INTERNATIONAL STANDARD

ISO 16484-5

Second edition
2007-03-15

Building automation and control systems —

Part 5: Data communication protocol

Systèmes d'automatisation et de gestion technique du bâtiment —

Partie 5: Protocole de communication de données

Reference number
ISO 16484-5:2007(E)

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

Published in Switzerland

Foreword

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

International Standards are drafted in accordance with the rules given in the ISO/IEC Directives, Part 2.

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

Attention is drawn to the possibility that some of the elements of this document may be the subject of patent rights. ISO shall not be held responsible for identifying any or all such patent rights.

ISO 16484-5 was prepared by the American Society of Heating, Refrigerating and Air-Conditioning Engineers (ASHRAE) (as ANSI/ASHRAE 135-2004) and was adopted without modifications by Technical Committee ISO/TC 205, *Building environment design*.

This second edition cancels and replaces the first edition (ISO 16484-5:2003), which has been technically revised, as detailed in the enclosed ANSI/ASHRAE publication, pages 598 to 601.

ISO 16484 consists of the following parts, under the general title *Building automation and control systems*:

— *Part 1: Overview and definitions*

— *Part 2: Hardware*

— *Part 3: Functions*

— *Part 5: Data communication protocol*

— *Part 6: Data communication conformance testing*

Applications and project implementation are to form the subjects of future parts 4 and 7.

Building automation and control systems —

Part 5:
Data communication protocol

1 Scope

This part of ISO 16484 defines data communication services and protocols for computer equipment used for monitoring and control of heating, ventilation, air-conditioning and refrigeration (HVAC&R) and other building systems. It defines, in addition, an abstract, object-oriented representation of information communicated between such equipment, thereby facilitating the application and use of digital control technology in buildings. The scope and field of application are furthermore detailed in Clause 2 of the enclosed ANSI/ASHRAE publication.

2 Requirements

Requirements are the technical recommendations made in the following publication (reproduced on the following pages), which is adopted as an International Standard:

> ANSI/ASHRAE 135-2004, *A Data Communication Protocol for Building Automation and Control Networks*

The text on the back of the title page of the ANSI/ASHRAE standard and the policy statement on the last page are not relevant for the purposes of international standardization.

The following International Standards are cited in the text:

ISO/IEC 7498 (all parts), *Information technology — Open Systems Interconnection — Basic Reference Model*

ISO/TR 8509, *Information processing systems — Open Systems Interconnection — Service conventions*

ISO/IEC 8649, *Information technology — Open Systems Interconnection — Service definition for the Association Control Service Element*

ISO/IEC 8802-2, *Information technology — Telecommunications and information exchange between systems — Local and metropolitan area networks — Specific requirements — Part 2: Logical link control*

ISO/IEC 8802-3, *Information technology — Telecommunications and information exchange between systems — Local and metropolitan area networks — Specific requirements — Part 3: Carrier sense multiple access with collision detection (CSMA/CD) access method and physical layer specifications*

ISO/IEC 8824 (all parts), *Information technology — Abstract Syntax Notation One (ASN.1)*

ISO/IEC 8825 (all parts), *Information technology — ASN.1 encoding rules*

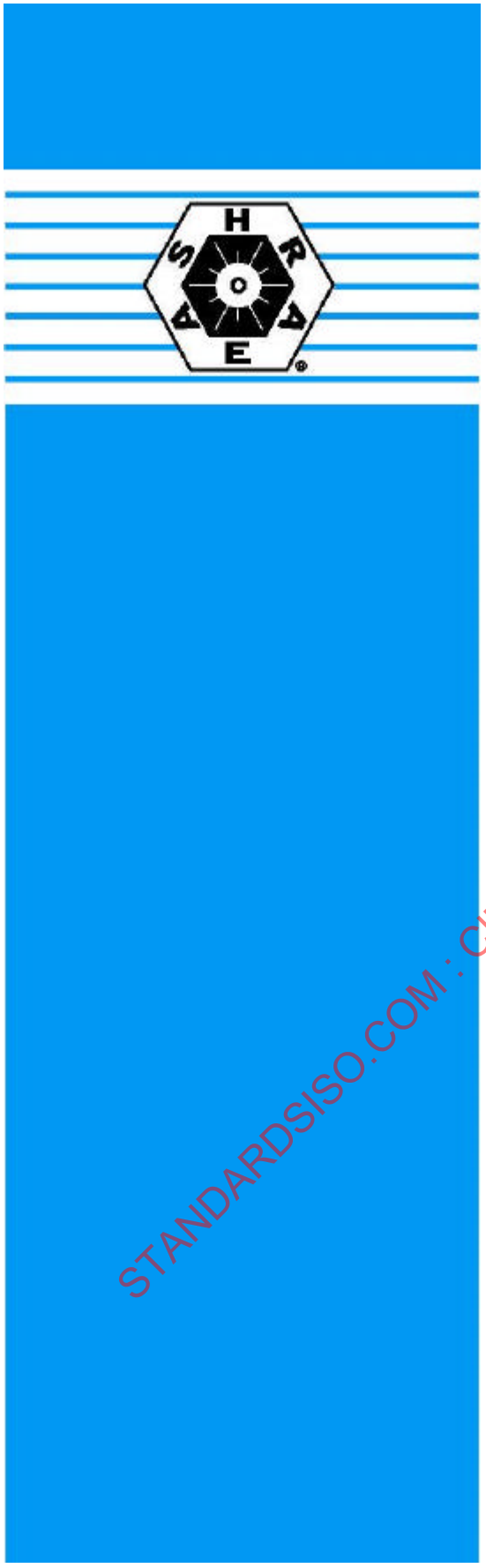
ISO/IEC 8859-1, *Information technology — 8-bit single-byte coded graphic character sets — Part 1: Latin alphabet No. 1*

ISO/IEC 9545, *Information technology — Open Systems Interconnection — Application Layer structure*

ISO/IEC 10646, *Information technology — Universal Multiple-Octet Coded Character Set (UCS)*

3 Revision of ANSI/ASHRAE 135

It has been agreed with the American Society of Heating, Refrigerating and Air-Conditioning Engineers (ASHRAE) that Technical Committee ISO/TC 205 will be consulted in the event of any revision or amendment of ANSI/ASHRAE 135. To this end, ANSI will act as a liaison body between ASHRAE and ISO.



ANSI/ASHRAE Standard 135-2004
(Including ANSI/ASHRAE addenda listed in the History of Revisions)

ASHRAE® STANDARD

BACnet® — A Data Communication Protocol for Building Automation and Control Networks

Approved by the ASHRAE Standards Committee October 5, 2003; by the ASHRAE Board of Directors Jaunaru 29, 2004; and by the American National Standards Institute February 25, 2004. See "History of Revisions" section for approval dates of addenda.

This standard is under continuous maintenance by a Standing Standard Project Committee (SSPC) for which the Standards Committee has established a documented program for regular publication of addenda or revisions, including procdedures for timely, documented, consensus action on requests for change to any part of the standard. The change submittal form, instructions and deadlines are given at the back of this document and may be obtained in electronic form from ASHRAE's Internet Home Page, *http://www.ashrae.org*, or in paper form from the Manager of Standards. The latest edition of an ASHRAE Standard and printed copies of a public review draft may be purchased from ASHRAE Customer Service, 1791 Tullie Circle, NE, Atlanta, GA 30329-2305. E-mail: *orders@ashrae.org*. Fax: 404-321-5478. Telephone: 404-636-8400 (worldwide), or toll free 1-800-527-4723 (for orders in U.S. and Canada).

ISSN 1041-2336

AMERICAN SOCIETY OF HEATING, REFRIGERATING AND AIR-CONDITIONING ENGINEERS, INC.
1791 Tullie Circle, NE· Atlanta GA 30329-2305

(Blank page)

ASHRAE STANDING STANDARD PROJECT COMMITTEE 135
Cognizant TC: TC 1.4, Control Theory and Applications
SPLS Liaison: Frank E. Jakob

Steven T. Bushby, *Chair**
William O. Swan III, *Vice-Chair*
Carl Neilson, *Secretary**
Barry B. Bridges*
James F. Butler*
A. J. Capowski*
Keith A. Corbett
Jeffery Cosiol
Troy D. Cowan*
Daniel P. Giorgis
Thomas S. Ertsgaard*
Craig P. Gemmill*
Robert L. Johnson
Stephen T. Karg*
J. Damian Ljungquist*
Jerald P. Martocci
Mark A. Railsback
David W. Robin
Ernest L. Senior
Daniel A. Traill*
J. Michael Whitcomb*
David F. White
Grant N. Wichenko

**Denotes members of voting status when this standard was approved for publication.*

The following persons served as consultants to the project committee:

Andrey Golovin
David G. Holmberg
H. Michael Newman
René Quirighetti
David H. Ritter
Takeji Toyoda

ASHRAE STANDARDS COMMITTEE 2003-2004

Van D. Baxter, *Chair*
Davor Novosel, *Vice-Chair*
Donald B. Bivens
Dean S. Borges
Paul W. Cabot
Charles W. Coward, Jr.
Hugh F. Crowther
Brian P. Dougherty
Hakim Elmahdy
Matt R. Hargan
Richard D. Hermans
John F. Hogan
Frank E. Jakob
Stephen D. Kennedy
David E. Knebel
Frederick H. Kohloss
Merle F. McBride
Mark P. Modera
Cyrus H. Nasseri
Gideon Shavit
David R. Tree
Thomas H. Williams
James E. Woods
Kent W. Peterson, CO
Ross D. Montgomery, BOD ExO

Claire B. Ramspeck, Manager of Standards

SPECIAL NOTE

This American National Standard (ANS) is a national voluntary consensus standard developed under the auspices of the American Society of Heating, Refrigerating and Air-Conditioning Engineers (ASHRAE). Consensus is defined by the American National Standards Institute (ANSI), of which ASHRAE is a member and which has approved this standard as an ANS, as "substantial agreement reached by directly and materially affected interest categories. This signifies the concurrence of more than a simple majority, but not necessarily unanimity. Consensus requires that all views and objections be considered, and that an effort be made toward their resolution." Compliance with this standard is voluntary until and unless a legal jurisdiction makes compliance mandatory through legislation.

ASHRAE obtains consensus through participation of its national and international members, associated societies, and public review.

ASHRAE Standards are prepared by a Project Committee appointed specifically for the purpose of writing the Standard. The Project Committee Chair and Vice-Chair must be members of ASHRAE; while other members may or may not be members of ASHRAE, all must be technically qualified in the subject area of the standard. Every effort is made to balance the concerned interests on all Project Committees.

The Manager of Standards of ASHRAE should be contacted for:

a. interpretation of the contents of this Standard,
b. participation in the next review of the Standard,
c. offering constructive criticism for improving the Standard,
d. permission to reprint portions of the Standard.

DISCLAIMER

ASHRAE uses its best efforts to promulgate standards for the benefit of the public in light of available information and accepted industry practices. However, ASHRAE does not guarantee, certify, or assure the safety or performance of any products, components, or systems tested, designed, installed, or operated in accordance with ASHRAE's Standards or Guidelines or that any tests conducted under its standards will be nonhazardous or free from risk.

ASHRAE INDUSTRIAL ADVERTISING POLICY ON STANDARDS

ASHRAE Standards and Guidelines are established to assist industry and the public by offering a uniform method of testing for rating purposes, by suggesting safe practices in designing and installing equipment, by providing proper definitions of this equipment, and by providing other information that may serve to guide the industry. The creation of ASHRAE Standards and Guidelines is determined by the need for them, and conformance to them is completely voluntary.

In referring to this standard and marking of equipment and in advertising, no claim shall be made, either stated or implied, that the product has been approved by ASHRAE.

(Blank page)

CONTENTS

FOREWORD

BACnet, the ASHRAE building automation and control networking protocol, has been designed specifically to meet the communication needs of building automation and control systems for applications such as heating, ventilating, and air-conditioning control, lighting control, access control, and fire detection systems. The BACnet protocol provides mechanisms by which computerized equipment of arbitrary function may exchange information, regardless of the particular building service it performs. As a result, the BACnet protocol may be used by head-end computers, general-purpose direct digital controllers, and application specific or unitary controllers with equal effect.

The motivation for this Standard was the widespread desire of building owners and operators for "interoperability," the ability to integrate equipment from different vendors into a coherent automation and control system - and to do so competitively. To accomplish this, the Standard Project Committee (SPC) solicited and received input from dozens of interested firms and individuals; reviewed all relevant national and international data communications standards, whether de facto or the result of committee activity; and spent countless hours in debate and discussion of the pros and cons of each element of the protocol.

What has emerged from the committee deliberations is a network protocol model with these principal characteristics:

(a) All network devices (except MS/TP slaves) are peers, but certain peers may have greater privileges and responsibilities than others.

(b) Each network device is modeled as a collection of network-accessible, named entities called "objects." Each object is characterized by a set of attributes or "properties." While this Standard prescribes the most widely applicable object types and their properties, implementors are free to create additional object types if desired. Because the object model can be easily extended, it provides a way for BACnet to evolve in a backward compatible manner as the technology and building needs change.

(c) Communication is accomplished by reading and writing the properties of particular objects and by the mutually acceptable execution of other protocol "services." While this Standard prescribes a comprehensive set of services, mechanisms are also provided for implementors to create additional services if desired.

(d) Because of this Standard's adherence to the ISO concept of a "layered" communication architecture, the same messages may be exchanged using various network access methods and physical media. This means that BACnet networks may be configured to meet a range of speed and throughput requirements with commensurately varying cost. Multiple BACnet networks can be interconnected within the same system forming an internetwork of arbitrarily large size. This flexibility also provides a way for BACnet to embrace new networking technologies as they are developed.

BACnet was designed to gracefully improve and evolve as both computer technology and demands of building automation systems change. Upon its original publication in 1995, a Standing Standards Project Committee was formed to deliberate enhancements to the protocol under ASHRAE rules for "continuous maintenance." Much has happened since the BACnet standard was first promulgated. BACnet has been translated into Chinese, Japanese, and Korean, and embraced across the globe. BACnet devices have been designed, built and deployed on all seven continents. Suggestions for enhancements and improvements have been continually received, deliberated, and, ultimately, subjected to the same consensus process that produced the original standard. This publication is the result of those deliberations and brings together all of the corrections, refinements, and improvements that have been adopted.

Among the features that have been added to BACnet are: increased capabilities to interconnect systems across wide area networks using internet protocols, new objects and services to support fire detection and other life safety applications, capabilities to backup and restore devices, standard ways to collect trend data, new tools to make specifying BACnet systems easier, a mechanism for making interoperable extensions to the standard visible, and many others. The successful addition of these features demonstrates that the concept of a protocol deliberately crafted to permit extension of its capabilities over time as technology and needs change is viable and sound.

