension uses this length representation to encode lengths within the 0–65535 range. Given this, the initial octet of the SPDU LI can be used to indicate whether the SPDU has one or multiple PPDUs (normally it is 0x00; 0xFF would indicate coalesced PPDUs).

Each coalesced PPDU also includes a length so that the multiple PPDUs can be separated. As a result, the PPDUs shall be an array of multiple entries of the following form (when coalescing is enabled[13]):

```
LL LL     Length of the PPDU
CC CC     INT-U16 with the Presentation Context ID
DD DD     MDAP-User-data
```

Therefore, an SPDU with three embedded PPDUs would have the following structure:

```
E1 FF     SI=MDAP-DT SPDU; LI=255 indicating multiple PPDUs
XX XX     Length of entire SPDU

LL LL     Length of the PPDU #1
CC CC     INT-U16 with the Presentation Context ID
DD DD     MDAP-User-data

LL LL     Length of the PPDU #2
CC CC     INT-U16 with the Presentation Context ID
DD DD     MDAP-User-data

LL LL     Length of the PPDU #3
CC CC     INT-U16 with the Presentation Context ID
DD DD     MDAP-User-data
```

Use of coalescing increases message latency, but bandwidth conservation benefits (fewer PDUs).

For some response functions, this increased latency is unwanted. An implementation of the MDAP communication stack should, therefore, supply a push or flush function that can force transmission of the session buffer.

[13]See E.1.1.9.

E.1.2 Presentation layer

As with the session-layer protocol, there are two different elements or parts in the MDAP presentation layer. The mOSI element provides the standard presentation Kernel and duplex functional units.

The MDAP presentation element, which has to be considered an extension to the normal presentation-layer protocol, can be used for normal user data only. It receives data from the MDAP session element and sends data to the MDAP session element. It does not work with the standard mOSI session element.

A full specification of mOSI requirements for presentation-layer facilities can be found in Annex B of ISO/IEC ISP 11188-3. As indicated, the presentation layer supports the kernel and duplex functional units only.

The PPDUs that are supported are defined in E.1.2.1 through E.1.2.7. The PPDU descriptions given here are incomplete; they are only intended to give an overview of the principle contents of the PDUs. The complete definitions can be found in ISO/IEC 8327-1. Supported options are according to ISO/IEC ISP 11188-3. The PPDUs are BER encoded even if the user data field itself is not. Examples of encoded PPDUs can be found in Annex F.

E.1.2.1 Connection presentation (CP) PPDU

The CP PPDU is defined as follows:

```
CP-type ::= SET {
  mode-selector            [0] IMPLICIT Mode-selector,  -- must be "normal
                                                        mode"
  normal-mode-parameters   [2] IMPLICIT SEQUENCE {
  protocol-version         [0] IMPLICIT Protocol-version DEFAULT
                              {version-1},
  presentation-context-definition-list
                           [4] IMPLICIT Presentation-context-
                              definition-list,
  user-data                User-data OPTIONAL
  } OPTIONAL
       -- shall be used for normal mode only.
       -- shall be the parameters of the CP PPDU
}
```

This PPDU contains the association request message as user data (see E.1.3.1). It is contained in a CN SPDU (see E.1.1.1).

The presentation context definition list (see E.1.2.8) defines tuples of abstract syntax (e.g., MDAP) and transfer syntax (e.g., MDAP big endian transfer syntax), identified as object identifiers, and assigns a presentation context identifier (an integer) for each of these combinations. This identifier is 16 bit at maximum.

The presentation context list will contain one entry for the ACSE protocol (ACSE abstract syntax and transfer syntax). One additional entry contains MDAP abstract syntax with a list of possible transfer syntaxes (e.g., one for little endian, one for big endian, if both are supported).

The optional presentation context user data (see E.1.2.8) defines an application context with an ACSE request message (see AARQ-apdu in E.1.3.1). The AARQ-apdu in turn contains the association user info (see MDSEUserInfo in E.2.3).

E.1.2.2 Connect presentation accept (CPA) PPDU

The CPA PPDU is defined as follows:

```
CPA-PPDU ::= SET
  mode-selector            [0] IMPLICIT Mode-selector,  -- must be "normal
                                                        mode"
  normal-mode-parameters   [2] IMPLICIT SEQUENCE {
   protocol-version        [0] IMPLICIT Protocol-version DEFAULT {version-1},
   presentation-context-definition-result-list
                           [5] IMPLICIT Presentation
                                                     -context-definition
                                                     -result-list,
   user-data               User-data OPTIONAL
  }
   -- shall be used for normal mode only.
}
```

This PPDU contains the (positive) association response message as user data (see E.1.2.8). It is contained in a AC SPDU (see E.1.1.2).

The result list specifies which presentation contexts are accepted or rejected by the responder. The result list will contain an entry for each proposed presentation context, in the same sequence, along with the appropriate accept or reject indication.

The presentation context result list shall accept the ACSE context and the MDAP context. For the MDAP context, it shall select one transfer syntax.

After that, two presentation contexts are defined in the association. In other words, there are only two valid p-context identifiers. (The two contexts define the presentation-defined context set.)

E.1.2.3 Connect presentation reject (CPR) PPDU

The CPR PPDU is defined as follows:

```
CPR-PPDU ::= CHOICE {          -- only normal mode def. here
  normal-mode-parameters SEQUENCE {
   presentation-context-definition-result-list[5] IMPLICIT Presentation
                                                  -context-definition-
                                                  result-list,
   provider-reason             [10] IMPLICIT Provider-reason,
   user-data                   User-data,
  }
```

This PPDU contains the negative association response message as user data, namely, an AARE with the result field set to rejected-permanent or rejected-transient. It is contained in an AC SPDU (even though it does not accept the connection).

E.1.2.4 Abnormal release provider (ARP) PPDU

The ARP PPDU does not contain user data. It is contained in an AB SPDU.

The ARP PPDU is sent when an ill-formed PPDU is received, for one of the following reasons:

a) The PPDU contains an invalid PPDU parameter value.

b) The PPDU contains an unexpected PPDU parameter .

c) The PPDU includes an unexpected presentation context identifier.

d) The PDV is not valid.

E.1.2.5 Abnormal release user (ARU) PPDU

The ARU PPDU contains the association abort message as user data. It is contained in an AB SPDU.

The ARU PPDU is sent when the application has requested that a presentation context be abnormally released.

E.1.2.6 Presentation data (TD) PPDU

The TD PPDU is defined as follows:

```
TD-PPDU ::= User-data
```

This PPDU contains a user data PDV message. It is contained in a DT SPDU. Refer to E.1.2.8 for user data type definitions.

NOTE—The TD PPDU is defined as a part of this profile because it is required for a minimal OSI presentation layer (ISO/IEC ISP 11188-3). However, all medical device data are expected to use the MDAP-TD presentation-layer extension service. There are no defined uses at this time for the TD PPDU.

E.1.2.7 MDAP-TD (presentation data for MDAP presentation-layer extension)

The presentation-layer extension defines one additional PPDU type as follows:

```
MDAP-PPDU ::= SEQUENCE {
   presentation-context-id    INT-U16,
   user-data                  MDAP-User-data
}
```

The MDAP-PPDU is encoded using MDER, not BER.

— The presentation context identifier is a 16 bit (big endian) integer number. The identifier value itself is defined in the CP PPDU.

— The user data field is opaque; no special data structure is defined (e.g., no length field). However, the content is typically a ROSE* APDU, which in turn nests a CMIP* APDU (see E.2.1).

For example, the beginning part of an unconfirmed event report would be presented in MDER as follows:

```
--   Abstract syntax                                        Hex encoding
--   _____________                                          _________________
--   MDAP Normal Data Transfer SPDU (MDAP-DT)               E1 00
--   presentation-context-id (e.g., #1)                     00 01
--   ROSE* Invoke (ROIVapdu), length                        00 01  xx xx
--   Invoke ID (e.g., #1, Unconfirmed mode, arg Length)     00 01  00 00  xx xx
```

The MDAP-PPDU can contain a single PDV only. It is contained in a MDAP-DT SPDU, and it contains MDAP application (a ROSE* PDU) as user data.

The MDAP presentation-layer extension is negotiated at presentation connect time. The protocol version must be explicitly specified as version-mdap(15).

E.1.2.8 Presentation-layer user data type definitions

Presentation-layer user data are composed of the following data structures:

```
User-data ::= CHOICE {
   [APPLICATION 1] IMPLICIT Fully-encoded-data } -- other choice left out
Fully-encoded-data ::= SEQUENCE OF PDV-list
PDV-list ::= SEQUENCE {
  transfer-syntax-name           Transfer-syntax-name OPTIONAL,
  presentation-context-identifer Presentation-context-identifier,
  presentation-data-values       CHOICE {
    single-ASN1-type [0] ABSTRACT-SYNTAX.&Type (CONSTRAINED BY {
      -- Type corresponding to presentation context identifier -- }),
    octet-aligned    [1] IMPLICIT OCTET STRING,
    abritrary        [2] IMPLICIT BIT STRING }
--Contains one or more PDVs from the same presentation context.
}
```

These PDU definitions need the following additional data types:

```
Mode-selector ::= SET {
  mode-value [0] IMPLICIT INTEGER {
  x410-1984-mode(0),
  normal-mode (1)             -- this is the only MDAP supported mode
                              }
  }

Abstract-syntax-name ::= OBJECT IDENTIFIER

Context-list ::= SEQUENCE OF SEQUENCE {
   presentation-context-identifier  Presentation-context-identifier,
   abstract-syntax-name             Abstract-syntax-name,
   transfer-syntax-name-list        SEQUENCE OF
                                      Transfer-syntax-name
   }

Presentation-context-definition-list ::= Context-list
Presentation-context-definition-result-list ::= Result-list
Presentation-context-identifier ::= INTEGER    -- MDAP supports only
                                               16 bit integers
Protocol-version ::= BIT STRING {version-1(0)}

Provider-reason ::= INTEGER {
   reason-not-specified(0)                -- other reasons left out for
                                             the sake of this document
                }

Result ::= INTEGER{
   acceptance (0),
   user-rejection (1),
   provider-rejection(2)
           }

Result-list ::= SEQUENCE OF SEQUENCE {
   result                     [0] IMPLICIT Result,
   transfer-syntax-name       [1] IMPLICIT Transfer-syntax-name OPTIONAL,
   provider-reason            [2] IMPLICIT INTEGER {
     reason-not-specified (0),
```

```
      abstract-syntax-not-supported (1),
      proposed-transfer-syntaxes-not-supported (2),
      local-limit-on-DCS-exceeded (3)
                                            } OPTIONAL
                          }

Transfer-syntax-name ::= OBJECT IDENTIFIER
```