All communication protocols are, in the end, a collection of arbitrary solutions to the problems of information exchange and all are subject to change as time and technology advance. BACnet is no exception. Still, it is the hope of those who have contributed their time, energies, and talents to this work that BACnet will help to fulfill, in the area of building automation and control, the promise of the information age for the public good!

(Blank page)

1 PURPOSE

The purpose of this standard is to define data communication services and protocols for computer equipment used for monitoring and control of HVAC&R and other building systems and to define, in addition, an abstract, object-oriented representation of information communicated between such equipment, thereby facilitating the application and use of digital control technology in buildings.

2 SCOPE

2.1 This protocol provides a comprehensive set of messages for conveying encoded binary, analog, and alphanumeric data between devices including, but not limited to:

(a) hardware binary input and output values,
(b) hardware analog input and output values,
(c) software binary and analog values,
(d) text string values,
(e) schedule information,
(f) alarm and event information,
(g) files, and
(h) control logic.

2.2 This protocol models each building automation and control computer as a collection of data structures called "objects," the properties of which represent various aspects of the hardware, software, and operation of the device. These objects provide a means of identifying and accessing information without requiring knowledge of the details of the device's internal design or configuration.

3 DEFINITIONS

3.1 Terms Adopted from International Standards

The following terms used in this standard are defined by international standards or draft standards for open system interconnection (OSI). The definitions are repeated here and a reference to the appropriate standard is provided. Clause 25 contains the titles of all national and international standards referenced in this clause and elsewhere in this standard. Words or phrases in italics refer to terms defined elsewhere in this clause.

3.1.1 abstract syntax: the specification of application layer data or *application-protocol-control-information* by using notation rules which are independent of the encoding technique used to represent them (ISO 8822).

3.1.2 application: a set of a USER's information processing requirements (ISO 8649).

3.1.3 application-entity: the aspects of an *application-process* pertinent to OSI (ISO 7498).

3.1.4 application-process: an element within a *real open system* which performs the information processing for a particular *application* (ISO 7498).

3.1.5 application-protocol-control-information: information exchanged between *application-entities*, using presentation services, to coordinate their joint operation (ISO 9545).

3.1.6 application-protocol-data-unit: a unit of data specified in an application protocol and consisting of *application-protocol-control-information* and possibly application-user-data (ISO 9545).

3.1.7 application-service-element: that part of an *application-entity* which provides an OSI environment capability, using underlying services when appropriate (ISO 7498).

3.1.8 concrete syntax: those aspects of the rules used in the formal specification of data which embody a specific representation of that data (ISO 7498).

3.1.9 peer-entities: *entities* within the same layer (ISO 7498).

3.1.10 real open system: a *real system* which complies with the requirements of OSI standards in its communication with other *real systems* (ISO 7498).

3.1.11 real system: a set of one or more computers, the associated software, peripherals, terminals, human operators, physical processes, information transfer means, etc., that forms an autonomous whole capable of performing information processing and/or information transfer (ISO 7498).

3.1.12 (N)-service-access-point: the point at which (N)-services are provided by an (N)-*entity* to an (N+1)-*entity* (ISO 7498).

3.1.13 (N)-service-data-unit: an amount of (N)-interface-data whose identity is preserved from one end of an (N)-connection to the other (ISO 7498).

3.1.14 service-user: an *entity* in a single open system that makes use of a service through *service-access-points* (ISO TR 8509).

3.1.15 service-provider: an abstract of the totality of those entities which provide a service to peer *service-users* (ISO TR 8509).

3.1.16 transfer-syntax: that *concrete syntax* used in the transfer of data between open systems (ISO 7498).

3.1.17 service-primitive; **primitive**: an abstract, implementation-independent representation of an interaction between the *service-user* and the *service-provider* (ISO TR 8509).

3.1.18 request (primitive): a representation of an interaction in which a *service-user* invokes some procedure (ISO TR 8509).

3.1.19 indication (primitive): a representation of an interaction in which a *service-provider* either

(a) indicates that it has, on its own initiative, invoked some procedure; or

(b) indicates that a procedure has been invoked by the *service-user* at the peer *service-access-point* (ISO TR 8509).

3.1.20 response (primitive): a representation of an interaction in which a *service-user* indicates that it has completed some procedure previously invoked by an interaction represented by an *indication* primitive (ISO TR 8509).

3.1.21 confirm (primitive): a representation of an interaction in which a *service-provider* indicates, at a particular *service-access-point*, completion of some procedure previously invoked, at that *service-access-point*, by an interaction represented by a *request* primitive (ISO TR 8509).

3.1.22 user element: the representation of that part of an *application-process* which uses those *application-service-elements* needed to accomplish the communications objectives of that *application-process* (ISO 7498).

3.2 Terms Defined for this Standard

3.2.1 access control: a method for regulating or restricting access to *network resources.*

3.2.2 alarm: 1. An annunciation, either audible or visual or both, that alerts an operator to an off-normal condition that may require corrective action. **2.** An abnormal condition detected by a device or controller that implements a rule or logic specifically designed to look for that condition.

3.2.3 algorithmic change reporting: the detection and reporting of an alarm or event, based on an algorithm specified in an Event Enrollment object. See *intrinsic reporting* in 3.2.27.

3.2.4 BACnet device: any device, real or virtual, that supports digital communication using the BACnet protocol.

3.2.5 BACnet-user: that portion of an *application-process* that is represented by the BACnet *user element*.

3.2.6 bridge: a device that connects two or more *segments* at the physical and data link layers. This device may also perform message filtering based upon MAC layer addresses.

3.2.7 broadcast: a message sent as a single unit, which may apply to more than one device.

3.2.8 change of state: an event that occurs when a measured or calculated Boolean or discrete enumerated value changes.

3.2.9 change of value: an event that occurs when a measured or calculated analog value changes by a predefined amount.

3.2.10 client: a system or device that makes use of another device for some particular purpose via a service *request* instance. A client requests service from a *server*.

3.2.11 context: a set of data or information that completely describes a particular communication environment at a particular point in time.

3.2.12 controller: a device for regulation or management of a system or component.

3.2.13 data confidentiality: the property that information is not made available or disclosed to unauthorized individuals, entities, or processes.

3.2.14 data integrity: the property that data has not been altered or destroyed in an unauthorized manner.

3.2.15 data origin authentication: the corroboration that the source of data received is as claimed.

3.2.16 directly connected network: a network that is accessible from a router without messages being relayed through an intervening router. A PTP connection is to a directly connected network if the PTP connection is currently active and no intervening router is used.

3.2.17 download: a particular type of file transfer that refers to the transfer of an executable program or database to a remote device where it may be executed.

3.2.18 entity: something that has a separate and distinct existence. An identifiable item that is described by a set or collection of properties.

3.2.19 error detection: a procedure used to identify the presence of errors in a communication.

3.2.20 error recovery: a procedure invoked in response to a detected error that permits the information exchange to continue.

3.2.21 gateway: a device that connects two or more dissimilar *networks*, permitting information exchange between them.

3.2.22 global: pertaining to all devices or nodes on a communication *internetwork*.

3.2.23 global broadcast: a message addressed to all devices or *nodes* on all *networks* in a BACnet *internet*.

3.2.24 half router: a device or *node* that can participate as one partner in a PTP connection. The two half-router partners that form an active PTP connection together make up a single *router*.

3.2.25 initialization: the process of establishing a known state, usually from a power up condition. Initialization may require re-establishment of a node's logical or physical address.

3.2.26 internetwork: a set of two or more *networks* interconnected by *routers*. In a BACnet *internetwork*, there exists exactly one message path between any two nodes.

3.2.27 intrinsic reporting: the detection and reporting of an alarm or event, based on an algorithm defined as part of the *object type* specification. No external reference to an Event Enrollment is involved. See *algorithmic change reporting* in 3.2.3.

3.2.28 key: a sequence of symbols that controls the operations of encipherment and decipherment.

3.2.29 local: pertaining to devices on the same *network* as the referenced device.

3.2.30 local broadcast: a message addressed to all devices or *nodes* on the same *network* as the originator.

3.2.31 medium: the physical transmission *entity*. Typical media are twisted-pair wire, fiber optic cable, and coaxial cable.

3.2.32 medium access control: a process used to maintain order and provide access to the communication *medium.*

3.2.33 network: a set of one or more *segments* interconnected by *bridges* that have the same network address.

3.2.34 network resource: any physical or logical *entity* that may be accessed via a communication *medium*.

3.2.35 node: an addressable device connected to the communication *medium.*

3.2.36 object profile: an object profile is a means of defining objects beyond those defined in Clause 12. A profile defines the set of properties, behavior, and/or requirements for a proprietary object, or for proprietary extensions to a standard object.

3.2.37 object type: a generic classification of data that is defined by a set of *properties.*

3.2.38 operator authentication: the corroboration that the operator logging on to a device is as claimed.

3.2.39 peer entity authentication: the corroboration that a peer entity in an association is the one claimed.

3.2.40 physical segment: a single contiguous *medium* to which BACnet nodes are attached.

3.2.41 printable character: a character that represents a printable symbol as opposed to a device control character. These include, but are not limited to, upper- and lowercase letters, punctuation marks, and mathematical symbols. The exact set depends upon the character set being used. In ANSI X3.4 the printable characters are represented by single octets in the range X'20' - X'7E'.

3.2.42 property: a particular characteristic of an *object type.*

3.2.43 proprietary: within the context of BACnet, any extension of or addition to *object types, properties,* PrivateTransfer services, or enumerations specified in this standard.

3.2.44 receiving BACnet-user: the *BACnet-user* that receives an *indication* or *confirm* service primitive.

3.2.45 remote: pertaining to devices or *nodes* on a different *network* than the referenced device.

3.2.46 remote broadcast: a message addressed to all devices or *nodes* on a different *network* than the originator.

3.2.47 repeater: a device that connects two or more *physical segments* at the physical layer.

3.2.48 requesting BACnet-user: the *BACnet-user* that assumes the role of a *client* in a confirmed service.

3.2.49 responding BACnet-user: the *BACnet-user* that assumes the role of a *server* in a confirmed service.

3.2.50 router: a device that connects two or more *networks* at the network layer.

3.2.51 security: any of a variety of procedures used to ensure that information exchange is guarded to prevent disclosure to unauthorized individuals. Security measures are intended to prevent disclosure of sensitive information even to those who have valid access to the communication *network.* Security is distinct from access control, although some security can be provided by limiting physical access to the communication medium itself.

3.2.52 segment: a segment consists of one or more *physical segments* interconnected by repeaters.

3.2.53 sending BACnet-user: the *BACnet-user* that issues a *request* or *response* service primitive.

3.2.54 server: a system or device that responds to a service *request* instance for some particular purpose. The server provides service to a *client*.

3.2.55 synchronization: a facility that allows processes to define and identify specific places in a transmission or exchange that can be used to reset a communication session to a predefined state.

3.2.56 timestamp: the date and time recorded for and accompanying the record of an event or operation.

3.2.57 unit_time: the length of time required to transmit one octet with a start bit and a single stop bit. Ten bit-times.

3.2.58 upload: the process of transferring an executable program image or a database from a remote device in such a manner as to allow subsequent download.

3.3 Abbreviations and Acronyms Used in this Standard

A	application layer (prefix)
AE	application entity
ANSI	American National Standards Institute
APCI	application protocol control information
APDU	application layer protocol data unit
API	application program interface
ARCNET	attached resource computer network
ASE	application service element
ASN.1	Abstract Syntax Notation One (ISO 8824)
B' '	denotes that binary notation is used between the single quotes
BAC	building automation and control
BBMD	BACnet/IP broadcast management device
BDT	broadcast distribution table
B/IP	BACnet/IP
B/IP-M	BACnet/IP multicast
BVLC	BACnet virtual link control
BVLCI	BACnet virtual link control information
BVLL	BACnet virtual link layer
C	conditional
C(=)	conditional (The parameter is semantically equivalent to the parameter in the service primitive to its immediate left in the table.)
CNF	confirm primitive
COV	change of value
CRC	cyclic redundancy check
D' '	denotes that decimal notation is used between the single quotes
DA	local destination MAC layer address
DADR	ultimate destination MAC layer address
DER	data expecting reply
DES	Data Encryption Standard (FIPS 46-1)
DID	ARCNET destination MAC address
DLEN	1-octet length of ultimate destination MAC layer address
DNET	2-octet ultimate destination network number
DSAP	LLC destination service access point (X'82' for BACnet)
EIB	European Installation Bus
EIBA	European Installation Bus Association
EXEC	capable of executing a service request
FDT	foreign device table
ICI	interface control information
IL	ARCNET information length field
IND	indication primitive
INF	"Infinity", a unique binary pattern representing positive infinity (see ANSI/IEEE 754-1985)
-INF	"Negative infinity", a unique binary pattern representing negative infinity (see ANSI/IEEE 754-1985)

3. DEFINITIONS

IEEE	Institute of Electrical and Electronics Engineers
INIT	capable of initiating a service request
IP	Internet Protocol - RFC 791
ISO	International Organization for Standardization
KNX	The Konnex System Specification: EIB is the core protocol of the Konnex standard. The Konnex System Specification reflects the current status for EIB.
L	data link (prefix)
LAN	local area network
LLC	logical link control (ISO 8802-2)
LPCI	link protocol control information
LPDU	link protocol data unit
LSAP	link service access point (X'82' for BACnet)
LSDU	link service data unit
M	mandatory
M(=)	mandatory (The parameter is semantically equivalent to the parameter in the service primitive to its immediate left in the table.)
MA	medium access (prefix)
MAC	medium access control
MPCI	MAC protocol control information
MPDU	MAC layer protocol data unit
MSDU	MAC service data unit
MS/TP	master-slave/token-passing
N	network layer (prefix)
NaN	"Not a Number", a unique binary pattern representing an invalid number (see ANSI/IEEE 754-1985)
NP	network priority
NPCI	network protocol control information
NPDU	network layer protocol data unit
NRZ	non-return to zero
NSAP	network service access point
NSDU	network service data unit
O	indicates that support of a property is optional
OSI	open systems interconnection
P	physical layer (prefix)
PAC	ARCNET data packet header octet
PCI	protocol control information
PDU	protocol data unit
PICS	protocol implementation conformance statement
PK	Private Key
PPCI	physical layer protocol control information
PPDU	physical protocol data unit
PPP	Point-To-Point protocol - RFC 1661
PSDU	physical service data unit
PTP	point-to-point
R	indicates that a property shall be supported and readable using BACnet services
REQ	request primitive
RFC	request for comment
RSP	response primitive
S	selection
S(=)	selection (The parameter is semantically equivalent to the parameter in the service primitive to its immediate left in the table.)
SA	local network source MAC layer address
SAP	service access point
SC	ARCNET system code (X'CD' for BACnet)
SDU	service data unit
SID	ARCNET source MAC address
SK	Session Key

SLEN	1-octet length of original source MAC layer address
SLIP	Serial Line Internet Protocol -RFC 1055
SNET	2-octet original source network number
SPC	standard project committee
SSAP	LLC source service access point (X'82' for BACnet)
TSM	transaction state machine
U	user option
U(=)	user option (The parameter is semantically equivalent to the parameter in the service primitive to its immediate left in the table.)
UART	universal asynchronous receiver/transmitter
UDP	User Datagram Protocol - RFC 768
VT	virtual terminal
W	indicates that a property shall be supported, readable, and writable using BACnet services
X'	denotes that hexadecimal notation is used between the single quotes
XID	eXchange IDentification (ISO 8802-2)

4 BACnet PROTOCOL ARCHITECTURE

The Open System Interconnection (OSI) - Basic Reference Model (ISO 7498) is an international standard that defines a model for developing multi-vendor computer communication protocol standards. The OSI model addresses the general problem of computer-to-computer communication and breaks this very complex problem into seven smaller, more manageable sub-problems, each of which concerns itself with a specific communication function. Each of these sub-problems forms a "layer" in the protocol architecture.

The seven layers are arranged in a hierarchical fashion as shown in Figure 4-1. A given layer provides services to the layers above and relies on services provided to it by the layers below. Each layer can be thought of as a black box with carefully defined interfaces on the top and bottom. An application process connects to the OSI application layer and communicates with a second, remote application process. This communication appears to take place between the two processes as if they were connected directly through their application layer interfaces. Minimal knowledge or understanding of the other layers is required. In a similar manner, each layer of the protocol relies on lower layers to provide communication services and establishes a virtual peer-to-peer communication with its companion layer on the other system. The only real connection takes place at the physical layer.

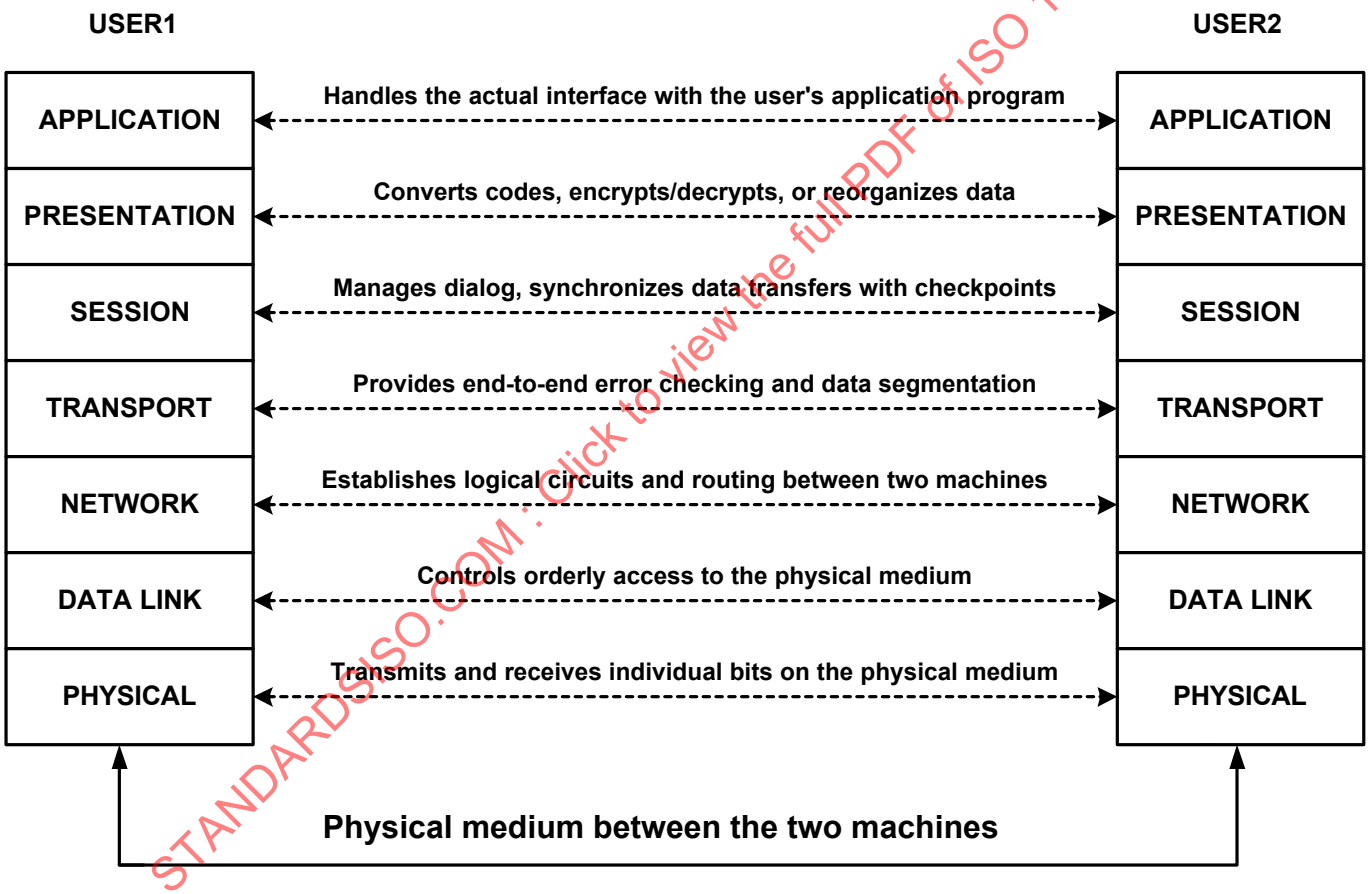


Figure 4-1. The ISO Open Systems Interconnection Basic Reference Model.

The OSI model addresses computer-to-computer communication from a very general perspective. It was designed to deal with the problems associated with computers in large, complex networks communicating with other computers in networks anywhere in the world. In this environment, computers can be separated by long distances and the messages might pass through several intermediate points, each of which may have to make routing decisions or perform some type of translation. Complex synchronization and error recovery schemes may also be needed.

The cost of implementing such a protocol today is prohibitively high for most building automation applications and is not generally required. Nevertheless, the OSI model is a good one to use for a building automation protocol if consideration is

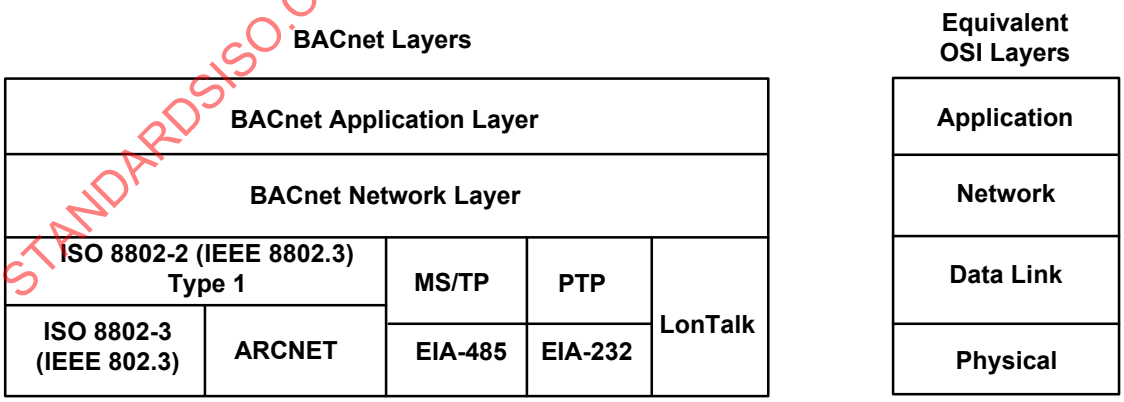
given to including only the OSI functionality that is actually needed, thereby collapsing the seven-layer architecture. In a collapsed architecture, only selected layers of the OSI model are included. The other layers are effectively null, thus reducing message length and communication processing overhead. Such a collapsed architecture permits the building automation industry to take advantage of lower cost, mass-produced processor and local area network technologies such as have been developed for the process control and office automation industries. The use of readily available, widespread technologies, such as Ethernet[1], ARCNET[2], and LonTalk[3], will lower the cost, increase performance, and open new doors to system integration.

4.1 The BACnet Collapsed Architecture

BACnet is based on a four-layer collapsed architecture that corresponds to the physical, data link, network, and application layers of the OSI model as shown in Figure 4-2. The application layer and a simple network layer are defined in the BACnet standard. BACnet provides five options that correspond to the OSI data link and physical layers. Option 1 is the logical link control (LLC) protocol defined by ISO 8802-2 Type 1, combined with the ISO 8802-3 medium access control (MAC) and physical layer protocol. ISO 8802-2 Type 1 provides unacknowledged connectionless service only. ISO 8802-3 is the international standard version of the well-known "Ethernet" protocol. Option 2 is the ISO 8802-2 Type 1 protocol combined with ARCNET (ATA/ANSI 878.1). Option 3 is a Master-Slave/Token-Passing (MS/TP) protocol designed specifically for building automation and control devices as part of the BACnet standard. The MS/TP protocol provides an interface to the network layer that looks like the ISO 8802-2 Type 1 protocol and controls access to an EIA-485 physical layer. Option 4, the Point-To-Point protocol, provides mechanisms for hardwired or dial-up serial, asynchronous communication. Option 5 is the LonTalk protocol. Collectively these options provide a master/slave MAC, deterministic token-passing MAC, high-speed contention MAC, dial-up access, star and bus topologies, and a choice of twisted-pair, coax, or fiber optic media. The details of these options are described in Clauses 7 through 11.

A four-layer collapsed architecture was chosen after careful consideration of the particular features and requirements of BAC networks, including a constraint that protocol overhead needed to be as small as possible. The reasoning behind the selection of the physical, data link, network, and application layers for inclusion in the BACnet architecture is outlined in this subclause.

What layers are required for the proper operation of a BAC network? BAC networks are local area networks. This is true even though in some applications it is necessary to exchange information with devices in a building that is very far away. This long-distance communication is done through telephone networks. The routing, relaying, and guaranteed delivery issues are handled by the telephone system and can be considered external to the BAC network. BAC devices are static. They don't move from place to place and the functions that they are asked to perform do not change in the sense that a manufacturing device may make one kind of part today and some very different part tomorrow. These are among the features of BAC networks that can be used to evaluate the appropriateness of the layers in the OSI model.



[1] Ethernet is a registered trademark of Digital Equipment Corporation, Intel, and Xerox and is the basis for international standard ISO 8802-3.

[2] ARCNET is a registered trademark of Datapoint Corporation.

[3] LonTalk is a registered trademark of Echelon Corporation.

Figure 4-2. BACnet collapsed architecture.

The physical layer provides a means of connecting the devices and transmitting the electronic signals that convey the data. Clearly the physical layer is needed in a BAC protocol.

The data link layer organizes the data into frames or packets, regulates access to the medium, provides addressing, and handles some error recovery and flow control. These are all functions that are required in a BAC protocol. The conclusion is that the data link layer is needed.

Functions provided by the network layer include translation of global addresses to local addresses, routing messages through one or more networks, accommodating differences in network types and in the maximum message size permitted by those networks, sequencing, flow control, error control, and multiplexing. BACnet is designed so that there is only one logical path between devices, thus eliminating the need for optimal path routing algorithms. A network is made up of one or more physical segments connected by repeaters or bridges but with a single local address space. In the case of a single network, most network layer functions are either unnecessary or duplicate data link layer functions. For some BACnet systems, however, the network layer is a necessity. This is the case when two or more networks in a BACnet internet use different MAC layer options. When this occurs, there is a need to recognize the difference between local and global addresses and to route messages to the appropriate networks. BACnet provides this limited network layer capability by defining a network layer header that contains the necessary addressing and control information.

The transport layer is responsible for guaranteeing end-to-end delivery of messages, segmentation, sequence control, flow control, and error recovery. Most of the functions of the transport layer are similar to functions in the data link layer, though different in scope. The scope of transport layer services is end-to-end whereas the scope of data link services is point-to-point across a single network. Since BACnet supports configurations with multiple networks, the protocol must provide the end-to-end services of the transport layer. Guaranteed end-to-end delivery and error recovery are provided in the BACnet application layer via message retry and timeout capabilities. Message segmentation and end-to-end flow control is required for buffer and processor resource management. This is because potentially large amounts of information may be returned for even simple BACnet requests. These functions are provided in the BACnet application layer. Last, sequence control is required in order to properly reassemble segmented messages. This is provided in the BACnet application layer within the segmentation procedure. Since BACnet is based on a connectionless communication model, the scope of the required services is limited enough to justify implementing these at a higher layer, thus saving the communication overhead of a separate transport layer.

The session layer is used to establish and manage long dialogues between communicating partners. Session layer functions include establishing synchronization checkpoints and resetting to previous checkpoints in the event of error conditions to avoid restarting an exchange from the beginning. Most communications in a BAC network are very brief. For example, reading or writing one or a few values, notifying a device about an alarm or event, or changing a setpoint. Occasionally longer exchanges take place, such as uploading or downloading a device. The few times when the services of this layer would be helpful do not justify the additional overhead that would be imposed on the vast majority of transactions, which are very brief and do not need them.

The presentation layer provides a way for communicating partners to negotiate the transfer syntax that will be used to conduct the communication. This transfer syntax is a translation from the abstract user view of data at the application layer to sequences of octets treated as data at the lower layers. If only one transfer syntax is permitted, then the presentation layer function reduces to an encoding scheme for representing the application data. BACnet defines such a fixed encoding scheme and includes it in the application layer, making an explicit presentation layer unnecessary.

The application layer of the protocol provides the communication services required by the applications to perform their functions, in this case monitoring and control of HVAC&R and other building systems. Clearly an application layer is needed in the protocol.

In summary:

(a) The resource and overhead costs for implementing a full OSI seven-layer architecture make it impractical for current building automation devices.

(b) Following the OSI model offers advantages in terms of adopting existing computer networking technology. This can result in cost savings and make integration with other computer network systems easier.

(c) The expectations and environment of building automation systems permit simplification of the OSI model by eliminating the functionality of some of the layers.

(d) A collapsed architecture made up of the physical, data link, network, and application layers is the optimum solution for today's building automation systems.

4.2 BACnet Network Topology

In the interest of application flexibility, BACnet does not prescribe a rigid network topology. Rather, BACnet devices are physically connected to one of four types of local area networks (LANs) or via dedicated or dial-up serial, asynchronous lines. These networks may then be further interconnected by BACnet routers as described in Clause 6.

In terms of LAN topology, each BACnet device is attached to an electrical medium or *physical segment*. A BACnet *segment* consists of one or more physical segments connected at the physical layer by *repeaters*. A BACnet *network* consists of one or more segments interconnected by *bridges*, devices that connect the segments at the physical and data link layers and may perform message filtering based upon MAC addresses; a network forms a single MAC address domain. Multiple networks, possibly employing different LAN technologies, may be interconnected by BACnet *routers* to form a BACnet *internetwork*. In a BACnet internetwork, there exists exactly one message path between any two nodes. These concepts are shown graphically in Figure 4-3.