E.1.3 Application

E.1.3.1 Association

Association uses only the mandatory fields and the user information, as derived from the ASN.1 definitions of ITU-T Recommendation X.227.

```
ASSOC-apdu ::=  CHOICE {
   aarq                AARQ-apdu,   -- Association Request
   aare                AARE-apdu,   -- Association Response
   rlrq                RLRQ-apdu,   -- Assoc. Release Request
   rlre                RLRE-apdu,   -- Assoc. Release Response
   abrt                ABRT-apdu    -- Assoc. Abort
   }

AARQ-apdu ::= [APPLICATION 0] IMPLICIT SEQUENCE {
   protocol-version           [0] IMPLICIT BIT STRING {version1(0)}
                                 DEFAULT {version1},
   application-context-name   [1] Application-context-name,
   user-information           [30] IMPLICIT Association-information
   }

AARE-apdu ::= [APPLICATION 1] IMPLICIT SEQUENCE {
   protocol-version           [0] IMPLICIT BIT STRING {version1(0)}
                                 DEFAULT {version1},
   application-context-name   [1] Application-context-name,
   result                     [2] Associate-result,
   result-source-diagnostic   [3] Associate-source-diagnostic,
   user-information           [30] IMPLICIT Association-information
   }

RLRQ-apdu ::= [APPLICATION 2] IMPLICIT SEQUENCE {
   reason                     [0] IMPLICIT Release-request-reason
   }

RLRE-apdu ::= [APPLICATION 3] IMPLICIT SEQUENCE {
   reason                     [0] IMPLICIT Release-response-reason
   }

ABRT-apdu ::= [APPLICATION 4] IMPLICIT SEQUENCE {
   abort-source               [0] IMPLICIT ABRT-source
   }

ABRT-source ::= INTEGER {
   acse-service-user(0),
   acse-service-provider(1)
   }

Application-context-name ::= OBJECT IDENTIFIER
```

```
Associate-result ::= INTEGER {
   accepted(0),
   rejected-permanent(1),
   rejected-transient(2)
   }

Associate-source-diagnostic ::= CHOICE {
   acse-service-user          [1] INTEGER {
      null(0),
      no-reason-given(1),
      application-context-name-not-supported (2),
      },
   acse-service-provider      [2] INTEGER {
      null(0),
      no-reason-given(1),
      no-common-acse-version(2)
             }
   }

Association-information ::= SEQUENCE OF EXTERNAL

EXTERNAL ::= [UNIVERSAL 8] IMPLICIT SEQUENCE {
   direct-reference       OBJECT IDENTIFIER OPTIONAL,
   indirect-reference     INTEGER OPTIONAL,
   data-value-descriptor  ObjectDescriptor OPTIONAL,
   encoding               CHOICE {
      single-ASN1-type        [0]  ANY,
      octet-aligned           [1] IMPLICIT OCTET STRING,
      arbitrary               [2] IMPLICIT BIT STRING
      }
   }

Release-request-reason  ::= INTEGER { normal(0),
                    urgent(1),
                    user-defined(30)
   }

Release-response-reason ::= INTEGER {normal(0),
                    not-finished(1),
                    user-defined(30)
   }
```

E.2 ASN.1 modules

MDAP-specific ASN.1 modules pertain to Association User Information and Remote Operation Protocol and CMIP, specifically ROSE* and CMIP*.

E.2.1 ROSE*

```
MDAP-ROSE DEFINITIONS ::= BEGIN

IMPORTS:

   ROSEapdus ::= CHOICE {
      roiv-apdu     [1] IMPLICIT ROIVapdu, -- Remote Operation Invoke
```

```
   rors-apdu     [2] IMPLICIT RORSapdu, -- Remote Operation Result
   roer-apdu     [3] IMPLICIT ROERapdu, -- Remote Operation Error
   rorj-apdu     [4] IMPLICIT RORJapdu, -- Remote Operation Reject
   roliv-apdu    [5] IMPLICIT ROLIVapdu -- Linked Invoke
   }

ROIVapdu ::= SEQUENCE {
   invokeID         InvokeIDType,
   operation-value  OPERATION,
   argument         ANY DEFINED BY operation-value
   }

RORSapdu ::= SEQUENCE {
   invokeID     InvokeIDType,
   SEQUENCE {
      operation-value     OPERATION,
      result              ANY DEFINED BY operation-value
      }
   }

ROERapdu ::= SEQUENCE {
   invokeID         InvokeIDType,
   error-value      ERROR,
   parameter        ANY DEFINED BY error-value
   }

RORJapdu ::= SEQUENCE {
   invokeID         InvokeIDType,
   problem          Problem
   }

ROLIVapdu ::= SEQUENCE {
   invokeID            InvokeIDType,    -- uses a special semantic!!
   linkedID            InvokeIDType,
   operation-value     OPERATION,
   argument            ANY DEFINED BY operation-value
   }

Problem ::= INTEGER  {
   unrecognizedAPDU(0),
   mistypedAPDU(1),
   badlyStructuredAPDU(2),
   duplicateInvocation(100),
   unrecognizedOperation(101),
   mistypedArgument(102),
   resourceLimitation(103),
   initiatorReleasing(104),
   unrecognizedResultInvocation(200),
   resultResponseUnexpected(201),
   mistypedResult(202),
   unrecognizedErrorInvocation(300),
   errorResponseUnexpected(301),
   unrecognizedError(302),
   unexpectedError(303),
   mistypedErrorParameter(304)
   } (0..65535)

InvokeIDType    ::=     INTEGER (0..65535)   -- normally INTEGER
```

```
--
-- OPERATION definition contains values for CMIP*
--
   OPERATION       ::=     INTEGER {
       cmipEventReport(0),
       cmipConfirmedEventReport(1),
       cmipGet(3),
       cmipSet(4),
       cmipConfirmedSet(5),
       cmipAction(6),
       cmipConfirmedAction(7),
       cmipCreate(8),
       cmipDelete(9)
       } (0..65535)   -- normally CHOICE (INT/OID)

--
-- ERROR definition contains values for CMIP*
--
   ERROR           ::=     INTEGER {
       noSuchObjectClass(0),
       noSuchObjectInstance(1),
       accessDenied(2),
       noSuchAttribute(5),
       invalidAttributeValue(6),
       getListError(7),
       setListError(8),
       noSuchAction(9),
       processingFailure(10),
       duplicateManagedObjectInstance(11),
       noSuchEventType(13),
       noSuchArgument(14),
       invalidArgumentValue(15),
       invalidScope(16),                    -- see footnote14
       invalidObjectInstance(17),
       missingAttributeValue(18),
       classInstanceConflict(19),
       mistypedOperation(21),
       noSuchInvokeId(22),
       } 0..65535)

END
```

The following definitions are used for the various identifer fields in the messages:

```
InvokeIDType ::= INT-U16      -- normally INTEGER
OPERATION ::=INT-U16          -- normally CHOICE (INT/OID)
ERROR::= INT-U16
```

E.2.1.1 Differences from ISO/IEC ROSE

These definitions in this standard contain the following differences from ISO/IEC 9072-2:

— Invoke APDU: The invoke APDU field in ISO/IEC 9072-2 contains an optional linked identifier field, which is used for a linked response mechanism. For CMISE (and CMISE*), this mechanism is

[14]Scope implies a range of objects over which an operation applies, e.g., a single object or a set of objects contained by it. Refer to E.2.2 for further specification.

used only if the multiple-reply functional unit, which is optional, is selected at association (see 7.2.3 and the definition of MDSE association user fields in ISO/IEC 9595).

MDDL encoding rules do not permit optional elements. Therefore, the additional (nonstandard) linked invoke APDU has been defined.

— Result APDU: The SEQUENCE that contains the operation value and the result field are not optional in this standard. CMIP requires these fields to be present in its response messages.

— Reject APDU: Compared to standard ROSE, the reject APDU is simpler in this standard. Its structure is the same as for the other APDUs.

  The problem values are mapped in a single integer value instead of a choice that allows differentiating between problem areas.

— Linked invoke APDU: The linked invoke APDU is used when a response to an operation invoke results in multiple reply messages. ISO/IEC ROSE uses the normal invoke APDU with the linked identifier field set instead of this special APDU. In other words, ISO/IEC ROSE does not need the linked invoke APDU.

  However the ASN.1 encoding rules used in MDDL do not support optional structure elements because they require additional tagging information that increases parsing overhead and message sizes. Therefore, this special APDU, which contains the linked identifer if it is required, is added.

— InvokeIDType, OPERATION, ERROR: These types are mapped to constraint integer types (instead of to an unconstrained integer for invoke identifier or to a choice between object identifier and unconstrained integer for the other types).

Therefore, the main differences from the standard ROSE protocol are optional elements that are either present or absent in the ROSE* definition, dependent on whether they are used by CMIP* or not used. Operation and error definitions are simplified according to the way they are used by CMIP*. A new APDU has been defined to deal with linked replies.

No nonstandard fields are in this standard, which is an adoption of ROSE for the specific needs of POC MDC.

As a result, the ROSE header has a constant size and a fixed structure that enables efficient parsing (as long as the linked invoke APDU is not used).

E.2.1.2 ROSE* message fields

An explanation of the fields in ROSE* APDUs is given in E.2.1.2.1 through E.2.1.2.5.

E.2.1.2.1 Invoke identifier field

The invoke identifier field is a numerical field that identifies the message on the sender side. If, for example, the client sends an invoke message to the server, the server puts this invoke identifier in the result message (e.g., a GET result or an Event Report acknowledge). This mechanism allows the client to correlate the response with its request. (The invoke ID is mirrored back to the client in the response message.)

The invoker of operations has to make sure that invoke identifiers (at least the ones that are currently processed in the system) are unique.

In other words, for implementation, a receive process or receive routine could be registered with the MDDL AE on the client that is called as soon as the response message arrives.

E.2.1.2.2 Linked identifier field

In cases of multiple replies (multiple result messages) to an operation invoke, the linked identifer field contains the value of the invoke identifier of the operation invoke.

As the invoke identifier in result messages, the linked identifier allows the receiver of the result to find out which process invoked the operation.

E.2.1.2.3 Operation value field

The remote operation service user or service provider defines the operations that can be used. Each operation has an associated numerical value for identification purposes (see E.2.2 for definitions).

E.2.1.2.4 Error value field

Just like the operation value, numerical values are defined for certain error conditions. The valid error values for MDDL are defined in CMIP*. Note that the error value is not the operation value.

NOTE—This requires some explanation. Take a CMIP* GET command as an example. Suppose the managed object class field in this message is wrong. Then, a remote operation error (ROER) APDU is sent back, the error value is noSuchObjectClass. In other words, the APDU does not carry the information that this is in response to a GET request. This information can only be retrieved by examining the invoke identifier, which is a reference to the original GET request message: hence the requirement that (active) invoke identifiers in the system be unique.

E.2.1.2.5 Argument field

A remote operation invoke can be considered a (remote) procedure call. Arguments that are passed to the procedure are simply attached to the ROSE* message. In MDDL, CMIP* PDUs are arguments of ROSE* messages.

E.2.1.3 Handling of linked replies

In certain cases, an operation invoke can result in multiple reply messages. In MDDL this may happen in cases of scoped operations or in cases of operations (e.g., GET) that return large amounts of data. These multiple replies are all linked to the same invoke identifier (hence linked replies).

For linked replies, the following applies:

- For all response messages except the very last one,
  - The remote operation linked invoke (ROLIV) APDU is used.
  - The invoke identifier field is set by the responder and has a specific semantic that allows to detect missing parts.
  - The linked identifier field has the value of the invoke identifier field of the request.
- For the very last message,
  - The remote operation result (RORS) APDU is used.
  - The invoke identifer field in this response has the value of the invoke identifier field of the request.

    NOTE—If two messages are needed in the reply, the first one is a ROLIV APDU with the roliv-last bit set to 1 and with the count set to 1,
- For the ROLIV APDU, the invoke ID field has the following semantic:

```
ROLIV-Invoke-ID ::= SEQUENCE {
   state     INT-U8 {
      roliv-first(1),                -- this is the first ROLIV apdu
      roliv-not-first-not-last(2),
```



```
            roliv-last(3)                   -- this is the last ROLIV apdu, one
                                               RORS apdu will follow
            },
        countINT-U8                         -- counter starts with value 1 (in
                                               first state)
    }
```

With this definition, it is possible to detect missing parts of the complete set of reply messages.

E.2.2 CMIP*

As with ROSE*, CMISE* is part of the application layer. CMISE provides object management services that allow access to attribute values, to invoke object functions, etc. The associated protocol (CMIP) defines the application-layer messages (APDUs) that allow invocation of these services. CMISE is layered on top of ROSE to perform its function.

Therefore, CMIP messages are normally defined by using ROSE ASN.1 macros (ISO/IEC 9596-1). For better understanding, this subclause does not use the ROSE macro notation. The ASN.1 definitions used here can be considered macro expansions. However, as explained, the message definitions in here are not fully compliant to the standard CMIP. While all data fields that are sent in CMIP* APDUs exist in standard CMIP definitions as well, the data types were changed to simplify definitions.

The CMIP* message (more precise: the CMIP* operation argument data structure) is simply attached as an argument field to a ROSE* APDU. The ROSE* operation value field must be interpreted to determine the type of attached argument.

Table E.1 shows which argument types (message types) are defined or used in CMIP*.

— The invoke data type is appended to a remote operation invoke (ROIV) APDU.
— The response data type is appended to an RORS APDU.
— Error messages in ROER APDUS are handled differently.

Table E.1—CMIP* argument types

Message type	Operation value	Invoke data type	Response data type
Event Report	0	EventReportArgument	—
Confirmed Event Report	1	EventReportArgument	EventReportResult
Get (always confirmed)	3	GetArgument	GetResult
Set	4	SetArgument	—
Confirmed Set	5	SetArgument	SetResult
Action	6	ActionArgument	—
Confirmed Action	7	ActionArgument	ActionResult
Create (always confirmed)	8	CreateArgument	CreateResult
Delete (always confirmed)	9	DeleteArgument	DeleteResult

The operation values conform to ISO/IEC 9596-1.

The operation value in Table E.1 is the value that is assigned to the operation value field in the ROSE* invoke and response APDUs. (The linked reply operation is not required here.)

If, for example, the host system requests an attribute value of some object from the server, it sends a ROSE* ROIV APDU with operation value 3 and appended GetArgument. The server responds with an RORS APDU with operation value 3 and appended GetResult. Therefore, both the ROSE* PDU type and the operation value are needed to determine the appended data type (i.e., CMIP* message type).

In case of an error, a ROSE* ROER PDU is sent as a response, and the error value field contains an error code (see definitions below in this subclause). Optionally, if the length field of the ANY DEFINED BY is > 0, additional information is provided.

All CMIP* invoke and response (and error) messages are defined below in this subclause. The definitions are derived from ISO/IEC 9596-1, with differences as explained.

The data types that are appended to a ROSE* APDU (i.e., invoke, result, or error) are in bold font in the following ASN.1 definitions:

```
MDAP-CMIP DEFINITIONS ::= BEGIN

-- The following are commonly used syntax types; it may be useful to
-- define these in a separate module and IMPORT them, as e.g., MDDL-TYPES
-- Begin MDDL-TYPES

    RelativeTime ::= INT-U32   -- 32 bit integer type

    OID-Type ::= INT-U16       -- 16 bit integer type

    AVA-Type ::= SEQUENCE {
        attribute-id        OID-Type,
        attribute-value     ANY DEFINED BY attribute-id
    }

    AttributeList ::= SEQUENCE OF AVA-Type

    AttributeIdList ::= SEQUENCE OF OID-Type

    ManagedObjectId ::= SEQUENCE {
        m-obj-class         OID-Type,
        m-obj-inst          GLB-HANDLE
    }

-- End MDDL-TYPES

    EventReportArgument  ::= SEQUENCE {
        managedObject           ManagedObjectId,
        eventTime               RelativeTime,
        eventType               OID-Type,
        eventInfo               ANY DEFINED BY eventType
        }

    EventReportResult ::= SEQUENCE {
        managedObject           ManagedObjectId,
        currentTime             RelativeTime,
        eventType               OID-Type,
        eventReplyInfo          ANY DEFINED BY eventType
```



```
    --NOTE:  each object class should define specific eventType format;
       as a result,
    --It is not specified in this document.  For the DIM, refer to
       individual object class
    --"ReplyInfo" definition.
    }

GetArgument  ::= SEQUENCE {
    managedObject          ManagedObjectId,
    scope                  Scope,
    attributeIdList        AttributeIdList
    }

GetResult ::= SEQUENCE {
    managedObject          ManagedObjectId,
    attributeList          AttributeList
    }

GetError ::= SEQUENCE {
    errorStatus            ErrorStatus,
    attributeId            OID-Type
    }

GetListError ::= SEQUENCE {
    managedObject          ManagedObjectId,
    getInfoList            SEQUENCE OF GetError
    }

ModifyOperator ::= INTEGER {
    replace(0),
    addValues(1),
    removeValues(2),
    setToDefault(3)
    } (0..65535)

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

ModificationList ::= SEQUENCE OF AttributeModEntry

SetArgument  ::= SEQUENCE {
    managedObject          ManagedObjectId,
    scope                  Scope,
    modificationList       ModificationList
    }

SetResult ::= SEQUENCE {
    managedObject          ManagedObjectId,
    attributeList          AttributeList
    }

SetError ::= SEQUENCE {
    errorStatus            ErrorStatus,
    modifyOperator         ModifyOperator,
    attributeId            OID-Type
    }
```

```
SetListError ::= SEQUENCE {
   managedObject          ManagedObjectId,
   setInfoList            SEQUENCE OF SetError
   }

ActionArgument  ::= SEQUENCE {
   managedObject          ManagedObjectId,
   scope                  Scope,
   actionInfo             ActionInfo
   }

ActionInfo ::= SEQUENCE {
   actionType             OID-Type,
   actionInfoArgs         ANY DEFINED BY actionType
   }

ActionResult ::= SEQUENCE {
   managedObject          ManagedObjectId,
   actionReply            ActionReply
   }

ActionReply ::= SEQUENCE {
   actionType             OID-Type,
   actionInfoArgs         ANY DEFINED BY actionType
   }

CreateArgument ::= SEQUENCE {
   managedObjectClass     OID-Type,
   superiorManagedObject  ManagedObjectId,
   attributeList          AttributeList
   }

CreateResult ::= SEQUENCE {
   managedObject          ManagedObjectId,
   attributeList          AttributeList
   }

DeleteArgument ::= SEQUENCE {
   managedObject          ManagedObjectId,
   scope                  Scope
   }

DeleteResult ::= SEQUENCE {
   managedObject          ManagedObjectId
   }

Scope ::= INTEGER { base-object(0) } (0.. 4294967295)-- see footnote[15]

NoSuchAction ::= SEQUENCE {
   managedObjectClass     OID-Type,
   actionType             OID-Type
   }
```

[15]At a minimum, the base-object scope shall be supported, but application profiles leveraging this standard may define additional scopes as appropriate.

```
    NoSuchArgument ::= SEQUENCE {
        managedObjectClass      OID-Type,
        eventType               OID-Type
        }

    NoSuchEventType ::= SEQUENCE {
        managedObjectClass      OID-Type,
        eventType               OID-Type
        }

    ErrorStatus     ::=  INTEGER {
            attr-accessDenied(2),             -- GET, SET
            attr-noSuchAttribute(5),          -- GET, SET
            attr-invalidAttributeValue(6),    -- SET
            attr-invalidOperation(24),        -- SET
            attr-invalidOperator(25)          -- SET
            } (0..65535)

    ProcessingFailure ::= SEQUENCE {
        error-id                OID-Type,    -- use MDC
        error-info              ANY DEFINED BY error-id
        }
END
```

E.2.3 Associate user information

User information is appended to the association request and response messages. For MDSE, the definition of these user fields is kept to a minimum due to the difficult encoding and processing of ACSE messages. Note that the MDSE user fields are encoded in the negotiated transfer syntax (MDER), not in BER.

The user information for MDSE is defined as follows (in the form of an ASN.1 module):

```
MDSEUserInfo ::= SEQUENCE {
    protocolVersion         ProtocolVersion,
    nomenclatureVersion     NomenclatureVersion,
    functionalUnits         FunctionalUnits,
    systemType              SystemType,
    startupMode             StartupMode,
    optionList              AttributeList,   -- this field is reserved for
                                                future extensibility
    supportedAProfiles      AttributeList    -- contains Profile Support
                                                attributes
}

ProtocolVersion  ::=BITS-32       -- values reference a specific MDAP
                                     standard

NomenclatureVersion ::= BITS-32   -- values reference a specific
                                     nomenclature standard

FunctionalUnits ::= BITS-32 {
    extendedObjectSelection (0),  -- supports scope fields other than just
                                     base-object
                                  -- Bit 1 reserved [filter(1)]
    multipleReply(2)              -- supports multiple linked replies
                                  -- Bit 3 reserved [extendedService(3)]
                                  -- Bit 4 reserved [cancelGet(4)]
    }
```



```
SystemType ::= BITS-32 {
    sys-type-manager(0),
    sys-type-agent(8)
    }

StartupMode ::= BITS-32 {
    hot-start(0),
    warm-start(1),
    cold-start(2)
    }

END
```

The usual rules of extensibility apply: unknown tags shall be ignored, unknown bits in a bit string shall be ignored. If additional attributes or additional fields cause incompatibilities, the protocol version shall be changed.