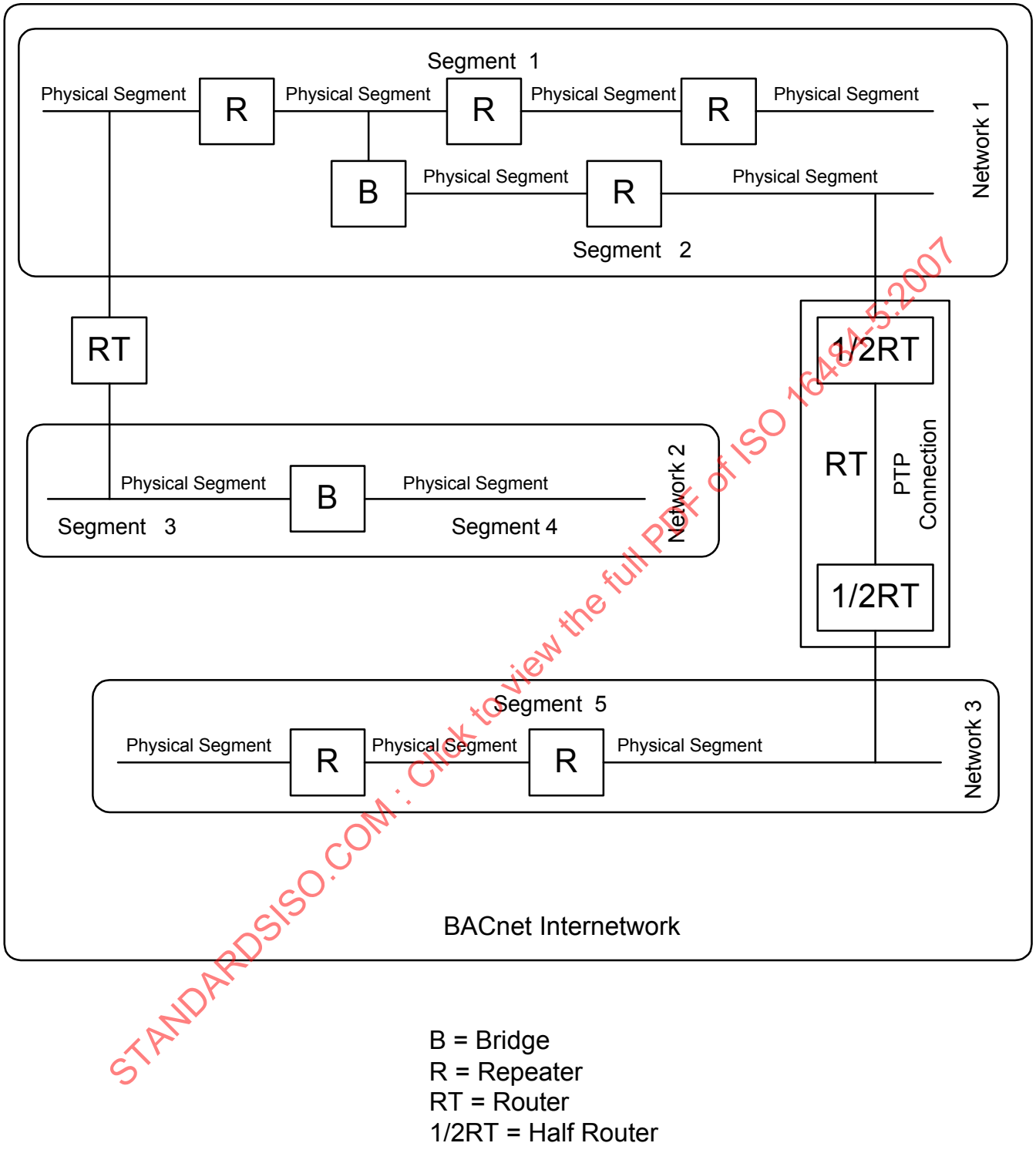


Figure 4-3. A BACnet internetwork illustrating the concepts of Physical Segments, Repeaters, Segments, Bridges, Networks, Half Routers, and Routers.

4.3 Security

The principal security threats to BACnet systems are people who, intentionally or by accident, modify a device's configuration or control parameters. Problems due to an errant computer are outside the realm of security considerations. One important place for security measures is the operator-machine interface. Since the operator-machine interface is not part of the communication protocol, vendors are free to include password protection, audit trails, or other controls to this interface as needed. In addition, write access to any properties that are not explicitly required to be "writable" by this standard may be restricted to modifications made only in virtual terminal mode or be prohibited entirely. This permits vendors to protect key properties with a security mechanism that is as sophisticated as they consider appropriate. BACnet also defines services that can be used to provide peer entity, data origin, and operator authentication. See Clause 24.

5 THE APPLICATION LAYER

5.1 The Application Layer Model

This clause presents a model of the BACnet application layer. The purpose of the model is to describe and illustrate the interaction between the application layer and application programs, the relationship between the application layer and lower layers in the protocol stack, and the peer-to-peer interactions with a remote application layer. This model is not an implementation specification.

An Application Process is that functionality within a system that performs the information processing required for a particular application. All parts of the Application Process outside the Application Layer, (i.e., those that do not concern the communication function) are outside the scope of BACnet. The part of the Application Process that is within the Application Layer is called the Application Entity. In other words, an Application Entity is that part of the Application Process related to the BACnet communication function. An application program interacts with the Application Entity through the Application Program Interface (API). This interface is not defined in BACnet, but it would probably be a function, procedure, or subroutine call in an actual implementation. These concepts are illustrated in Figure 5-1. The shaded region indicates the portion of the Application Process that is within the BACnet Application Layer.

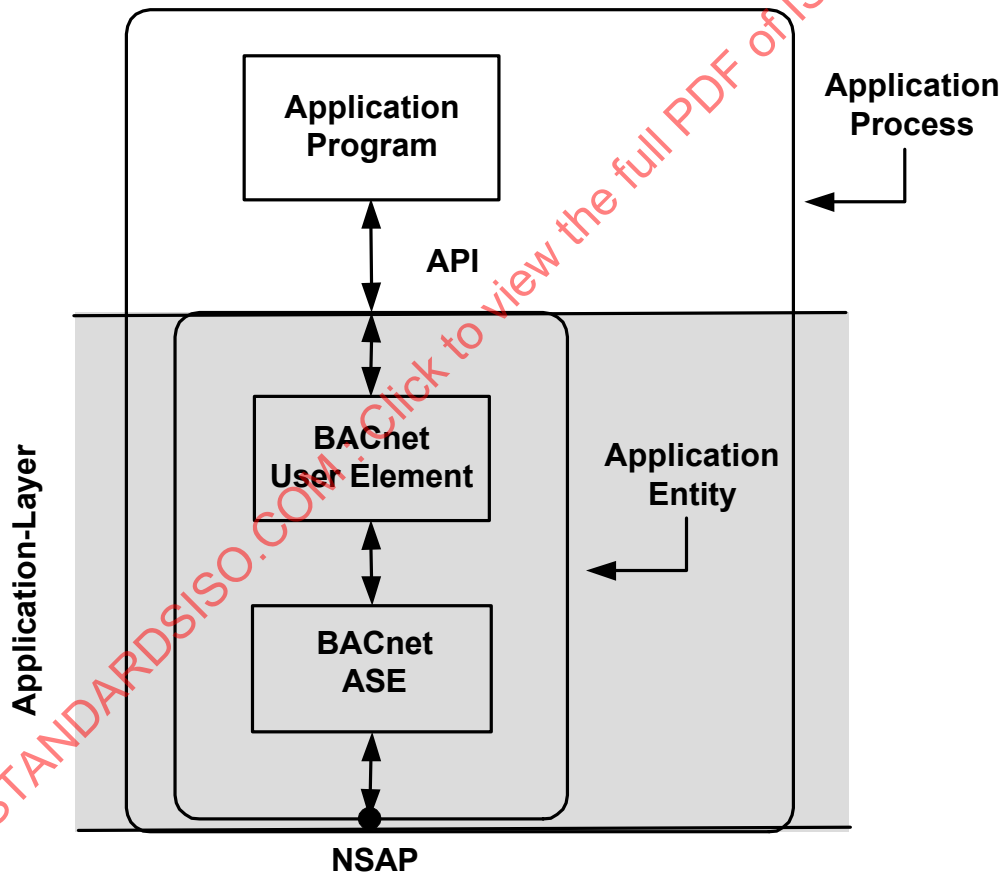


Figure 5-1. Model of a BACnet Application Process.

The Application Entity is itself made up of two parts: the BACnet User Element and the BACnet Application Service Element (ASE). The BACnet ASE represents the set of functions or application services specified in Clauses 13 through 17 and Clause 24. The BACnet User Element carries out several functions in addition to supporting the local API. It represents the implementation of the "service procedure" portion of each application service. It is responsible for maintaining information about the context of a transaction, including generating invoke IDs and remembering which invoke ID goes with which application service request (response) to (from) which device. It is also responsible for maintaining the time-out

counters that are required for the retrying of a transmission. The BACnet User Element also presides over the mapping of a device's activities into BACnet objects.

Information exchanged between two peer application processes is represented in BACnet as an exchange of abstract service primitives, following the ISO conventions contained in the OSI technical report on service conventions, ISO TR 8509. These primitives are used to convey service-specific parameters that are defined in Clauses 13 through 17 and Clause 24. Four service primitives are defined: request, indication, response, and confirm. The information contained in the primitives is conveyed using a variety of protocol data units (PDUs) defined in this standard. In order to make clear which BACnet PDU is being used, the notation will be as follows:

CONF_SERV.request	CONF_SERV.indication	CONF_SERV.response	CONF_SERV.confirm
UNCONF_SERV.request	UNCONF_SERV.indication		
SEGMENT_ACK.request	SEGMENT_ACK.indication		
REJECT.request	REJECT.indication		
ABORT.request	ABORT.indication		

The designation CONF_SERV indicates that BACnet confirmed service PDUs are being used. Similarly, the designations UNCONF_SERV, SEGMENT_ACK, ERROR, REJECT, and ABORT indicate that unconfirmed service PDUs, segment acknowledge PDUs, error PDUs, reject PDUs, and abort PDUs, respectively, are being used.

An application program that needs to communicate with a remote application process accesses the local BACnet User Element through the API. Some of the API parameters, such as the identity (address) of the device to which the service request is to be sent and protocol control information, is passed directly down to the network or data link layers. The remainder of the parameters make up an application service primitive that is passed from the BACnet User Element to the BACnet ASE. Conceptually, the application service primitive results in the generation of an APDU that becomes the data portion of a network service primitive, which is passed to the network layer through the Network Service Access Point (NSAP). Similarly this request passes down through the lower layers of the protocol stack in the local device. This process is illustrated in Figure 5-2. The message is then transmitted to the remote device, where it is passed up through the protocol stack in the remote device, eventually appearing as an indication primitive passed from the remote BACnet ASE to the remote BACnet User Element. The response from the remote device, if any, returns to the initiator of the service in a similar fashion (see 5.5).

In addition to the service primitives and the service specific parameters, the application entity exchanges interface control information (ICI) parameters with the application program via the API. The content of the ICI is dependent upon the service primitive type. The ICI parameters received by the application entity provide the information that is passed on to the lower layers (as ICI across layer interfaces) to help them construct their PDUs. The ICI parameters that are provided by the application entity to the application programs contain information recovered by the lower layers from their respective PDUs.

The following ICI parameters are exchanged with the various service primitives across an API:

'destination_address' (DA): the address of the device(s) intended to receive the service primitive. Its format (device name, network address, etc.) is a local matter. This address may also be a multicast, local broadcast or global broadcast type.

'source_address' (SA): the address of the device from which the service primitive was received. Its format (device name, network address, etc.) is a local matter.

'network_priority' (NP): a four-level network priority parameter described in 6.2.2.

'data_expecting_reply' (DER): a Boolean parameter that indicates whether (TRUE) or not (FALSE) a reply service primitive is expected for the service being issued.

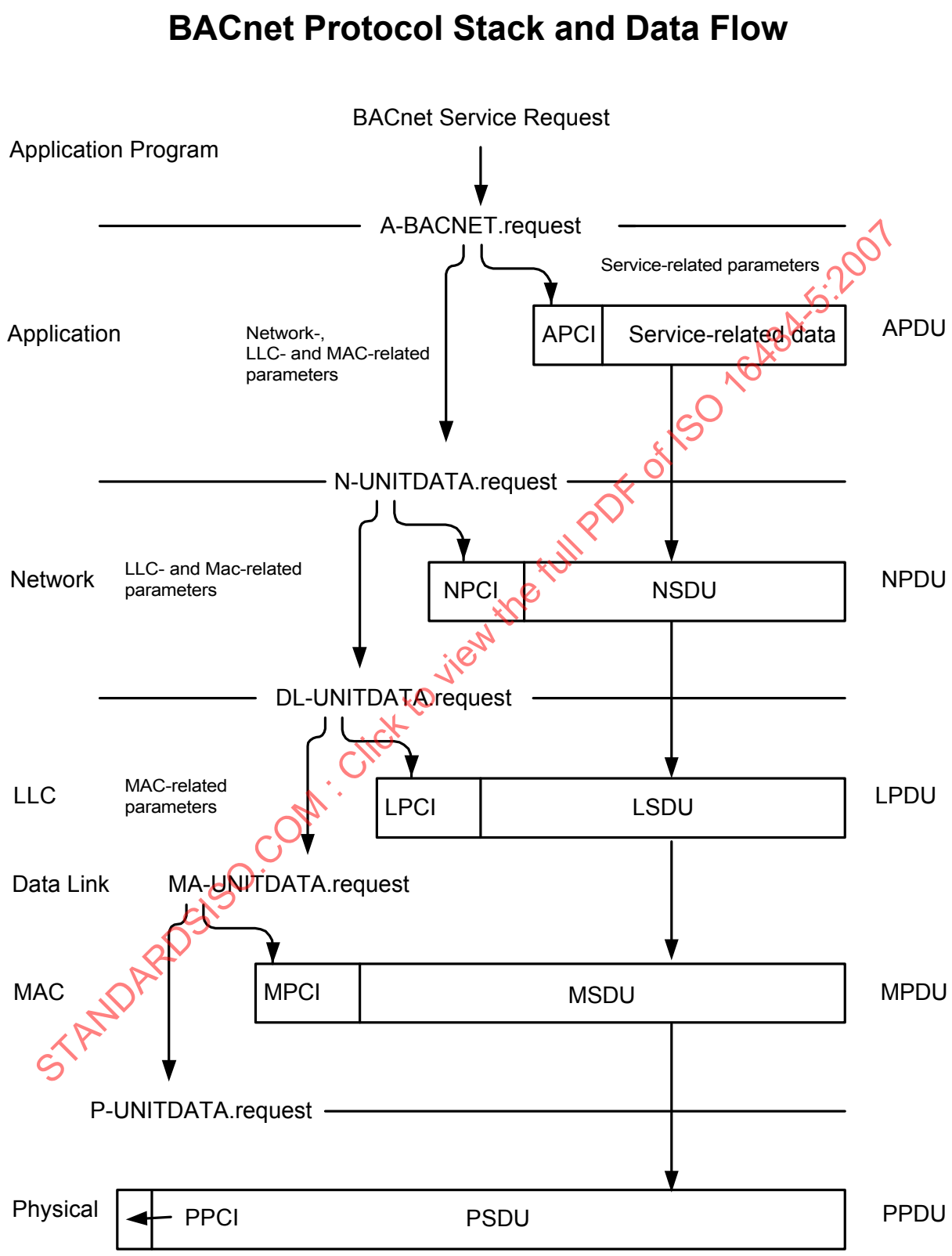


Figure 5-2. BACnet protocol stack and data flow.