NOTES

1—If no bits are set in the protocol version field or the nomenclature version field, then it is implied that the actual versions used are coded in the optionList field. This use of the optionList field allows for future extensibility when all version bits are assigned.

2—The nomenclature version here needs to be changed only if any nomenclature codes are modified that are used in the user information field or in the initial MDS create event message. Other versions (minor version changes) are coded in the corresponding MDS object attribute.

3—This ACSE user information references the MDER presentation context defined in the presentation context definition and result list of the ACSE messages. The user information field does not use BER.

4—Attribute identifiers in the optionList and supportedAProfiles list structures are defined in the infrastructure elements nomenclature table.

E.3 Abstract syntax extensions

Most of the abstract syntax specific to medical devices is defined in MDDL in the object-oriented DIM. By convention, object classes, attributes, attribute groups, notifications, actions, and other nomenclature are defined in MDDL common nomenclature (ISO/IEEE 11073-10101).

However, ISO association control presentation context identification requires a single object identifier for abstract syntax mapping. As a result, extensions to the MDAP object identifier for abstract syntax are allocated to MDDL, specifically 16 bit context-sensitive and 32 bit context-free partitions.

E.4 MIB definitions

MIB definitions include object identifier extensions and abstract syntax representing the content of the information.

Object identifiers for MIB extensions are allocated from asn1Modules(2) mib(4), as described in Figure B.2. Nomenclature is then appended to the mib(4) arc as defined in the ISO/IEEE 11073-10101 nomenclature communication infrastructure tables.

Annex F

(informative)

PDU examples

This annex provides an exposition of PDUs in terms of both abstract and concrete syntax (encodings) to facilitate understanding and implementation.

The first examples deal with ISO association phase PDUs, which are based on ASN.1 and BER (see F.1). Additional examples deal with configuration and operation phase PDUs (see F.2 and F.3, respectively). Abstract syntax detail is contained in the DIM, and encoding rule (MDER) detail is contained in Annex A.

Two types of PDU formats are used in this annex, as follows:

- The first format provides a decomposition of abstract syntax, the related encoding (hex format), and relevant notes. For an example of this format, refer to Figure F.1 through Figure F.5.
- The second format, for brevity, does not provide abstract syntax, but provides simply annotated encodings. Refer to the DIM for details of abstract syntax in these cases. For an example of this format, refer to Figure F.6 and Figure F.7, which show object create notification event report and object create notification event report response PDU examples, respectively.

Examples have been adapted from infusion and ventilation medical device demonstration projects. To the extent practical, field values are accurate, although some tradeoffs are necessary to present information in this format. In particular, ISO/IEEE 11073 arc derivations are shown based on the listing in Annex B, but hex listings are not shown.

F.1 Association

Association examples include association request and response (see Figure F.1 and Figure F.2, respectively), association release request and release response (see Figure F.3 and Figure F.4, respectively), and association abort (see Figure F.5).

F.2 Configuration

Examples include MDS object create notification event report and object create notification event report response (refer to Figure F.6 and Figure F.7, respectively).

F.3 Operation

Examples include periodic scanner event report and buffered scanner event report (refer to Figure F.8 and Figure F.9, respectively)


```
STRUCTURED DECOMPOSITION
Structure                                                    Hex encoding
-- Session Connect PDU
Session Header  (CN SPDU)                                    0d de          -- SI='0d'==13.(CN), LI = length in octets='de'==222.
                                                             05 08          -- PGI='05'==5.(connect/accept item), LI='08'==8.
                                                             13 01 00       -- PI='13'==19.(options), LI=1, value = 0
                                                             16 01 02       -- PI='16'==22.(version), LI=1, value = 2
                                                             80 00          -- PI='80'==128.(enable MDAP Session Extensions), LI=0
                                                                            --   Note: ➔ defaults to no coalescing. To enable Coalescing add ➔ 81 01 xx,
                                                                            --         where xx ➔ coalescing period (see pdu definition)
                                                             14 02 00 02    -- PI=20 (user requirements),
                                                                            --   LI=2, value:  select full duplex functional unit
                                                             c1 ce          -- PGI='c1'==193. (user data), LI= length of  user data ='ce' == 206.
--  Connect Presentation – Presentation PDU
CP-PPDU ::= SET {                                       (1)  31 80          -- [CONSTRUCTED]+Tag #17
  -- mode-selector
  [0] IMPLICIT Mode-selector,                           (2)  a0 80          -- [context- specific +CONSTRUCTED]+ Tag #0 “[0]”
     ::= SET {mode-value [0]
         IMPLICIT INTEGER { normal-mode(1)                   80 01 01       -- must be "normal mode" (i.e., the only mode supported by MDDL)
             }                                          (2)  00 00          -- end Mode-selector
  -- normal-mode-parameters
  [2] IMPLICIT SEQUENCE{                                (3)  a2 80          -- [context- specific +CONSTRUCTED] + Tag #2  “[2]”
     -- protocol-version
     [0] IMPLICIT Protocol-version ::= BIT STRING { version-mdap(15) }  a0 03 00 00 01  -- Must be version-mdap; see E.1.2.7
                                                                            -- [context-specific+CONSTRUCTED]+Tag #0 “[0]” + len + unused bits + BITS-16
     -- presentation-context-definition-list
     [4] IMPLICIT Presentation-context-definition-list := Context-list  (4)  a4 80  -- [context-specific+CONSTRUCTED] + Tag #4  “[4]”
         ::= SEQUENCE OF
          -- Presentation Context Definition #1
             SEQUENCE                                   (5)  30 80          -- Presentation Context #1:  ISO ACSE ASN.1 / ISO BER
                 { Presentation-context-identifier::= INTEGER     02 01 01
                  Abstract-syntax-name ::= OBJECT IDENTIFIER      06 04     -- ISO ACSE version1
                   { joint-iso-ccitt association-control(2)       52 01 00 01
                      abstract-syntax(1) apdus(0) version1(1) }
                  SEQUENCE OF                           (6)  30 80
                   Transfer-syntax-name ::= OBJECT IDENTIFIER    06 02 51 01  -- ISO BER
                  }                                     (6)  00 00
             }                                          (5)  00 00          -- end Presentation Context Definition #1
```

NOTE—This example is based on a ventilator medical device (agent) association request.

Figure F.1—Association request formatting example

```
        -- Presentation Context Definition #2
            SEQUENCE                                    (7)   30 80          --  Presentation Context #2:  IEEE 1073 MDDL/ IEEE 1073 MDER
                { Presentation-context-identifier::= INTEGER  02 01 02
                 Abstract-syntax-name ::= OBJECT IDENTIFIER   06 0c          -- Abstract Syntax: MDDL 16-bit Nomenclature iso(1) member-body(2) US(840)
                                                                             -- ieee1073(10004) mdap(2) version1(1) mdap-0(0) standardSpecificExt(0) modules(0)
                                                                             -- abstractSyntax(1)  nomen16(1)
                                                                             --  iso(1) X 40 + member-body(2) = 40 = 0x2A; US(840) = 0x348 = 11 0100 1000 => 0x80 +
                                                                             -- 110, 100 1000 = 0x86 0x48
                                                              2A 86 48 CE 14 02 01 00 00 00 01 01
                 SEQUENCE OF                            (8)   30 80
                Transfer-syntax-name::= OBJECT IDENTIFIER     06 0c          -- Transfer Syntax:  MDER iso(1) member-body(2) US(840) ieee1073(10004) mdap(2)
                                                                             -- version1(1) mdap-0(0) standardSpecificExt(0) modules(0) transferSyntax(2)
                                                                             -- mderBigEndian(1)
                                                              2A 86 48 CE 14 02 01 00 00 00 02 01
                 }                                      (8)   00 00
            }                                           (7)   00 00          -- end Presentation Context Definition #2
        }                                               (4)   00 00          -- end Presentation Context Definition List
-- (normal mode) User Data
User-data ::= CHOICE { [APPLICATION 1] IMPLICIT Fully-encoded-data} (9)  61 80   -- Presentation-layer user data
::= SEQUENCE OF PDV-list
        ::= SEQUENCE {                                  (10)  30 80
          Presentation-context-identifier ::= INTEGER         02 01 01       -- Use Presentation Context #1 to encode this User-data (i.e., BER encoding)
          CHOICE { single-ASN.1-type[0] ANY             (11)  a0 80
            { AARQ-apdu                                                      -- PPDV User-data == AARQ (ACSE Request)
            ::= [APPLICATION 0] IMPLICIT SEQUENCE {     (12)  60 80
                protocol-version [0] IMPLICIT BIT STRING { version-1(0) }    -- DEFAULT {version-1}, omitted
                application-context-name [1] Application-context-name
                                                        (13)  a1 80
                            ::= OBJECT IDENTIFIER             06 0c          -- Application Context Normal Operation Mode iso(1) member-body(2) US(840)
                                                                             -- ieee1073(10004) mdap(2) version1(1) mdap-0(0) standardSpecificExt(0) modules(0)
                                                                             -- applicationContext(3) normalMode(1)
                                                              2A 86 48 CE 14 02 01 00 00 00 03 01
                                                        (13)  00 00
                user-information [30] IMPLICIT Association-information
                                                        (14)  be 80          -- context-specific + CONSTRUCTED + Tag #30  "[30]"
                ::= SEQUENCE OF EXTERNAL
                    EXTERNAL                            (15)  28 80          -- CONSTRUCTED + Tag #8  "EXTERNAL"†
                    ::= IMPLICIT SEQUENCE {                                  -- MDDL a-Associate User Info – Encoded using Presentation Context #2 (MDER)
```

† The EXTERNAL type provides that information that is necessary for the message receiver to be able to properly switch presentation contexts and properly process the subsequent information, which in this case is encoded using MDER.

Figure F.1—Association request formatting example *(continued)*

```
                  direct-reference ::= OBJECT IDENTIFIER,   06 0c           -- Object Identifier for MDAP Presentation Context iso(1) member-body(2) US(840)
                                                                            -- ieee1073(10004) mdap(2) version1(1) mdap-0(0) standardSpecificExt(0) modules(0)
                                                                            -- transferSyntax(2) mderBigEndian(1)
                                                            2A 86 48 CE 14 02 01 00 00 00 02 01
                  indirect-reference ::= INTEGER            02 01 02        -- Presentation context ID‡ for MDAP (per X.209 34 Note 2)
                  encoding ::= CHOICE { [1] IMPLICIT OCTET STRING
                                                            81 3A
        -- MDDL User Information
        MDSEUserInfo ::= SEQUENCE {                                         -- NOTE:  the remainder is in the selected encoding rules, e.g., MDER
             protocolVersion [0] IMPLICIT ProtocolVersion
                  ::= BITS-32 {  mddl-version1 (0), }       80 00 00 00     -- Protocol Version 1 [some higher-order bits used for minor version ctl]
             nomenclatureVersion  [1] IMPLICIT NomenclatureVersion
                  ::= BITS-32 {  nomen-version1(1), }       40 00 00 00     -- Nomenclature Version 1
             functionalUnits [2] IMPLICIT FunctionalUnits
                  ::= BITS-32 {extendedObjectSelection(0) } 00 00 00 00     -- Extended object selection function not used
             systemType [3] IMPLICIT SystemType
                  ::= BITS-32 { sys-type-manager(0), }      80 00 00 00     -- [Medical data] Server
             startupMode [4] IMPLICIT StartupMode
                  ::= BITS-32 { cold-start(2), }            20 00 00 00     -- Device implementation specific (for Demo … fixed at Cold Start)
-- INTEROP extensions
--    optionList    AttributeList,                          00 00 00 00     -- ZERO AttributeList
    --    supportedAProfiles    AttributeList
        ::= SEQUENCE OF  AVA-Type                           00 01 00 1E     -- One  Profile Identified, Length = 30
                                                            00 02 00 1A     -- OID-Type == NOM_BASELINE_PROFILE_SUPPORT  (Infrastructure Partition)
                                                            80 00 00 00     -- BaselineRevision::baseline-rev-0(0)
                                                            00 00 00 00     -- max-mtu-rx ~  max. receive message size in bytes
                                                            00 00 00 00     -- max-mtu-tx ~ max. transmit message size in bytes
                                                            FF FF FF FF     -- max-bw-tx ~ max. transmit bandwidth (0xFFFFFFFF=unspecified)
                                                            00 01           -- max-mds-hierarchy ~ max. depth of MDS object hierarchy,
                                                                            --     from MDIB root to deepest MDS
                                                            00 00 00 00     -- BaselineOptions ~No dynamic create/deleteobjects
                                                                            -- FUTURE:  Will Add Patient Demographics Opt. Pkg.
                                                            00 00 00 00     -- optional-packages ~ ZERO AttributeList
                                                                            -- Note:  Will have to add Patient Demographics opt pkg later.
                                                       (15) 00 00           (14)  00 00
                                                       (12) 00 00           (11)  00 00
                                                       (10) 00 00           (9)   00 00
                                                       (3)  00 00           (1)   00 00
```

‡ This value is included in the presentation context identifier field of all MDDL messages.

Figure F.1—Association request formatting example *(continued)*

STRUCTURED DECOMPOSITION

```
Structure                                                   Hex encoding
-- Session Accept PDU
Session Header      (AC SPDU)                               0e c0            -- SI='0e'==13.(CN), LI = length in octets='c0'==192.
                                                            05 08            -- PGI='05'==5.(connect/accept item), LI='08'==8.
                                                            13 01 00         -- PI='13'==19.(options), LI=1, value = 0
                                                            16 01 02         -- PI='16'==22.(version), LI=1, value = 2
                                                            80 00            -- PI='80'==128.(MDAP Session Extensions), LI=0
                                                                             --   Note: ➔ defaults to no coalescing. To confirm enabling of coallescence
                                                                             --     add ➔ 81 01 xx, where xx ➔ coalescing period (see pdu definition)
                                                            14 02 00 02      -- PI=20 (user requirements), LI=2, value:  select full duplex functional unit
                                                            c1 b0            -- PGI='c1'==193. (user data), LI= length of  user data ='b0' == 176.
 [Session user data == Presentation Connect Accept PDU]
  CPA-PPDU ::= SET {                                   (1)  31 80            -- [CONSTRUCTED]+Tag #17
   -- mode-selector
   [0] IMPLICIT Mode-selector                          (2)  a0 80            -- [context- specific +CONSTRUCTED]+ Tag #0 "[0]"
     ::= SET {mode-value [0]
         IMPLICIT INTEGER { normal-mode(1) }                80 01 01         -- must be "normal mode" (i.e., the only mode supported by MDDL)
             }                                         (2)  00 00            -- end Mode-selector
   -- normal-mode-parameters
   [2] IMPLICIT SEQUENCE{                              (3)  a2 80            -- [context- specific +CONSTRUCTED] + Tag #2  "[2]"
      -- protocol-version
      [0] IMPLICIT Protocol-version ::= BIT STRING { version-mdap(15) }    A0 03 00 00 01   -- Version must be version-mdap; see E.1.2.7
      -- presentation-context-definition-result-list
      [5] IMPLICIT Presentation-context-definition-result-list  Result-list  (4)  a5 80   -- [context-specific+CONSTRUCTED] + Tag #5  "[5]"
         ::= SEQUENCE OF
        -- Presentation Context Definition Result #1
             SEQUENCE {                                (5)  30 80
        [0] IMPLICIT Result ::= INTEGER { acceptance(0) }   80 01 00         -- Presentation Context #1 ACCEPT!
        [1] IMPLICIT Transfer-syntax-name OPTIONAL,
          ::= OBJECT IDENTIFIER                             81 02 51 01      -- ISO BER
                                                       (5)  00 00            -- end Presentation Context Definition Result #1
        -- Presentation Context Definition Result #2
             SEQUENCE {                                (6)  30 80
        [0] IMPLICIT Result ::= INTEGER { acceptance(0) }   80 01 00         -- Presentation Context #2 ACCEPT!
        [1] IMPLICIT Transfer-syntax-name OPTIONAL,
        ::= OBJECT IDENTIFIER                               81 0c            -- IEEE 1073 MDER Big Endian iso(1) member-body(2) US(840) ieee1073(10004)
                                                                             -- mdap(2) version1(1) mdap-0(0) standardSpecificExt(0) modules(0) transferSyntax(2)
                                                                             -- mderBigEndian(1)
                                                            2A 86 48 CE 14 02 01 00 00 00 02 01
```

Figure F.2—Association response formatting example

```
                }                                              (6)   00 00         -- end Presentation Context Definition Result #1
        }                                                      (4)   00 00         -- end Presentation Context Definition Result List
-- (normal mode) User Data
User-data ::= CHOICE { [APPLICATION 1] IMPLICIT Fully-encoded-data}  (7)  61 80   -- Presentation-layer user data
        ::= SEQUENCE OF PDV-list
        ::= SEQUENCE {                                         (8)   30 80
                presentation-context-identifier ::= INTEGER          02 01 01      -- ACSE Presentation Context t #1
                CHOICE { single-ASN.1-type[0] ANY              (9)   a0 80
   AARE-apdu ::=                                                                   -- PPDV User-data == AARE (ACSE Response)
    [APPLICATION 1] IMPLICIT SEQUENCE {                        (10)  61 80
      protocol-version [0] IMPLICIT BIT STRING { version-1(0) }                    -- DEFAULT {version-1}, omitted
      application-context-name  [1] Application-context-name   (11)  a1 80
                                    ::= OBJECT IDENTIFIER      06 0c               -- Application Context Normal Operation Mode iso(1) member-body(2) US(840)
                                                                                   -- ieee1073(10004) mdap(2) version1(1) mdap-0(0) standardSpecificExt(0) modules(0)
                                                                                   -- applicationContext(3) normalMode(1)
                                                               2A 86 48 CE 14 02 01 00 00 00 03 01
                                                          (11) 00 00
   result [2] Associate-result ::= INTEGER { accepted(0)       a2 03 02 01 00      -- context-specific+CONSTRUCTED+ Tag #2, INTEGER TLV
        result-source-diagnostic [3] Associate-source-diagnostic ::= CHOICE {
                                                               a3 05               -- context-specific+CONSTRUCTED+Tag #3, len=5
        acse-service-user [1] INTEGER { null(0) }              a1 03 02 01 00      -- context-specific+CONSTRUCTED+
        user-information  [30] IMPLICIT Association-information (12) be 80         -- context-specific + CONSTRUCTED + Tag #30  "[30]"
        ::= SEQUENCE OF EXTERNAL
                EXTERNAL                                  (13) 28 80               -- CONSTRUCTED + Tag #8  "EXTERNAL"
                ::=IMPLICIT SEQUENCE {
                        direct-reference ::= OBJECT IDENTIFIER                     -- OPTIONAL, omitted
                        indirect-reference ::= INTEGER         02 01 02            -- Presentation context ID†, MDER Big Endian
                        encoding ::= CHOICE { [1] IMPLICIT OCTET STRING
                                                               81 3A
                                                                                   -- NOTE:  the presentation context and format for this
                                                                                   -- have been "accepted", so what follows is "MDDL/MDER"
                                MDSEUserInfo ::= SEQUENCE {
                                        protocolVersion [0] IMPLICIT ProtocolVersion
                                        ::= BIT STRING { mddl-version1 (0) }   80 00 00 00   -- Protocol Version 1
                                        nomenclatureVersion  [1] IMPLICIT NomenclatureVersion
                                        ::= BIT STRING { nomen-version1(1) }   40 00 00 00   -- Nomenclature Version 1
                                        functionalUnits [2] IMPLICIT FunctionalUnits
                                        ::= BIT STRING { extendedObjectSelection(0) }
                                                               00 00 00 00         -- Extended object selection function not used
```

† This value is included in the presentation context identifier field of all MDDL messages.