Table 5-1 describes the applicability of the ICI parameters to the service primitives.

Table 5-1. Applicability of ICI parameters for abstract service primitives

Service Primitive	DA	SA	NP	DER
CONF_SERV.request	Yes	No	Yes	Yes
CONF_SERV.indication	Yes	Yes	Yes	Yes
CONF_SERV.response	Yes	No	Yes	Yes
CONF_SERV.confirm	Yes	Yes	Yes	No
UNCONF_SERV.request	Yes	No	Yes	No
UNCONF_SERV.indication	Yes	Yes	Yes	No
REJECT.request	Yes	No	Yes	No
REJECT.indication	Yes	Yes	Yes	No
SEGMENT_ACK.request	Yes	No	Yes	No
SEGMENT_ACK.indication	Yes	Yes	Yes	No
ABORT.request	Yes	No	Yes	No
ABORT.indication	Yes	Yes	Yes	No

A "BACnetDevice" is any device, real or virtual, that supports digital communication using the BACnet protocol. Each BACnet Device contains exactly one Device object, as defined in 12.11. A BACnet Device is uniquely located by an NSAP, which consists of a network number and a MAC address.

In most cases, a physical device will implement a single BACnet Device. It is possible, however, that a single physical device may implement a number of "virtual" BACnet Devices. This is described in Annex H.

5.1.1 Confirmed Application Services

BACnet defines confirmed application services based on a client and server communication model. A client requests service from a server via a particular service request instance. The server provides service to a client and responds to the request. This relationship is illustrated in Figure 5-3. The BACnet-user that assumes the role of a client is called the "requesting BACnet-user" and the BACnet-user that assumes the role of the server is called the "responding BACnet-user."

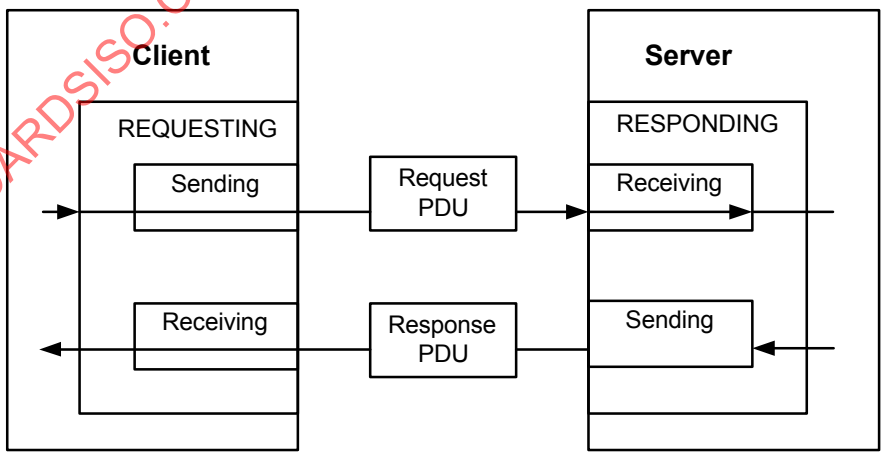


Figure 5-3. Relationship of a client and server.

A requesting BACnet-user issues a CONF_SERV.request primitive, which causes a request PDU to be sent. When a response PDU arrives, the requesting BACnet-user receives a CONF_SERV.confirm primitive. When a request PDU arrives, the

responding BACnet-user receives a CONF_SERV.indication primitive. The responding BACnet-user issues a CONF_SERV.response primitive, which causes a response PDU to be sent. Thus, the requesting BACnet-user and the responding BACnet-user play a role in both sending and receiving PDUs. The term "sending BACnet-user" applies to a BACnet user that initiates the sending of a PDU. The term "receiving BACnet-user" applies to a BACnet-user that receives an indication that a PDU has arrived.

5.1.2 Unconfirmed Application Services

The client and server model, and the terms "requesting BACnet-user" and "responding BACnet-user," do not apply to unconfirmed services. The terms "sending BACnet-user" and "receiving BACnet-user" do apply, however, and they are used to define the service procedure for unconfirmed services.

5.2 Segmentation of BACnet Messages

To provide for messages that are longer than the maximum length supported by a communications network, or by the sending or receiving device, BACnet provides a method to perform application layer segmentation. In BACnet, only Confirmed-Request and ComplexACK messages may be segmented. Segmentation is an optional feature of BACnet.

5.2.1 Message Segmentation Rules

This subclause prescribes rules for dividing a message into segments.

5.2.1.1 Rules for Segmenting APDU Data Streams

Each BACnet message is encoded into a sequence of tags and values according to the relevant ASN.1 definitions in Clause 21 and the encoding rules of Clause 20. The following rules apply to segmenting this data stream:

(a) If possible, an entire message shall be sent in a single APDU.

(b) If an entire message cannot be sent in a single APDU, the message shall be segmented into the minimum number of APDUs possible.

(c) Messages shall be segmented only at octet boundaries.

5.2.1.2 Maximum APDU Length

The maximum length of a BACnet APDU shall be the smallest of

(a) the maximum APDU size transmittable by a device, which may be restricted by local buffer limitations and is a local matter;

(b) the maximum APDU size conveyable by the internetwork to the remote device, which is constrained by the maximum NPDU length permitted by the data links used by the local, remote, and any intervening networks, as specified in Clause 6;

(c) the maximum APDU size accepted by the remote peer device, which must be at least 50 octets.

If the sending device is the requesting BACnet-user, i.e., the APDU to be sent is a BACnet-Confirmed-Request-PDU or a BACnet-Unconfirmed-Request-PDU, then the maximum APDU size accepted by the remote peer is specified by the Max_APDU_Length_Accepted property of the remote peer's Device object. The value of this property may be read using the read property services described in Clause 15 or the value may be obtained from the 'Max APDU Length Accepted' parameter of an I-Am service request received from the remote device. The remote peer may be solicited to transmit an I-Am service request by sending it a Who-Is service request, as described in 16.9

If the sending device is not the requesting BACnet-user, i.e., the APDU to be sent is a BACnet-ComplexACK-PDU, then the maximum APDU size accepted by the remote peer is specified in the 'Max APDU Length Accepted' parameter of the BACnet-Confirmed-Request-PDU for which this is a response.

The value determined by the above constraints will be designated the maximum-transmittable-length. Note that maximum-transmittable-length will in general not be a constant unless minimum values are used for each constraint.

5.2.1.3 Maximum Segments Accepted

The maximum number of segments transmitted in a Confirmed-Request or ComplexACK message shall be the smallest of:

(a) the maximum number of segments transmittable by a device, which may be restricted by local limitations and is a local matter;

(b) the maximum number of segments accepted by the remote peer device.

If the sending device is the requesting BACnet-user, i.e., the message to be sent is a Confirmed-Request, then the maximum number of segments accepted by the remote peer device is specified in the Max_Segments_Accepted property of the remote peer's Device object.

If the sending device is not the requesting BACnet-user, i.e., the message to be sent is a ComplexACK, then the maximum number of segments accepted by the remote peer device is specified in the 'Max Segments Accepted' parameter of the BACnet-Confirmed-Request-PDU for which this is a response.

5.2.2 Segmentation Protocol Control Information (PCI)

To provide for the possibility of segmented messages, the headers of the BACnet-Confirmed-Request-PDU and BACnet-ComplexACK-PDU contain two Boolean parameters called 'Segmented Message' and 'More Follows.'

If the length of a fully encoded message of the type conveyed by one of the above APDUs results in an APDU whose length is less than or equal to the maximum-transmittable-length as determined according to 5.2.1, the 'Segmented Message' and 'More Follows' parameters shall both be set to FALSE.

If, however, the encoded length of a message would result in an APDU length greater than the maximum-transmittable-length as determined according to 5.2.1, the 'Segmented Message' parameter shall be set to TRUE for all segments, and the 'More Follows' parameter shall be set to TRUE for all segments except the last.

Two additional parameters are also present, conditionally, in the header of each APDU carrying a segment of a Confirmed-Request message or a ComplexACK message. The first conditional parameter is the 'Sequence Number.' This one-octet unsigned integer is used by the segment transmitter to indicate the position of the current segment in the series of segments composing the complete message. The second conditional parameter is the 'Proposed Window Size.' This one-octet unsigned integer is used by the segment transmitter to indicate the maximum number of message segments that it is prepared to transmit before it must receive a SegmentACK. The use of these parameters in the transmission of segmented messages is described in 5.3 and 5.4.

The 'Sequence Number' of the initial segment shall be zero. The segment receiver may request the transmission of the next segment or group of segments by sending a SegmentACK-PDU containing the 'Sequence Number' parameter of the last successfully received segment. Such a request shall also serve as an acknowledgment of this segment. If the Window Size is greater than one, such a SegmentACK-PDU shall also serve to acknowledge any previously transmitted but unacknowledged segments.

If either party in a segmented transaction wishes to terminate the transaction, that party may issue an Abort-PDU.

5.3 Transmission of BACnet APDUs

The formal description of the transmission and reception protocol for BACnet APDUs is contained in the Transaction State Machine description given in 5.4. This subclause is intended only as an overview of the protocol.

5.3.1 Confirmed-Request Message Transmission

Upon transmitting a complete unsegmented Confirmed-Request message or upon receiving the SegmentACK acknowledging the final segment of a segmented Confirmed-Request message, a client device shall start a timer that indicates the length of time the message has been outstanding. The timer shall be canceled upon the receipt of an Error, Reject, Abort, SimpleACK, or ComplexACK APDU for the outstanding Confirmed-Request message, and the client application shall be notified. If the timer exceeds the value of the APDU_Timeout property in the client's Device object, then the complete Confirmed-Request message shall be retransmitted and the timer shall be reset to zero. All retransmitted Confirmed-Request messages shall follow this same procedure until the message has been retransmitted the number of times indicated in the

Number_Of_APDU_Retries property of the client's Device object. If, after the Confirmed-Request message is retransmitted the appropriate number of times, a response is still not received, the message shall be discarded and the client application shall be notified.

5.3.2 Segmented Confirmed-Request Message Transmission

Before sending the first segment of a segmented Confirmed-Request-PDU, a client device shall choose a Proposed Window Size to indicate the maximum number of message segments it is prepared to transmit before it must receive a SegmentACK. The means of choosing the Proposed Window Size are a local matter, except that the value shall be in the range 1 to 127, inclusive. The Proposed Window Size shall be carried by the parameter of that name in each segment of the Confirmed-Request-PDU. The value of Proposed Window Size shall be the same in each segment of the Confirmed-Request-PDU.

Upon transmitting the first segment of a Confirmed-Request message, a client device shall start a timer that indicates the length of time this message segment has been outstanding. The timer shall be canceled upon the receipt of a Reject, Abort, or SegmentACK APDU for the outstanding Confirmed-Request message segment. If the timer exceeds the value of the APDU_Segment_Timeout property in the client's Device object, then the segment shall be retransmitted and the timer shall be reset to zero. All retransmitted segments shall follow this same procedure until the message segment has been retransmitted the number of times indicated in the Number_Of_APDU_Retries property of the client's Device object. If, after the message segments are retransmitted the appropriate number of times, a response is still not received, the message shall be discarded and the client application shall be notified.

Upon receipt of the first segment of a segmented Confirmed-Request-PDU, the server device shall choose an Actual Window Size to indicate the number of sequential message segments it expects to receive before it transmits a SegmentACK. The means of choosing the Actual Window Size are a local matter, except that the value shall be less than or equal to the 'proposed-window-size' parameter contained in the Confirmed-Request-PDU and shall be in the range 1 to 127, inclusive. The value of Actual Window Size shall be the same in each SegmentACK sent in response to a given Confirmed-Request. Regardless of the value of Actual Window Size, a SegmentACK shall be sent in response to the first segment of a Confirmed-Request.

Upon receipt of a SegmentACK APDU, the client device shall set its Actual Window Size equal to the value associated with the 'actual-window-size' parameter in the SegmentACK APDU. After this point, the client has authorization to send as many segments as the 'actual-window-size' parameter indicates before waiting for a SegmentACK APDU. No more than T_{seg} may be allowed to elapse between the receipt of a SegmentACK APDU and the transmission of a segment. No more than T_{seg} may be allowed to elapse between the transmission of successive segments of a group. After transmitting a set of segments that fills the window or completes the message, a client device shall start a timer that indicates the length of time these message segments have been outstanding. The timer shall be canceled upon receipt of a Reject, Abort, or SegmentACK APDU for some or all of the outstanding Confirmed-Request message segments. If the timer exceeds the value of the APDU_Segment_Timeout property in the client's Device object, then the segments shall be retransmitted and the timer shall be reset to zero. All retransmitted segments shall follow this same procedure until the message segments have been retransmitted the number of times indicated in its Device object's Number_Of_APDU_Retries property. If, after the Confirmed-Request message segments are retransmitted the appropriate number of times, a response is still not received, the message shall be discarded and the client application shall be notified.

It is possible to receive a Reject, Abort, or SegmentACK APDU during the sending of a sequence of Confirmed-Request segments even though the number of outstanding segments is less than indicated by the Actual Window Size. In this case, receipt of a Reject or Abort APDU shall terminate the Confirmed-Request transaction. Receipt of a SegmentACK APDU shall be considered as an acknowledgment for the segments up to and including the number indicated in the 'sequence-number' parameter of the SegmentACK APDU. Any unacknowledged segments shall be retransmitted following the above procedure.

It is recognized that in some cases where a Reject, Abort, or SegmentACK APDU is received, the client device may have sent, or irretrievably queued for sending, one or more (but less than Actual Window Size) additional Confirmed-Request-PDU segments.