Figure F.2—Association response formatting example *(continued)*

```
                  systemType [3] IMPLICIT SystemType
                  ::= BIT STRING { sys-type-agent(8), }   00 80 00 00      -- [Medical data] Agent
                  startupMode [4] IMPLICIT StartupMode
                  ::= BIT STRING  { cold-start(2) }       20 00 00 00      -- Device implementation specific (for Demo … fixed at Cold Start)
-- INTEROP extensions
--   optionList    AttributeList,                         00 00 00 00      -- ZERO AttributeList
--   supportedAProfiles    AttributeList
       ::= SEQUENCE OF  AVA-Type                          00 01 00 1E      -- One  Profile Identified, Length = 30
                                                          00 02 00 1A      -- OID-Type == NOM_BASELINE_PROFILE_SUPPORT  (Infrastructure Partition)
                                                          80 00 00 00      -- BaselineRevision::baseline-rev-0(0)
                                                          00 00 00 00      -- max-mtu-rx ~  max. receive message size in bytes
                                                          00 00 00 00      -- max-mtu-tx ~ max. transmit message size in bytes
                                                          FF FF FF FF      -- max-bw-tx ~ max. transmit bandwidth (0xFFFFFFFF=unspecified)
                                                          00 01            -- max-mds-hierarchy ~ max. depth of MDS object hierarchy,
                                                                           --     from MDIB root to deepest MDS
                                                          00 00 00 00      -- BaselineOptions ~No dynamic create/deleteobjects
                                                                           -- FUTURE:  Will Add Patient Demographics Opt. Pkg.
                                                          00 00 00 00      -- optional-packages ~ ZERO AttributeList
                                                                           -- Note:  Will need to add Patient Demographics package later
                                                    (13)  00 00            (12)  00 00
                                                    (10)  00 00            (9)   00 00
                                                    (8)   00 00            (3)   00 00
                                                    (2)   00 00            (1)   00 00
```

Figure F.2—Association response formatting example *(continued)*

STRUCTURED DECOMPOSITION

Structure		Hex encoding	
-- Session Finish PDU			
Session Header (FINISH SPDU)		09 18	-- SI='09'==9.(FN), LI = length in octets='18'==24.
		c1 16	-- PGI='c1'==193. (user data), LI= length of user data ='16' == 22.
-- Presentation PDU			
User-data ::= CHOICE [APPLICATION 0] IMPLICIT Fully-encoded-data	(1)	61 80	
SEQUENCE OF PDV-list	(2)	30 80	
Presentation-context-identifier		02 01 01	
presentation-data-values ::= CHOICE single-ASN.1-type [0] ANY	(3)	A0 80	
ACSE-apdu ::= CHOICE RLRQ-apdu ::= [APPLICATION 2] IMPL SEQ(4)		62 80	
Reason [0] IMPLICIT Release-request-reason normal [0]		80 01 00	
	(4)	00 00	
	(3)	00 00	
	(2)	00 00	
	(1)	00 00	

Figure F.3—Association release request formatting example

STRUCTURED DECOMPOSITION

Structure		Hex encoding	
-- Session Disconnect PDU			
Session Header (DISCONNECT SPDU)		0a 18	-- SI='0a'==10.(DISCONNECT), LI = length in octets='18'==24.
		c1 16	-- PGI='c1'==193. (user data), LI= length of user data ='16' == 22.
-- Presentation PDU			
User-data ::= CHOICE [APPLICATION 0] IMPLICIT Fully-encoded-data	(1)	61 80	
SEQUENCE OF PDV-list	(2)	30 80	
Presentation-context-identifier		02 01 01	
presentation-data-values ::= CHOICE single-ASN.1-type [0] ANY	(3)	A0 80	
ACSE-apdu ::= CHOICE RLRE-apdu ::= [APPLICATION 3] IMPL SEQ(4)		63 80	
Reason [0] IMPLICIT Release-request-reason normal [0]		80 01 00	
	(4)	00 00	
	(3)	00 00	
	(2)	00 00	
	(1)	00 00	

Figure F.4—Association release response formatting example

STRUCTURED DECOMPOSITION

Structure		Hex encoding	
-- Session Abort PDU			
Session Header (ABORT SPDU)		19 03	-- SI='19'==25.(DISCONNECT), LI = length in octets='03'==3.
Transport Disconnect		11 01 09	-- bit 4: no reason, bit 1: transport connection is released

STRUCTURED DECOMPOSITION

Structure		Hex encoding	
-- Session Abort PDU			
Session Header (ABORT SPDU)		19 2E	-- SI='19'==25.(DISCONNECT), LI = length in octets='2E'==46.
Transport Disconnect		11 01 03	-- bit 2: user abort, bit 1: transport connection is released
		c1 29	-- PGI='c1'==193. (user data), LI= length of user data ='29' == 41
Abort-type ::= CHOICE {			
ARU-PPDU ::= CHOICE [0] IMPLICIT SEQUENCE	(1)	A0 80	
[0] IMPLICIT Presentation-context-identifier-list	(2)	A0 80	
Presentation-context-identifier-list ::= SEQUENCE OF SEQUENCE	(3)	30 80	
Presentation-context-identifier		02 01 01	-- ACSE PCID
Transfer-syntax-name		06 02 51 01	-- ACSE Transfer Syntax OID
	(3)	00 00	
	(2)	00 00	
User-data ::= CHOICE [APPLICATION 0] IMPLICIT Fully-encoded-data(4)		61 80	
SEQUENCE OF PDV-list	(5)	30 80	
Presentation-context-identifier		02 01 01	-- ACSE PCID
presentation-data-values ::= CHOICE single-ASN.1-type [0] ANY (6)		A0 80	
ACSE-apdu ::= CHOICE ABRT-apdu ::= [APPLICATION 4] IMPL SEQ(7)		64 80	
abort-source [0] IMPLICIT ABRT-source acse-service-provider (1)		80 01 01	
	(7)	00 00	
	(6)	00 00	
	(5)	00 00	
	(4)	00 00	
	(1)	00 00	

Figure F.5—Association abort formatting example

```
// PDU Header:
0xE1, 0x00,                        // MDAP_SPDU_SI
0x00, 0x02,                        // Presentation Context ID
// -- ROSE Section
0x00, 0x01, 0x00, 0xA8,            // Invoke (ROIVapdu), length
0x00, 0x01, 0x00, 0x01, 0x00, 0xA2,// Invoke ID=1,cmipConfirmedEventReport(1), length
// -- CMDIP Section (EventReportArgument)
0x00, 0x24, 0x00, 0x00, 0x00, 0x01,// NOM_MOC_VMS_MDS_HYD(36), Context=0, Handle=1
0x00, 0x00, 0x00, 0x00,            // Relative Time
0x0D, 0x06, 0x00, 0x94,            // NOM_NOTI_MDS_CREAT(3334), length

// MDS::MdsCreateInfo Section
0x00, 0x24, 0x00, 0x00, 0x00, 0x01,// MDS MgdObjId
0x00, 0x09, 0x00, 0x8A,            // AttributeList (SEQ OF 9 AVA), length
// -- MDS Attribute 1:  System-Type
0x09, 0x86, 0x00, 0x04,            // NOM_ATTR_SYS_TYPE, length
0x00, 0x01, 0x11, 0x61,            // Partition(1), NOM_DEV_PUMP_INFUS_MDS (4449)
// -- MDS Attribute 2:  System-Model { manufacturer; model-number; }
0x09, 0x28, 0x00, 0x3C,            // NOM_ATTR_ID_MODEL(2344), length
// Note:  Text strings are in UTF-16
0x00, 0x24,                        // manufacturer="Baxter Healthcare"
       0x00,'B', 0x00,'a', 0x00,'x', 0x00,'t', 0x00,'e', 0x00,'r', 0x00,' ',
       0x00,'H', 0x00,'e', 0x00,'a', 0x00,'l', 0x00,'t', 0x00,'h', 0x00,'c',
       0x00,'a', 0x00,'r', 0x00,'e',
     0x00, 0x00,                   // <zero terminate + pad to even byte count>
0x00, 0x14,                        // model-number="Colleague"
       0x00,'C', 0x00,'o', 0x00,'l', 0x00,'l', 0x00,'e', 0x00,'a', 0x00,'g',
      0x00,'u', 0x00,'e',
          0x00, 0x00,              // <zero terminate + pad to even length>
// -- MDS Attribute 3:  System-Id
0x09, 0x84, 0x00, 0x0A,            // NOM_ATTR_SYS_ID(2436), length
0x00, 0x08,                        // OCTET STRING length
0x00, 0x00, 0x00,                  // company-id ("00-00-00-00-00-00-00-00")
0x00, 0x00, 0x00, 0x00, 0x00,      // serial number (not serialized)
// -- MDS Attribute 4:  Compatibility-Id
0x09, 0x20, 0x00, 0x04,            // NOM_ATTR_ID_COMPAT, length
0x00, 0x00, 0x00, 0x00,            // INT-U32 [default=ZERO]
// -- MDS Attribute 5:  Nomenclature-Version
0x09, 0x48, 0x00, 0x04,            // NOM_ATTR_NOM_VERS
0x00, 0x01, 0x00, 0x00,            // Version = 1.0
// -- MDS Attribute 6:  System-Capability
0x09, 0x83, 0x00, 0x04,            // NOM_ATTR_SYS_CAPAB, length
0x18, 0x00, 0x00, 0x00,            // BITS-32 [Auto-init+auto-update scanlist]
// -- MDS Attribute 7:  System-Specification
0x09, 0x85, 0x00, 0x08,            // NOM_ATTR_SYS_SPECN (2437), length
0x01, 0x01,                        // NOM_MED_DEV_SPEC_STD_SUPPORT (257)
0x00, 0x01, 0x00, 0x02,            // SEQUENCE OF 1 OID-Type, length
0x10, 0x01,                        // NOM_DEV_SPEC_PROFILE_INFUS (4097)
// -- MDS Attribute 8:  Mds-Status
0x09, 0xA7, 0x00, 0x02,            // NOM_ATTR_VMS_MDS_STAT (2471), length
0x00, 0x04,                        // MDSStatus= configuring(4)
// -- MDS Attribute 9:  Locale { language; country; charset; str-spec }
0x0A, 0x28, 0x00, 0x0E,            // NOM_ATTR_LOCALE (2600)
0x65, 0x6E, 0x00, 0x00,            // language = "en"
0x55, 0x53, 0x00, 0x00,            // country = "US"
0x03, 0xE8,                        // charset = charset-iso-10646-ucs-2(1000)
0x00, 0x40, 0x80, 0x00             // str-spec: str-max-len=64; str-flag-nt(0)
```

Figure F.6—MDS object create notification event report formatting example

```
// PDU Header:
0xE1, 0x00,                       // MDAP_SPDU_SI
0x00, 0x02,                       // Presentation Context ID
// -- ROSE Section
0x00, 0x02, 0x00, 0x14,           // Result (RORSapdu), length
0x00, 0x01, 0x00, 0x01, 0x00, 0x0E,// Invoke ID=1,cmipConfirmedEventReport(1),length
// -- CMDIP Section  (EventReportResult)
0x00, 0x24, 0x00, 0x00, 0x00, 0x01,// NOM_MOC_VMS_MDS_HYD(36), Context=0, Handle=1
0x00, 0x00, 0x00, 0x00,           // Relative Time
0x0D, 0x06, 0x00, 0x00            // NOM_NOTI_MDS_CREAT(3334), result info len=ZERO
```

Figure F.7—MDS object create notification event report response formatting example

```
Structure                                     Nominal Value        Hex encoding
Session
 Id :                                         MDAP_SPDU_SI         E1 00
Presentation
 Pres. context id:                            0x0001               00 01
 Application
   ROSEapdu
       choice     :                           ROIVapdu             00 01
       length     :                           64                   00 40
       ROIVapdu
        invoke identifier   :                                      xx xx
        operation value     :                 cmipEventReport      00 00
        argument length    :                  58                   00 3A
       CMDIP EventReportArg.
        managed object class:                 NOM_MOC_SCAN_CFG_PERI 00 1B
        man. obj. context Id :                                     xx xx
        managed obj. handle :                                      xx xx
        event time   :                                             xx xx xx xx
        event type (Any defined by)
         nom oid   :                          NOM_NOTI_BUF_SCAN_RPT 09 03
         event info length   :                44                   00 2C
           ScanReportInfo
               scan-report-no                 1                    00 01
               glb-scan-info.count                                 00 01
               SingleCtxtScan
               context-id                                          00 00
                scan-info.cnt                                      00 02
                   ObservationScan
                       obj-handle                                  xx xx
                       attributes.cnt                              00 01
                           AVA-Type
                           attribute-id                            05 79
                           attribute-val len                       00 0A
                            .data
                               physio-id(...INF_RATE)              22 03
                               state (Test data)                   00 10
                               units                               xx xx
                               value (FLOAT-Type                   xx xx xx xx
                   ObservationScan
                       obj-handle                                  xx xx
                       attributes.cnt                              00 01
                        AVA-Type
                           attribute-id                            05 79
                           attribute-val.len                       00 0A
                            .data
                               physio-id (...INF_VOL)              22 04
                               state (Test data)                   00 10
                               units                               xx xx
                       value (FLOAT-Type)                          xx xx xx xx
```

Figure F.8—Configured periodic scanner event report formatting example


```
// PDU Header:
0xE1, 0x00,                        // MDAP_SPDU_SI
0x00, 0x02,                        // Presentation Context ID
// -- ROSE Section
0x00, 0x01, 0x00, 0xAC,            // Invoke (ROIVapdu), length
0x00, 0x06, 0x00, 0x00, 0x00, 0xA6,// Invoke ID=6, cmipEventReport(0),length
// -- CMDIP Section (EventReportArgument)
0x00, 0x13, 0x00, 0x00, 0x00, 0x0C,// NOM_MOC_SCAN_CFG_PERI(19)/Handle=12
0x00, 0x00, 0x00, 0x00,            // Relative Time Stamp for Report
0x0D, 0x03, 0x00, 0x98,            // NOM_NOTI_BUF_SCAN_RPT(3331), length

// Periodic Scanner::ScanReportInfo Section
0x00, 0x01,                        // scan-report-no = 1
0x00, 0x01, 0x00, 0x92,            // glb-scan-info = SEQ OF 1 SingleCtxtScan
0x00, 0x00,                        // context-id = 0x0000
0x00, 0x07, 0x00, 0x8C,            // scan-info = SEQ OF 7 Observation Scan

// Object Observation Scan 1: Primary VTBI (Metric Observed Value Group)
0x00, 0x70,0x00, 0x01, 0x00, 0x0E, // Handle + AttributeList (SEQ OF 1 AVA)
// -- Attribute 1: Nu-Observed-Value
0x09, 0x50, 0x00, 0x0A,            // NOM_ATTR_NU_VAL_OBS(2384), length
0x68, 0xB0, 0x08, 0x00,            // NOM_VOL_FLUID_TBI_REMAIN(26800),test-data(4)
0x06, 0x52,                        // NOM_DIM_MILLI_L(1618)
0x00, 0x00, 0x00, 0x00,            // value=FLOAT-Type

// Object Observation Scan 2: Primary Duration (Metric Observed Value Group)
0x00, 0x71,0x00, 0x01, 0x00, 0x0E, // Handle + AttributeList (SEQ OF 1 AVA)
// -- Attribute 1: Nu-Observed-Value
0x09, 0x50, 0x00, 0x0A,            // NOM_ATTR_NU_VAL_OBS(2384), length
0x68, 0xDC, 0x08, 0x00,            // NOM_TIME_PD_REMAIN(26844),test-data(4)
0x08, 0xA0,                        // NOM_DIM_MIN(2208)
0x00, 0x00, 0x00, 0x00,            // value=FLOAT-Type

// Object Observation Scan 3: Primary VI (Metric Observed Value Group)
0x00, 0x73,0x00, 0x01, 0x00, 0x0E, // Handle + AttributeList (SEQ OF 1 AVA)
// -- Attribute 1: Nu-Observed-Value
0x09, 0x50, 0x00, 0x0A,            // NOM_ATTR_NU_VAL_OBS(2384), length
0x68, 0xA8, 0x08, 0x00,            // NOM_VOL_FLUID_DELIV(26792),test-data(4)
0x06, 0x52,                        // NOM_DIM_MILLI_L(1618)
0x00, 0x00, 0x00, 0x00,            // value=FLOAT-Type

// Object Observation Scan 4: Secondary VTBI (Metric Observed Value Group)
0x00, 0x84,0x00, 0x01, 0x00, 0x0E, // Handle + AttributeList (SEQ OF 1 AVA)
// -- Attribute 1: Nu-Observed-Value
0x09, 0x50, 0x00, 0x0A,            // NOM_ATTR_NU_VAL_OBS(2384), length
0x68, 0xB0, 0x08, 0x00,            // NOM_VOL_FLUID_TBI_REMAIN(26800),test-data(4)
0x06, 0x52,                        // NOM_DIM_MILLI_L(1618)
0x00, 0x00, 0x00, 0x00,            // value=FLOAT-Type

// Object Observation Scan 5: Secondary Duration (Metric Observed Value Group)
0x00, 0x85,0x00, 0x01, 0x00, 0x0E, // Handle + AttributeList (SEQ OF 1 AVA)
// -- Attribute 1: Nu-Observed-Value
0x09, 0x50, 0x00, 0x0A,            // NOM_ATTR_NU_VAL_OBS(2384), length
0x68, 0xDC, 0x08, 0x00,            // NOM_TIME_PD_REMAIN(26844),test-data(4)
0x08, 0xA0,                        // NOM_DIM_MIN(2208)
0x00, 0x00, 0x00, 0x00,            // value=FLOAT-Type
```

Figure F.9—Periodic scanner buffered scan event report formatting example

```
// Object Observation Scan 6: Secondary VI (Metric Observed Value Group)
0x00, 0x87,0x00, 0x01, 0x00, 0x0E, // Handle + AttributeList (SEQ OF 1 AVA)
// -- Attribute 1: Nu-Observed-Value
0x09, 0x50, 0x00, 0x0A,            // NOM_ATTR_NU_VAL_OBS(2384), length
0x68, 0xA8, 0x08, 0x00,            // NOM_VOL_FLUID_DELIV(26792),test-data(4)
0x06, 0x52,                        // NOM_DIM_MILLI_L(1618)
0x00, 0x00, 0x00, 0x00,            // value=FLOAT-Type

// Object Observation Scan 7: Total VI (Metric Observed Value Group)
0x00, 0x98,0x00, 0x01, 0x00, 0x0E, // Handle + AttributeList (SEQ OF 1 AVA)
// -- Attribute 1: Nu-Observed-Value
0x09, 0x50, 0x00, 0x0A,            // NOM_ATTR_NU_VAL_OBS(2384), length
0x68, 0xFC, 0x08, 0x00,            // NOM_VOL_INFUS_ACTUAL_TOTAL(26876),test-data(4)
0x06, 0x52,                        // NOM_DIM_MILLI_L(1618)
0x00, 0x00, 0x00, 0x00             // value=FLOAT-Type
```

Figure F.9—Periodic scanner buffered scan event report formatting example *(continued)*

Annex G

(informative)

Specialization of ASN.1

G.1 Introduction

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. The notation is also a key component of the DIM and the ISO/IEEE 11073-10300 family of device specialization standards.

The MDER described in Annex A defines methods to transform ASN.1 syntax into a byte stream suitable for communication. It should be noted that MDER functions only on a subset of ASN.1.

This annex describes the specialization of ASN.1 for encoding with MDER. All ASN.1 PDU components destined for encoding with MDER are subject to this specialization.

G.2 ASN.1 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.

The specialization of ASN.1 data types is summarized in Table G.1. Refer to Annex A for MDER encoding relationships.

Table G.1—Specialization of ASN.1 data types

ASN.1 type	Status	Comments
BOOLEAN	E	—
INTEGER	E	Refer to Table G.2 for a list of alternate types supported.
ENUMERATED	E	Use a NamedNumberedList with the INTEGER types in Table G.2.
REAL	E	Use FLOAT.
BITSTRING	E	Refer to Table G.2 for a list of alternate types supported.
OCTETSTRING	I	—
NULL	R	Null primitive is generally excluded in MDER but is included with restrictions in CHOICE and ANY DEFINED BY primitives in MDER.
SEQUENCE	R	May not use OPTIONAL, DEFAULT or COMPONENTS OF keywords nor automatic tagging.
SEQUENCE OF	I	—
SET	E	Use the SEQUENCE type.
SET OF	E	Use the SEQUENCE OF type.
CHOICE	R	Alternatives must be tagged. Automatic tagging is not supported.

Table G.1—Specialization of ASN.1 data types *(continued)*

ASN.1 type	Status	Comments
SELECTION	E	—
TAGGED	R	Only for use with an alternative in a CHOICE type.
OBJECT IDENTIFIER	E	—
EMBEDDED PDV	E	—
EXTERNAL	E	—
CHARACTER STRING	E	—
ANY DEFINED BY	R	An ANY DEFINED BY shall identify a component of the containing SEQUENCE. That component shall be an OBJECT IDENTIFIER (nomenclature). The OBJECT IDENTIFIER may be context-free or context-sensitive.