5.3.3 Segmented ComplexACK Message Transmission

Before sending the first segment of a segmented ComplexACK-PDU, a server device shall choose a Proposed Window Size to indicate the maximum number of message segments it is prepared to transmit before it must receive a SegmentACK. The

means of choosing the Proposed Window Size are a local matter, except that the value shall be in the range 1 to 127, inclusive. The Proposed Window Size shall be carried by the parameter of that name in each segment of the Confirmed-Request-PDU. The value of Proposed Window Size shall be the same in each segment of the ComplexACK-PDU.

Upon transmitting the first segment of a ComplexACK message, a server device shall start a timer that indicates the length of time this message segment has been outstanding. The timer shall be canceled upon the receipt of an Abort or SegmentACK APDU for the outstanding ComplexACK message segment. If the timer exceeds the value of the APDU_Segment_Timeout property in the server's Device object, then the segment shall be retransmitted and the timer shall be reset to zero. All retransmitted segments shall follow this same procedure until the message segment has been retransmitted the number of times indicated in the Number_Of_APDU_Retries property of the server's Device object. If, after the message segments are retransmitted the appropriate number of times, a response is still not received, the message shall be discarded.

Upon receipt of the first segment of a segmented ComplexACK-APDU, the client device shall choose an Actual Window Size to indicate the number of sequential message segments it expects to receive before it transmits a SegmentACK. The means of choosing the Actual Window Size are a local matter, except that the value shall be less than or equal to the 'proposed-window-size' parameter contained in the ComplexACK-PDU and shall be in the range 1 to 127, inclusive. The value of Actual Window Size shall be the same in each SegmentACK sent in response to a given ComplexACK. Regardless of the value of Actual Window Size, a SegmentACK shall be sent in response to the first segment of a ComplexACK.

Upon receipt of a SegmentACK APDU, the server device shall set its Actual Window Size equal to the value associated with the 'actual-window-size' parameter in the SegmentACK APDU. After this point, the server has authorization to send as many segments as the 'actual-window-size' parameter indicates before waiting for a SegmentACK APDU. No more than T_{seg} may be allowed to elapse between the receipt of a SegmentACK APDU and the transmission of a segment. No more than T_{seg} may be allowed to elapse between the transmission of successive segments of a group. After transmitting a set of segments that fills the window or completes the message, a server device shall start a timer that indicates the length of time these message segments have been outstanding. The timer shall be canceled upon receipt of an Abort or SegmentACK APDU for some or all of the outstanding ComplexACK message segments. If the timer exceeds the value of the APDU_Segment_Timeout property in the server's Device object, then the segments shall be retransmitted and the timer shall be reset to zero. All retransmitted segments shall follow this same procedure until the message segments have been retransmitted the number of times indicated in the Number_Of_APDU_Retries property of the server's Device object. If, after the ComplexACK message segments are retransmitted the appropriate number of times, a response is still not received, the message shall be discarded.

It is possible to receive an Abort or SegmentACK APDU during the sending of a sequence of ComplexACK segments even though the number of outstanding segments is less than indicated by the Actual Window Size. In this case, receipt of an Abort APDU shall terminate the ComplexACK transaction. Receipt of a SegmentACK APDU shall be considered as an acknowledgment for the segments up to and including the number indicated in the 'sequence-number' parameter of the SegmentACK APDU. Any unacknowledged segments shall be retransmitted following the above procedure.

It is recognized that in some cases where an Abort or SegmentACK APDU is received, the server device may have sent, or irretrievably queued for sending, one or more (but less than Actual Window Size) additional ComplexACK segments.

5.3.4 SegmentACK APDU Transmission

A device shall transmit a SegmentACK upon any of the following conditions:

(a) The device receives the initial segment of a segmented message. In this case, the 'negative-ACK' parameter of the SegmentACK shall have a value of FALSE, indicating that this is a positive acknowledgment, and the 'sequence-number' parameter of the SegmentACK shall have a value of zero, indicating that the first segment has been acknowledged and that the segment transmitter may continue sending, commencing with the next sequential segment.

(b) The device receives a quantity of unacknowledged, sequentially numbered segments for this transaction equal to the Actual Window Size. In this case, the 'negative-ACK' parameter of the SegmentACK shall have a value of FALSE, indicating that this is a positive acknowledgment, and the 'sequence-number' parameter of the SegmentACK shall have a value equal to the 'sequence-number' parameter of the last received segment, indicating that all segments up to and including 'sequence-number' have been acknowledged and that the segment transmitter may continue sending, commencing with the next sequential segment.

(c) The device receives a segment out of order (possibly indicating that a segment has been missed). In this case, the segment receiver shall discard the out-of-order segment. In this context, "out of order" means a segment whose 'sequence-number' is not equal to the next expected 'sequence-number.' The 'negative-ACK' parameter of the SegmentACK shall have a value of TRUE, indicating that this is a negative acknowledgment. The 'sequence-number' parameter of the SegmentACK shall have a value equal to the 'sequence-number' parameter of the last received correctly ordered segment, indicating that all segments up to and including 'sequence-number' have been acknowledged and that the segment transmitter should resend, commencing with the next sequential segment after that indicated by the 'sequence-number' parameter contained in the SegmentACK.

(d) The device receives the final segment of a message. In this case, the 'negative-ACK' parameter of the SegmentACK shall have a value of FALSE, indicating that this is a positive acknowledgment, and the 'sequence-number' parameter of the SegmentACK shall have a value equal to the 'sequence-number' parameter of the final message segment, indicating that all segments up to and including the final segment have been acknowledged.

5.3.5 Duplicate APDUs and Message Segments

5.3.5.1 Terminating Client TSMs

When using the BACnet error recovery procedures there is a possibility of the reception of duplicate messages or message segments during a transaction. At the client, a transaction begins, and a Transaction State Machine is created, when the first or only segment of a Confirmed-Request APDU is sent. The transaction ends when the client discards the Transaction State Machine due to one of the following circumstances:

(a) after reception from the server of a SimpleACK, unsegmented ComplexACK, Error, Reject, or Abort APDU containing the transaction's invokeID;

(b) after transmission to the server of a SegmentACK APDU for the final segment of a segmented ComplexACK APDU received from the server;

(c) after exhausting the timeout and retry logic described in the previous subclauses;

(d) after transmission to the server of an Abort APDU containing the transaction's invokeID (i.e., the client aborts the transaction).

5.3.5.2 Terminating Server TSMs

At the server, a transaction begins, and a Transaction State Machine is created when the first or only segment of a Confirmed-Request APDU is received. The transaction ends when the server discards the Transaction State Machine due to one of the following circumstances:

(a) after transmission to the client of a SimpleACK, unsegmented ComplexACK, Error, Reject, or Abort APDU containing the transaction's invokeID;

(b) after reception from the client of a SegmentACK APDU for the final segment of a segmented ComplexACK APDU transmitted by the server;

(c) after reception from the client of an Abort APDU containing the transaction's invokeID;

(d) after exhausting the timeout and retry logic described in the previous subclauses during the transmission of a segmented ComplexACK APDU.

5.3.5.3 Duplicate Message Procedures

The procedure for handling duplicate messages and message segments is as follows:

(a) The server receives a duplicate Confirmed-Request message. If the server has the capability of detecting a duplicate Confirmed-Request message, the message shall be discarded. If the server cannot distinguish between duplicate and

non-duplicate messages, then the Confirmed-Request message shall be serviced. In this case, the client shall discard the server's response since the Invoke ID of the response will not bind to an active state transaction state machine.

(b) The server receives a duplicate Confirmed-Request message segment, that is, one that has already been acknowledged with a SegmentACK. In this case, the server shall discard the duplicate segment but shall return an appropriate SegmentACK APDU. A segment can be identified uniquely by the peer address, Invoke ID, and Sequence Number of the segment.

(c) The client receives a duplicate ComplexACK segment, that is, one that has already been acknowledged with a SegmentACK. In this case, the client shall discard the duplicate segment but shall return an appropriate SegmentACK APDU. A segment can be identified uniquely by the peer address, Invoke Id, and Sequence Number of the segment.

(d) A Device receives a duplicate SegmentACK APDU. In this case, the device shall discard the duplicate SegmentACK APDU. Other actions, including the possible re-sending of message segments, shall occur as specified in 5.4.

5.3.6 Stale Resource Disposal

The error recovery procedure described here requires resources from both the server and the client. In the event that the error recovery process fails, the resources dedicated to this process need to be freed. In general, the resources that need to be freed are transaction specific and consist of a Transaction State Machine (TSM), timers, and APDU or APDU segment buffers. The exact time period before the resources should be freed is a local matter dependent upon the system design. As a design suggestion, it is recommended that resources should be considered stale and consequently freed:

(a) at the client, when a complete response to the Confirmed-Request APDU is received;

(b) at the client, when a Confirmed-Request APDU has been retransmitted the number of times specified in the Number_Of_APDU_Retries property without success;

(c) at the client, when a Confirmed-Request APDU segment has been retransmitted the number of times specified in the Number_Of_APDU_Retries property without success;

(d) at the server, when a complete response to a Confirmed-Request APDU has been transmitted and any associated SegmentACK received;

(e) at the server, when a ComplexACK APDU segment has been retransmitted the number of times specified in the Number_Of_APDU_Retries property without success;

(f) at any device, when a SegmentACK APDU has been transmitted and additional segments have not been received before the segment-timeout expires.

5.4 Application Protocol State Machines

BACnet APDUs may be divided into two classes: those sent by requesting BACnet-users (clients) and those sent by responding BACnet-users (servers). All BACnet devices shall be able to act as responding BACnet-users and therefore shall be prepared to receive APDUs sent by requesting BACnet-users. Many devices will also be able to act as requesting BACnet-users, and such devices shall be prepared to receive APDUs sent by responding BACnet-users.

APDUs sent by requesting BACnet-users (clients):

BACnet-Unconfirmed-Request-PDU
BACnet-Confirmed-Request-PDU
BACnet-SegmentACK-PDU with 'server' = FALSE
BACnet-Abort-PDU with 'server' = FALSE

APDUs sent by responding BACnet-users (servers):

BACnet-SimpleACK-PDU
BACnet-ComplexACK-PDU
BACnet-Error-PDU
BACnet-Reject-PDU
BACnet-SegmentACK-PDU with 'server' = TRUE
BACnet-Abort-PDU with 'server' = TRUE

Both the requesting and the responding BACnet-user shall create and maintain a Transaction State Machine (TSM) for each transaction. The TSM shall be created when the transaction begins and shall be disposed of when the transaction ends. In the state machine descriptions that follow, the creation of a TSM is represented by a transition out of the IDLE state, and the disposal of a TSM is represented by a transition into the IDLE state. A transaction is uniquely identified by the client BACnetAddress, the server BACnetAddress, and the Invoke ID (if any).

When a PDU is received from the network layer, the PDU type, the source and destination BACnetAddresses, and the Invoke ID (if any) of the PDU shall be examined to determine the type (requesting BACnet-user or responding BACnet-user) and the identity of the TSM to which the PDU shall be passed. If no such TSM exists, one shall be created.

When a request is received from the application program, the request type, the source and destination BACnetAddresses, and the Invoke ID (if any) of the request shall be examined to determine the type (requesting BACnet-user or responding BACnet-user) and the identity of the TSM to which the request shall be passed. If no such TSM exists, one shall be created.

In order to simplify the state machine description, only the case of segmentation by the Application Entity is shown. Segmentation by the Application Program is possible as well. In this case, wherever the current TSM receives a segment or group of segments and sends SegmentACK, the modified TSM would instead pass the segments to the Application Program, and SegmentACK would be sent only upon direction from the Application Program via the SEGMENT_ACK.request primitive. Reception by the modified state machine of a SegmentACK-PDU would cause it to pass a SEGMENT_ACK.indication primitive to the Application Program.

5.4.1 Variables And Parameters

The following variables are defined for each instance of Transaction State Machine:

RetryCount used to count APDU retries

SegmentRetryCount used to count segment retries

SentAllSegments used to control APDU retries and the acceptance of server replies

LastSequenceNumber stores the sequence number of the last segment received in order

InitialSequenceNumber stores the sequence number of the first segment of a sequence of segments that fill a window

ActualWindowSize stores the current window size

ProposedWindowSize stores the window size proposed by the segment sender

SegmentTimer used to perform timeout on PDU segments

RequestTimer used to perform timeout on Confirmed Requests

The following parameters are used in the description:

$\mathbf{T}_{seg}$ This parameter is the length of time a node shall wait for a SegmentACK-PDU after sending the final segment of a sequence. Its value is the value of the APDU_Segment_Timeout property of the node's Device object.

$\mathbf{T}_{wait\_for\_seg}$ This parameter is the length of time a node shall wait after sending a SegmentACK-PDU for an additional segment of the message. Its value is equal to four times the value of the APDU_Segment_Timeout property of the node's Device object.

$\mathbf{T}_{out}$ This parameter represents the value of the APDU_Timeout property of the node's Device object.

$\mathbf{N}_{retry}$ This parameter represents the value of the Number_Of_APDU_Retries property of the node's Device object.

5.4.2 Function InWindow

The function "InWindow" performs a modulo 256 compare of two unsigned eight-bit sequence numbers. All computations and comparisons are modulo 256 operations on unsigned eight-bit quantities.

function InWindow(seqA, seqB)

(a) if seqA minus seqB, modulo 256, is less than ActualWindowSize, then return TRUE
(b) else return FALSE.

Example (not normative): if ActualWindowSize is equal to 4, then
InWindow(0, 0) returns TRUE
InWindow(1, 0) returns TRUE
InWindow(3, 0) returns TRUE
InWindow(4, 0) returns FALSE
InWindow(4, 5) returns FALSE (since the modulo 256 difference 4 - 5 = 255)
InWindow(0, 255) returns TRUE (since the modulo 256 difference 0 - 255 = 1)

5.4.3 Function FillWindow

The function "FillWindow" sends PDU segments either until the window is full or until the last segment of a message has been sent. No more than $\mathbf{T}_{seg}$ may be allowed to elapse between the receipt of a SegmentACK APDU and the transmission of a segment. No more than $\mathbf{T}_{seg}$ may be allowed to elapse between the transmission of successive segments of a sequence.

function FillWindow(sequenceNumber)

(a) Set local variable ix to zero.

(b) If the next segment to transmit (the segment numbered sequenceNumber plus ix) is the final segment, goto step (g).

(c) Issue an N-UNITDATA.request with 'data_expecting_reply' = TRUE to transmit the next BACnet APDU segment, with 'segmented-message' = TRUE, 'more-follows' = TRUE, 'proposed-window-size' equal to ProposedWindowSize, and 'sequence-number' = sequenceNumber plus ix, modulo 256.

(d) Set ix equal to ix plus one.

(e) If ix is less than ActualWindowSize, goto step (b).

(f) Goto step (i).