Table G.2—Supported integer, bitstring, and float types

Types	Notation	Definition	Description
INTEGER	INT-U8	INTEGER (0..255)	Unsigned 8 bit integer
	INT-I8	INTEGER (–128..127)	Signed 8 bit integer
	INT-U16	INTEGER (0..65535)	Unsigned 16 bit integer
	INT-I16	INTEGER (–32768..32767)	Signed 16 bit integer
	INT-U32	INTEGER (0..4294967295)	Unsigned 32 bit integer
	INT-I32	INTEGER (–2147483648 ..2147483647)	Signed 32 bit integer
BITSTRING	BITS-16	BIT STRING (SIZE(16))	16 bit bitstring
	BITS-32	BIT STRING (SIZE(32))	32 bit bitstring
FLOAT	FLOAT-Type	REAL (WITH COMPONENTS {mantissa (–8388605..8388605), base (10), exponent (–128..127)})	32 bit floating point

Elements of Table G.1 are described briefly as follows:

- BOOLEAN: The BOOLEAN type is not a part of the ASN.1 specialization.
- INTEGER: The ASN.1 INTEGER type is not a part of the ASN.1 specialization. Instead, MDER offers encodings for the INTEGER types in Table G.2. These types follow the syntax shown in the table.
- ENUMERATED: The ASN.1 ENUMERATED type is not a part of the ASN.1 specialization. Instead, MDER offers encodings for INTEGER types in Table G.2. These INTEGER types may be used with a NamedNumberList, which is analogous to an ENUMERATED type.
- REAL: The ASN.1 REAL type is not a part of the ASN.1 specialization. Instead, MDER offers an encoding of the FLOAT-Type, which is a 32 bit floating point type. This type appears in Table G.2.
- BITSTRING: The BIT STRING type is not a part of the ASN.1 specialization. Instead, MDER offers encodings for the BIT STRING types listed in Table G.2.

— OCTETSTRING: The OCTET STRING type is a part of the ASN.1 specialization. There are no restrictions on its use.

— NULL: Null primitive is generally excluded in MDER, but is included with restrictions in CHOICE and ANY DEFINED BY primitives in MDER.

— SEQUENCE: The SEQUENCE type is a part of the ASN.1 specialization, with certain restrictions. A component of the SEQUENCE may not be specified with the OPTIONAL, DEFAULT, or COMPONENTS OF keywords. Automatic tagging is not supported.

— SEQUENCE OF: The SEQUENCE OF type is a part of the ASN.1 specialization, There are no restrictions on its use.

— SET: The SET type is not a part of the ASN.1 specialization. The SEQUENCE type is the recommended alternative.

— SET OF: The SET OF type is not a part of the ASN.1 specialization. The SEQUENCE OF type is the recommended alternative.

— CHOICE: The CHOICE type is a part of the ASN.1 specialization, with certain restrictions. Each alternative in the CHOICE must be a TAGGED type. Automatic tags are not supported.

— SELECTION: The SELECTION type operator < is not a part of the ASN.1 specialization.

— TAGGED: In general, TAGGED types are not a part of the ASN.1 specialization. However, each alternative type of a CHOICE must be a TAGGED type.

— OBJECT IDENTIFIER: The OBJECT IDENTIFIER type is not a part of the ASN.1 specialization.

— EMBEDDED PDV: The EMBEDDED PDV type is not a part of the ASN.1 specialization.

— EXTERNAL: The EXTERNAL type is not a part of the ASN.1 specialization.

— CHARACTER STRING: CHARACTER STRING types are not a part of the ASN.1 specialization. Instead, the use of the OCTETSTRING type is recommended.

— ANY DEFINED BY: The ANY DEFINED BY type is part of the ASN.1 specialization. The ANY DEFINED BY shall identify a component of the containing SEQUENCE. That component shall be an OBJECT IDENTIFIER (nomenclature). The OBJECT IDENTIFIER may be context-free or context-sensitive. More information about the use of ANY DEFINED BY may be found in Annex H.

Annex H

(informative)

Compatibility cases

This annex provides information to rationalize compatibility issues and decisions.

H.1 ANY DEFINED BY

Changes in ISO ASN.1 and related encoding rules (e.g., BER) between 1988/90 and 1994 versions resulted in a particular change to the syntax for the (1988/90) ANY DEFINED BY definition. The following paragraphs are extracts from relevant ISO documents concerning the changes; refer to the main body of this standard for normative specifications concerning impact on MDAP.

H.1.1 Migrating to current ASN.1 notation

The following extract is from Annex A of ISO/IEC 8824-1:1994:

> A.3 Migration to the current ASN.1 notation
>
> When modifying a module (originally written to conform to the ASN.1-88/90 notation) to conform to the current notation, the following points should be noted:
>
> …
>
> b) All uses of ANY and ANY DEFINED BY shall be supported by a suitable information object class definition, with the ANY and ANY DEFINED BY (and the referenced component) replaced by appropriate references to fields of that object class. In most cases the specification can be greatly improved by careful attention to the insertion of table and component relation constraints. In many cases the specification can be further improved if the table or component relation constraint is made a parameter of the type.

H.1.2 TYPE-IDENTIFER information object class in ASN.1

The following extract is from Annex A of ISO/IEC 8824-2:1994:

> A.1 This annex specifies a useful information object class, with class reference TYPE-IDENTIFIER.
>
> NOTE—This information object class is the simplest useful class, having just two fields, an identifier field of type OBJECT IDENTIFIER, and a type field which defines the ASN.1 type for carrying all information concerning any particular object in the class. It is defined in this Recommendation | International Standard because of the widespread use of information objects of this form.

A.2 The TYPE-IDENTIFIER information object class is defined as:

```
TYPE-IDENTIFIER ::= CLASS
{
    &id OBJECT IDENTIFIER UNIQUE,
    &Type
}
WITH SYNTAX {&Type IDENTIFIED BY &id}
```

A.3 This class is defined as a “useful” information object class, and is available in any module without the necessity for importing it.

A.4 Example

The body of an MHS communication can be defined as:

```
MHS-BODY-CLASS ::= TYPE-IDENTIFIER

g4FaxBody MHS-BODY-CLASS ::=
    {BIT STRING IDENTIFIED BY {mhsbody 3}}
```

A protocol designer would typically define a component to carry an MHS-BODY-CLASS by specifying the type "INSTANCE OF MHS-BODY-CLASS" defined in C.9.

H.1.3 instance-of type encoding in BER

The following extract is from ISO/IEC 8825-1:1994:

8.16 Encoding of an instance-of value

8.16.1 The encoding of the instance-of type shall be the BER encoding of the following sequence type with the value as specified in 8.16.2:

```
[UNIVERSAL 8] IMPLICIT SEQUENCE
{
    type-id    <DefinedObjectClass>.&id,
    value      [0] EXPLICIT <DefinedObjectClass>.&Type
}
```

where "<DefinedObjectClass>" is replaced by the particular "DefinedObjectClass" used in the "InstanceOfType" notation.

NOTE—When the value is a value of a single ASN.1 type and BER encoding is used for it, the encoding of this type is identical to an encoding of a corresponding value of the external type, where the "syntax" alternative is in use for representing the abstract value.

8.16.2 The value of the components of the sequence type in 8.16.1 shall be the same as the values of the corresponding components of the associated type in ITU-T Rec. X.681 | ISO/IEC 8824-2, C.7.

Annex I

(informative)

Bibliography

[B1] Black, Uyless, *Network Management Standards—SNMP, CMIP, TMN, MIBs, and Object Libraries*, New York: McGraw-Hill, 1994.

[B2] Dubuisson, Oliver, *ASN.1—Communication Between Heterogeneous Systems*, Morgan Kaufmann, 2000.

[B3] IEEE 100, *The Authoritative Dictionary of IEEE Standards Terms and Definitions*, Seventh Edition.[16]

[B4] IEEE Std 1003.1g™, Information Technology—Portable Operating System Interface (POSIX®)—Protocol Independent Interfaces (PII).

[B5] IEEE Std 1073.3.1, IEEE Standard for Medical Device Communications—Transport Profile—Connection Mode.

[B6] IETF RFC 2030, Simple Network Time Protocol (SNTP) Version 4 for IPv4, IPv6 and OSI.[17]

[B7] ISO/IEEE P11073-20102, Health informatics — Point-of-care medical device communication — Part 20101: Application profiles — MIB elements.[18]

[B8] ISO/IEEE P11073-20201, Health informatics — Point-of-care medical device communication — Part 20201: Application profile — Polling mode.

[B9] ISO/IEEE P11073-20202, Health informatics — Point-of-care medical device communication — Part 20202: Application profile — Baseline.

[B10] ITU-T Recommendation X.225, Information Technology—Open Systems Interconnection—Connection-oriented session protocol: Protocol Specification.[19] (same as ISO/IEC 8327-1)

[B11] ITU-T Recommendation X.226, Information Technology—Open Systems Interconnection—Connection-oriented presentation protocol: Protocol Specification.

[B12] ITU-T Recommendation X.227, Information Technology—Open Systems Interconnection—Connection-oriented protocol for the association control service element. (same as ISO/IEC 8650-1)

[B13] ITU-T Recommendation X.680, Information Technology—Abstract Syntax Notation One (ASN.1)—Specification of Basic Notation. (same as ISO/IEC 8824-1)

[16]IEEE publications are available from the Institute of Electrical and Electronics Engineers, Inc., 445 Hoes Lane, Piscataway, NJ 08854, USA (http://standards.ieee.org/).

[17]IETF publications are available from the Internet Engineering Task Force (http://www.ietf.org/).

[18]ISO/IEEE publications are available from the ISO Central Secretariat, Case Postale 56, 1 rue de Varembé, CH-1211, Genève 20, Switzerland/Suisse (http://www.iso.ch/); in the United States from the Sales Department, American National Standards Institute, 25 West 43rd Street, 4th Floor, New York, NY 10036, USA (http://www.ansi.org/); and from the Institute of Electrical and Electronics Engineers, Inc., 445 Hoes Lane, Piscataway, NJ 08854, USA (http://standards.ieee.org/).

[19]ITU-T publications are available from the International Telecommunications Union, Place des Nations, CH-1211, Geneva 20, Switzerland/Suisse (http://www.itu.int/).

[B14] ITU-T Recommendation X.690, Information Technology—ASN.1 Encoding Rules: Specification of Basic Encoding Rules (BER), Canonical Encoding Rules (CER) and Distinguished Encoding Rules (DER). (same as ISO/IEC 8825-1)

[B15] Piscitello, David M., and Chapin, A. Lyman, *Open Systems Networking: TCP/IP and OSI*, Pearson Addison Wesley, 1993.

[B16] Stallings, William, *SNMP, SNMPv2, and CMIP—The Practical Guide to Network-Management Standards*, Pearson Addison Wesley, 1993.

[B17] Stevens, W. Richard, *UNIX Network Programming*, Prentice Hall, 1990.

ISO/IEEE 11073-20101:2004(E)

ISBN 0-7381-4091-0

ICS 35.240.80

Price based on 78 pages