(g) Issue an N-UNITDATA.request with 'data_expecting_reply' = TRUE to transmit the final BACnet APDU segment with 'segmented-message' = TRUE, 'more-follows' = FALSE, 'proposed-window-size' = ProposedWindowSize, and 'sequence-number' = sequenceNumber plus ix, modulo 256.

(h) Set SentAllSegments to TRUE, indicating that all segments have been transmitted at least once.

(i) Return to the caller.

5.4.4 State Machine for Requesting BACnet User (client)

5.4.4.1 IDLE

In the IDLE state, the device waits for the local application program to request a service.

SendUnconfirmed

If UNCONF_SERV.request is received from the local application program,

then issue an N-UNITDATA.request with 'data_expecting_reply' = FALSE to transmit a BACnet-Unconfirmed-Request-PDU, and enter the IDLE state.

SendConfirmedUnsegmented

If CONF_SERV.request is received from the local application program and the length of the APDU is less than or equal to maximum-transmittable-length as determined according to 5.2.1,

then assign an 'invokeID' to this transaction; set SentAllSegments to TRUE; set RetryCount to zero; start RequestTimer; issue an N-UNITDATA.request with 'data_expecting_reply' = TRUE to transmit a BACnet-Confirmed-Request-PDU with 'segmented-message' = FALSE; and enter the AWAIT_CONFIRMATION state to await a reply.

CannotSend

If CONF_SERV.request is received from the local application program and the length of the APDU is greater than maximum-transmittable-length as determined according to 5.2.1 and the Max_Segments_Accepted property of the destination's Device object is known and the total APDU cannot be transmitted without exceeding the maximum number of segments accepted,

then send an ABORT.indication to the local application program and enter the IDLE state.

SendConfirmedSegmented

If CONF_SERV.request is received from the local application program and the length of the APDU is greater than maximum-transmittable-length as determined according to 5.2.1, and the Max_Segments_Accepted property of the destination's Device object is not known, or Max_Segments_Accepted is known and the total APDU can be transmitted without exceeding the maximum number of segments accepted,

then assign an 'invokeID' to this transaction; set SentAllSegments to FALSE; set RetryCount to zero; set SegmentRetryCount to zero; set InitialSequenceNumber to zero; set ProposedWindowSize to whatever value is desired; set ActualWindowSize to 1; start SegmentTimer; issue an N-UNITDATA.request with 'data_expecting_reply' = TRUE to transmit a BACnet-Confirmed-Request-PDU containing the first segment of the message, with 'segmented-message' = TRUE, 'more-follows' = TRUE, 'sequence-number' = zero, and 'proposed-window-size' = ProposedWindowSize; and enter the SEGMENTED_REQUEST state to await an acknowledgment. (The method used to determine ProposedWindowSize is a local matter, except that the value shall be in the range 1 to 127, inclusive.)

UnexpectedSegmentInfoReceived

If an unexpected PDU indicating the existence of an active server TSM (BACnet-ComplexACK-PDU with 'segmented-message' = TRUE or BACnet-SegmentACK-PDU with 'server' = TRUE) is received from the network layer,

then issue an N-UNITDATA.request with 'data_expecting_reply' = FALSE to transmit a BACnet-Abort-PDU with 'server' = FALSE and enter the IDLE state.

UnexpectedPDU_Received

If an unexpected PDU not indicating the existence of an active server TSM (BACnet-SimpleACK-PDU, BACnet-ComplexACK-PDU with 'segmented-message' = FALSE, BACnet-Error-PDU, BACnet-Reject-PDU, or BACnet-Abort-PDU with 'server' = TRUE) is received from the network layer,

then enter the IDLE state. (There is no reason to issue REJECT.indication, ABORT.indication, etc., as the client has no knowledge of the transaction in question.)

5.4.4.2 SEGMENTED_REQUEST

In the SEGMENTED_REQUEST state, the device waits for a BACnet-SegmentACK-PDU for one or more segments of a BACnet-Confirmed-Request-PDU.

DuplicateACK_Received

If a BACnet-SegmentACK-PDU whose 'server' parameter is TRUE is received from the network layer and InWindow ('sequence-number' parameter of the BACnet-SegmentACK-PDU, InitialSequenceNumber) returns a value of FALSE,

then restart SegmentTimer and enter the SEGMENTED_REQUEST state to await an acknowledgment.

NewACK_Received

If a BACnet-SegmentACK-PDU whose 'server' parameter is TRUE is received from the network layer and InWindow ('sequence-number' parameter of the BACnet-SegmentACK-PDU, InitialSequenceNumber) returns a value of TRUE and there is at least one segment remaining to send,

then set InitialSequenceNumber equal to the 'sequence-number' parameter of the BACnet-SegmentACK-PDU plus one, modulo 256; set ActualWindowSize equal to the 'actual-window-size' parameter of the BACnet-SegmentACK-PDU; restart SegmentTimer; set SegmentRetryCount to zero; call FillWindow (InitialSequenceNumber) to transmit one or more BACnet-Confirmed-Request-PDUs containing the next ActualWindowSize segments of the message; and enter the SEGMENTED_REQUEST state to await an acknowledgment.

FinalACK_Received

If a BACnet-SegmentACK-PDU whose 'server' parameter is TRUE is received from the network layer and InWindow ('sequence-number' parameter of the BACnet-SegmentACK-PDU, InitialSequenceNumber) returns a value of TRUE and there are no more segments to send,

then stop SegmentTimer; start RequestTimer; and enter the AWAIT_CONFIRMATION state to await a reply.

Timeout

If SegmentTimer becomes greater than T_{seg} and SegmentRetryCount is less than N_{retry},

then increment SegmentRetryCount; restart SegmentTimer; call FillWindow(InitialSequenceNumber) to retransmit one or more BACnet-Confirmed-Request-PDUs containing the next ActualWindowSize segments of the message; and enter the SEGMENTED_REQUEST state to await an acknowledgment.

FinalTimeout

If SegmentTimer becomes greater than T_{seg} and SegmentRetryCount is greater than or equal to N_{retry},

then stop SegmentTimer; send CONF_SERV.confirm(-) to the local application program; and enter the IDLE state.

AbortPDU_Received

If a BACnet-Abort-PDU whose 'server' parameter is TRUE is received from the network layer,

then stop SegmentTimer; send ABORT.indication to the local application program; and enter the IDLE state.

SimpleACK_Received

If a BACnet-SimpleACK-PDU is received from the network layer and SentAllSegments is TRUE,

then stop SegmentTimer; send CONF_SERV.confirm(+) to the local application program; and enter the IDLE state.

UnsegmentedComplexACK_Received

If a BACnet-ComplexACK-PDU is received from the network layer whose 'segmented-message' parameter is FALSE and SentAllSegments is TRUE,

then stop SegmentTimer; send CONF_SERV.confirm(+) to the local application program; and enter the IDLE state.

SegmentedComplexACK_Received

If a BACnet-ComplexACK-PDU is received from the network layer whose 'segmented-message' parameter is TRUE and whose 'sequence-number' parameter is zero and this device supports segmentation and SentAllSegments is TRUE,

then stop SegmentTimer; compute ActualWindowSize based on the 'proposed-window-size' parameter of the received BACnet-ComplexACK-PDU and on local conditions; issue an N-UNITDATA.request with 'data_expecting_reply' = FALSE to transmit a BACnet-SegmentACK-PDU with 'negative-ACK' = FALSE, 'server' = FALSE, and 'actual-window-size' = ActualWindowSize; start SegmentTimer; set LastSequenceNumber to zero; set InitialSequenceNumber to zero; and enter the SEGMENTED_CONF state to receive the remaining segments. (The method used to determine ActualWindowSize is a local matter, except that the value shall be less than or equal to the 'proposed-window-size' parameter of the received BACnet-ComplexACK-PDU and shall be in the range 1 to 127, inclusive.)

ErrorPDU_Received

If a BACnet-Error-PDU is received from the network layer and SentAllSegments is TRUE,

then stop SegmentTimer; send CONF_SERV.confirm(-) to the local application program; and enter the IDLE state.

RejectPDU_Received

If a BACnet-Reject-PDU is received from the network layer and SentAllSegments is TRUE,

then stop SegmentTimer; send REJECT.indication to the local application program; and enter the IDLE state.

UnexpectedPDU_Received

If a BACnet-SimpleACK-PDU, BACnet-ComplexACK-PDU, BACnet-Error-PDU, or BACnet-Reject-PDU is received from the network layer and SentAllSegments is FALSE,
or if a BACnet-ComplexACK-PDU is received from the network layer whose 'segmented-message' parameter is TRUE and this device does not support segmentation,

or if a BACnet-ComplexACK-PDU is received from the network layer whose 'segmented-message' parameter is TRUE and whose 'sequence-number' parameter is not zero,

then stop SegmentTimer; issue an N-UNITDATA.request with 'data_expecting_reply' = FALSE to transmit a BACnet-Abort-PDU with 'server' = FALSE; send CONF_SERV.confirm(-) to the local application program; and enter the IDLE state.

SendAbort

If ABORT.request is received from the local application program,

then stop SegmentTimer; issue an N-UNITDATA.request with 'data_expecting_reply' = FALSE to transmit a BACnet-Abort-PDU with 'server' = FALSE; and enter the IDLE state.

5.4.4.3 AWAIT_CONFIRMATION

In the AWAIT_CONFIRMATION state, the device waits for a response to a BACnet-Confirmed-Request-PDU.

SimpleACK_Received

If a BACnet-SimpleACK-PDU is received from the network layer,

then stop RequestTimer; send CONF_SERV.confirm(+) to the local application program; and enter the IDLE state.

UnsegmentedComplexACK_Received

If a BACnet-ComplexACK-PDU is received from the network layer whose 'segmented-message' parameter is FALSE,

then stop RequestTimer; send CONF_SERV.confirm(+) to the local application program; and enter the IDLE state.

SegmentedComplexACK_Received

If a BACnet-ComplexACK-PDU is received from the network layer whose 'segmented-message' parameter is TRUE and whose 'sequence-number' parameter is zero and this device supports segmentation,

then stop RequestTimer; compute ActualWindowSize based on the 'proposed-window-size' parameter of the received BACnet-ComplexACK-PDU and on local conditions; issue an N-UNITDATA.request with 'data_expecting_reply' = FALSE to transmit a BACnet-SegmentACK-PDU with 'negative-ACK' = FALSE, 'server' = FALSE, and 'actual-window-size' = ActualWindowSize; start SegmentTimer; set LastSequenceNumber to zero; set InitialSequenceNumber to zero; and enter the SEGMENTED_CONF state to receive the remaining segments. (The method used to determine ActualWindowSize is a local matter, except that the value shall be less than or equal to the 'proposed-window-size' parameter of the received BACnet-ComplexACK-PDU and shall be in the range 1 to 127, inclusive.)

ErrorPDU_Received

If a BACnet-Error-PDU is received from the network layer,

then stop RequestTimer; send CONF_SERV.confirm(-) to the local application program; and enter the IDLE state.

RejectPDU_Received

If a BACnet-Reject-PDU is received from the network layer,

then stop RequestTimer; send REJECT.indication to the local application program; and enter the IDLE state.

AbortPDU_Received

If a BACnet-Abort-PDU whose 'server' parameter is TRUE is received from the network layer,

then stop RequestTimer; send ABORT.indication to the local application program; and enter the IDLE state.

SegmentACK_Received

If a BACnet-SegmentACK-PDU whose 'server' parameter is TRUE is received from the network layer,

then discard the PDU as a duplicate, and re-enter the current state.

UnexpectedPDU_Received

If an unexpected PDU (BACnet-ComplexACK-PDU with 'segmented-message' = TRUE and 'sequence-number' not equal to zero or 'segmented-message' = TRUE and this device does not support segmentation) is received from the network layer,

then stop RequestTimer; issue an N-UNITDATA.request with 'data_expecting_reply' = FALSE to transmit a BACnet-Abort-PDU with 'server' = FALSE; send CONF_SERV.confirm(-) to the local application program; and enter the IDLE state.

TimeoutUnsegmented

If RequestTimer becomes greater than $\mathbf{T}_{out}$ and RetryCount is less than Number_Of_APDU_Retries and the length of the Confirmed Request APDU is less than or equal to maximum-transmittable-length as determined according to 5.2.1,

then stop RequestTimer; increment RetryCount; issue an N-UNITDATA.request with 'data_expecting_reply' = TRUE to transmit a BACnet-Confirmed-Request-PDU with 'segmented-message' = FALSE; start RequestTimer; and enter the AWAIT_CONFIRMATION state to await a reply.

TimeoutSegmented

If RequestTimer becomes greater than $\mathbf{T_{out}}$ and RetryCount is less than Number_Of_APDU_Retries and the length of the Confirmed-Request APDU is greater than maximum-transmittable-length as determined according to 5.2.1,

then stop RequestTimer; increment RetryCount; set SegmentRetryCount to zero; set SentAllSegments to FALSE; start SegmentTimer; set InitialSequenceNumber to zero; set ActualWindowSize to 1; issue an N-UNITDATA.request with 'data_expecting_reply' = TRUE to transmit a BACnet-Confirmed-Request-PDU containing the first segment of the message, with 'segmented-message' = TRUE, 'more-follows' = TRUE, and 'sequence-number' = zero; and enter the SEGMENTED_REQUEST state to await an acknowledgment.

FinalTimeout

If RequestTimer becomes greater than $\mathbf{T_{out}}$ and RetryCount is greater than or equal to Number_Of_APDU_Retries,

then stop RequestTimer; send CONF_SERV.confirm(-) to the local application program; and enter the IDLE state.

SendAbort

If ABORT.request is received from the local application program,

then stop RequestTimer; issue an N-UNITDATA.request with 'data_expecting_reply' = FALSE to transmit a BACnet-Abort-PDU with 'server' = FALSE; and enter the IDLE state.

5.4.4.4 SEGMENTED_CONF

In the SEGMENTED_CONF state, the device waits for one or more segments in response to a BACnet-SegmentACK-PDU.

NewSegmentReceived_NoSpace

If a BACnet-ComplexACK-PDU is received from the network layer whose 'segmented-message' parameter is TRUE; whose 'sequence-number' parameter is equal to LastSequenceNumber plus 1, modulo 256; and the segment cannot be saved due to local conditions,

then stop SegmentTimer; issue an N-UNITDATA.request with 'data_expecting_reply' = FALSE to transmit a BACnet-Abort-PDU with 'server' = FALSE; send CONF_SERV.confirm(-) to the local application program; and enter the IDLE state.

NewSegmentReceived

If a BACnet-ComplexACK-PDU is received from the network layer whose 'segmented-message' parameter is TRUE; whose 'more-follows' parameter is TRUE; whose 'sequence-number' parameter is equal to LastSequenceNumber plus 1, modulo 256; and whose 'sequence-number' parameter is not equal to InitialSequenceNumber plus ActualWindowSize, modulo 256,

then save the BACnet-ComplexACK-PDU segment; increment LastSequenceNumber, modulo 256; restart SegmentTimer; and enter the SEGMENTED_CONF state to receive additional segments.

LastSegmentOfGroupReceived

If a BACnet-ComplexACK-PDU is received from the network layer whose 'segmented-message' parameter is TRUE; whose 'sequence-number' parameter is equal to LastSequenceNumber plus 1, modulo 256; whose 'more-follows' parameter is TRUE; and whose 'sequence-number' parameter is equal to InitialSequenceNumber plus ActualWindowSize, modulo 256,

then save the BACnet-ComplexACK-PDU segment; increment LastSequenceNumber, modulo 256; set InitialSequenceNumber to LastSequenceNumber; issue an N-UNITDATA.request with 'data_expecting_reply' = FALSE to transmit a BACnet-SegmentACK-PDU with 'negative-ACK' = FALSE, server = FALSE, and 'actual-

window-size' = ActualWindowSize; restart SegmentTimer; and enter the SEGMENTED_CONF state to receive additional segments.

LastSegmentOfComplexACK_Received

If a ComplexACK PDU is received from the network layer whose 'segmented-message' parameter is TRUE; whose 'sequence-number' parameter is equal to LastSequenceNumber plus 1, modulo 256; and whose 'more-follows' parameter is FALSE (i.e., the final segment),

then stop SegmentTimer; issue an N-UNITDATA.request with 'data_expecting_reply' = FALSE to transmit a BACnet-SegmentACK-PDU with 'negative-ACK' = FALSE, 'server = FALSE', and 'actual-window-size' = ActualWindowSize; send CONF_SERV.confirm(+) containing all of the received segments to the local application program; and enter the IDLE state.

SegmentReceivedOutOfOrder

If a BACnet-ComplexACK-PDU is received from the network layer whose 'segmented-message' parameter is TRUE and whose 'sequence-number' parameter is not equal to LastSequenceNumber plus 1, modulo 256,

then discard the BACnet-ComplexACK-PDU segment; issue an N-UNITDATA.request with 'data_expecting_reply' = FALSE to transmit a BACnet-SegmentACK-PDU with 'negative-ACK' = TRUE, 'server' = FALSE, 'sequence-number' = LastSequenceNumber, and 'actual-window-size' = ActualWindowSize; restart SegmentTimer; and enter the SEGMENTED_CONF state to receive the remaining segments.

AbortPDU_Received

If a BACnet-Abort-PDU whose 'server' parameter is TRUE is received from the network layer,

then stop SegmentTimer; send ABORT.indication to the local application program; and enter the IDLE state.

UnexpectedPDU_Received

If an unexpected PDU (BACnet-SimpleACK-PDU, BACnet-ComplexACK-PDU with 'segmented-message' = FALSE, BACnet-Error-PDU, BACnet-Reject-PDU, or BACnet-SegmentACK-PDU with 'server' = TRUE) is received from the network layer,

then stop SegmentTimer; issue an N-UNITDATA.request with 'data_expecting_reply' = FALSE to transmit a BACnet-Abort-PDU with 'server' = FALSE; send CONF_SERV.confirm(-) to the local application program; and enter the IDLE state.

Timeout

If SegmentTimer becomes greater than T_{seg} times four,

then stop SegmentTimer; send CONF_SERV.confirm(-) to the local application program; and enter the IDLE state.

SendAbort

If ABORT.request is received from the local application program,

then stop SegmentTimer; issue an N-UNITDATA.request with 'data_expecting_reply' = FALSE to transmit a BACnet-Abort-PDU with 'server' = FALSE; and enter the IDLE state.

5.4.5 State Machine for Responding BACnet User (server)

5.4.5.1 IDLE

In the IDLE state, the device waits for a PDU from the network layer.

UnconfirmedReceived

If a BACnet-Unconfirmed-Request-PDU is received from the network layer,

then send an UNCONF_SERV.indication to the local application program, and enter the IDLE state.

ConfirmedBroadcastReceived
If a BACnet-Confirmed-Request-PDU whose destination address is a multicast or broadcast address is received from the network layer,

then enter the IDLE state.

ConfirmedUnsegmentedReceived
If a BACnet-Confirmed-Request-PDU whose 'segmented-message' parameter is FALSE is received from the network layer,

then send a CONF_SERV.indication to the local application program, start RequestTimer; and enter the AWAIT_RESPONSE state.

ConfirmedSegmentedReceivedNotSupported
If a BACnet-Confirmed-Request-PDU whose 'segmented-message' parameter is TRUE is received from the network layer, and this device does not support the reception of segmented messages,

then issue an N-UNITDATA.request with 'data_expecting_reply' = FALSE to transmit a BACnet-Abort-PDU with 'server' = TRUE and 'abort-reason' = SEGMENTATION_NOT_SUPPORTED, and enter the IDLE state.

ConfirmedSegmentedReceived
If a BACnet-Confirmed-Request-PDU whose 'segmented-message' parameter is TRUE and whose 'sequence-number' parameter is zero is received from the network layer and the local device supports the reception of segmented messages,

then compute ActualWindowSize based on the 'proposed-window-size' parameter of the received BACnet-Confirmed-Request-PDU and on local conditions; issue an N-UNITDATA.request with 'data_expecting_reply' = FALSE to transmit a BACnet-SegmentACK-PDU with 'negative-ACK' = FALSE, 'server' = TRUE, and 'actual-window-size' = ActualWindowSize; start SegmentTimer; set LastSequenceNumber to zero; set InitialSequenceNumber to zero; and enter the SEGMENTED_REQUEST state to receive the remaining segments. (The method used to determine ActualWindowSize is a local matter, except that the value shall be less than or equal to the 'proposed-window-size' parameter of the received BACnet-Confirmed-Request-PDU and shall be in the range 1 to 127, inclusive.)

AbortPDU_Received
If a BACnet-Abort-PDU whose 'server' parameter is FALSE is received from the network layer,

then enter the IDLE state.

UnexpectedPDU_Received
If an unexpected PDU (BACnet-Confirmed-Request-PDU with 'segmented-message' = TRUE and 'sequence-number' not equal to zero or BACnet-SegmentACK-PDU with 'server' = FALSE) is received from the network layer,

then issue an N-UNITDATA.request with 'data_expecting_reply' = FALSE to transmit a BACnet-Abort-PDU with 'server' = TRUE; and enter the IDLE state.

5.4.5.2 SEGMENTED_REQUEST

In the SEGMENTED_REQUEST state, the device waits for segments of a BACnet-Confirmed-Request-PDU.

NewSegmentReceived
If a BACnet-Confirmed-Request-PDU is received from the network layer whose 'segmented-message' parameter is TRUE; whose 'more-follows' parameter is TRUE; whose 'sequence-number' parameter is equal to

LastSequenceNumber plus 1, modulo 256; and whose 'sequence-number' parameter is not equal to InitialSequenceNumber plus ActualWindowSize, modulo 256,

then save the BACnet-Confirmed-Request-PDU segment; increment LastSequenceNumber, modulo 256; restart SegmentTimer; and enter the SEGMENTED_REQUEST state to receive the remaining segments.

LastSegmentOfGroupReceived

If a BACnet-Confirmed-Request-PDU is received from the network layer whose 'segmented-message' parameter is TRUE; whose 'sequence-number' parameter is equal to LastSequenceNumber plus 1, modulo 256; whose 'more-follows' parameter is TRUE; and whose 'sequence-number' parameter is equal to InitialSequenceNumber plus ActualWindowSize, modulo 256,

then save the BACnet-Confirmed-Request-PDU segment; increment LastSequenceNumber, modulo 256; issue an N-UNITDATA.request with 'data_expecting_reply' = FALSE to transmit a BACnet-SegmentACK-PDU with 'negative-ACK' = FALSE, 'server' = TRUE, 'sequence-number' = LastSequenceNumber, and 'actual-window-size' = ActualWindowSize; restart SegmentTimer; set InitialSequenceNumber = LastSequenceNumber; and enter the SEGMENTED_REQUEST state to receive the remaining segments.

LastSegmentOfMessageReceived

If a BACnet-Confirmed-Request-PDU is received from the network layer whose 'segmented-message' parameter is TRUE; whose 'sequence-number' parameter is equal to LastSequenceNumber plus 1, modulo 256; and whose 'more-follows' parameter is FALSE (i.e., the final segment),

then save the BACnet-Confirmed-Request-PDU segment; increment LastSequenceNumber, modulo 256; stop SegmentTimer; issue an N-UNITDATA.request with 'data_expecting_reply' = FALSE to transmit a BACnet-SegmentACK-PDU with 'negative-ACK' = FALSE, 'server' = TRUE, 'sequence-number' = LastSequenceNumber, and 'actual-window-size' = ActualWindowSize; set InitialSequenceNumber = LastSequenceNumber; send CONF_SERV.indication(+) containing all of the received segments to the local application program; start RequestTimer; and enter the AWAIT_RESPONSE state.

SegmentReceivedOutOfOrder

If a BACnet-Confirmed-Request-PDU is received from the network layer whose 'segmented-message' parameter is TRUE and whose 'sequence-number' parameter is not equal to LastSequenceNumber plus 1, modulo 256,

then discard the PDU; issue an N-UNITDATA.request with 'data_expecting_reply' = FALSE to transmit a BACnet-SegmentACK-PDU with 'negative-ACK' = TRUE, 'server' = TRUE, 'sequence-number' = LastSequenceNumber, and 'actual-window-size' = ActualWindowSize; restart SegmentTimer; set InitialSequenceNumber = LastSequenceNumber; and enter the SEGMENTED_REQUEST state to receive the remaining segments.

AbortPDU_Received

If a BACnet-Abort-PDU whose server parameter is FALSE is received from the network layer,

then stop SegmentTimer and enter the IDLE state.

UnexpectedPDU_Received

If an unexpected PDU (BACnet-Confirmed-Request-PDU with 'segmented-message' = FALSE or BACnet-SegmentACK-PDU with 'server' = FALSE) is received from the network layer,

then stop SegmentTimer; issue an N-UNITDATA.request with 'data_expecting_reply' = FALSE to transmit a BACnet-Abort-PDU with 'server' = TRUE; and enter the IDLE state.

Timeout

If SegmentTimer becomes greater than $\mathbf{T}_{seg}$ times four,

then stop SegmentTimer and enter the IDLE state.

SendAbort

If ABORT.request is received from the local application program,

then stop SegmentTimer, issue an N-UNITDATA.request with 'data_expecting_reply' = FALSE to transmit a BACnet-Abort-PDU with 'server' = TRUE, and enter the IDLE state.

5.4.5.3 AWAIT_RESPONSE

In the AWAIT_RESPONSE state, the device waits for the local application program to respond to a BACnet-Confirmed-Request-PDU. See 9.8 for specific considerations in MS/TP networks.

SendSimpleACK

If a CONF_SERV.response(+) is received from the local application program, which is to be conveyed via a BACnet-SimpleACK-PDU,

then issue an N-UNITDATA.request with 'data_expecting_reply' = FALSE to transmit a BACnet-SimpleACK-PDU and enter the IDLE state.

SendUnsegmentedComplexACK

If a CONF_SERV.response(+) is received from the local application program, which is to be conveyed via a BACnet-ComplexACK-PDU, and the length of the APDU is less than or equal to maximum-transmittable-length as determined according to 5.2.1,

then issue an N-UNITDATA.request with 'data_expecting_reply' = FALSE to transmit a BACnet-ComplexACK-PDU with 'segmented-message' = FALSE and enter the IDLE state.

CannotSendSegmentedComplexACK

If a CONF_SERV.response(+) is received from the local application program, which is to be conveyed via a BACnet-ComplexACK-PDU, and the length of the APDU is greater than maximum-transmittable-length as determined according to 5.2.1, and either

(a) this device does not support the transmission of segmented messages or

(b) the client will not accept a segmented response (the 'segmented-response-accepted' parameter in BACnet-ConfirmedRequest-PDU is FALSE), or

(c) the client's max-segments-accepted parameter in the BACnet-ConfirmedRequest-PDU is fewer than required to transmit the total APDU or,

(d) the number of segments transmittable by this device is fewer than required to transmit the total APDU,

then issue an N-UNITDATA.request with 'data_expecting_reply' = FALSE to transmit a BACnet-Abort-PDU with 'server' = TRUE and 'abort-reason' = SEGMENTATION_NOT_SUPPORTED for case (a) and (b), or BUFFER_OVERFLOW for case (c) and (d), and enter the IDLE state.

SendSegmentedComplexACK

If a CONF_SERV.response(+) is received from the local application program that is to be conveyed via a BACnet-ComplexACK-PDU, and the length of the APDU is greater than maximum-transmittable-length as determined according to 5.2.1, and the device supports the transmission of segmented messages, and the client will accept a segmented response ('segmented-response-accepted' parameter in BACnet-ConfirmedRequest-PDU is TRUE),

then set SegmentRetryCount to zero; set InitialSequenceNumber to zero; set ProposedWindowSize to whatever value is desired; set ActualWindowSize to 1; start SegmentTimer; issue an N-UNITDATA.request with 'data_expecting_reply' = TRUE to transmit a BACnet-ComplexACK-PDU containing the first segment of the message, with 'segmented-message' = TRUE, 'more-follows' = TRUE, 'sequence-number' = zero, and 'proposed-window-size' = ProposedWindowSize; and enter the SEGMENTED_RESPONSE state to await an acknowledgment.

SendErrorPDU

If a CONF_SERV.response(-) is received from the local application program,

then issue an N-UNITDATA.request with 'data_expecting_reply' = FALSE to transmit a BACnet-Error-PDU and enter the IDLE state.

SendAbort

If ABORT.request is received from the local application program,

then stop SegmentTimer; issue an N-UNITDATA.request with 'data_expecting_reply' = FALSE to transmit a BACnet-Abort-PDU with 'server' = TRUE; and enter the IDLE state.

SendReject

If REJECT.request is received from the local application program,

then stop SegmentTimer; issue an N-UNITDATA.request with 'data_expecting_reply' = FALSE to transmit a BACnet-Reject-PDU; and enter the IDLE state.

AbortPDU_Received

If a BACnet-Abort-PDU whose 'server' parameter is FALSE is received from the network layer,

then send ABORT.indication to the local application program; and enter the IDLE state.

DuplicateRequestReceived

If a BACnet-Confirmed-Request-PDU whose 'segmented-message' parameter is FALSE is received from the network layer,

then discard the PDU as a duplicate request, and re-enter the current state.

DuplicateSegmentReceived

If a BACnet-Confirmed-Request-PDU whose 'segmented-message' parameter is TRUE is received from the network layer,

then discard the PDU as a duplicate segment; issue an N-UNITDATA.request with 'data_expecting_reply' = FALSE to transmit a BACnet-SegmentACK-PDU with 'negative-ACK' = FALSE, 'server' = TRUE, 'sequence-number' = LastSequenceNumber, and 'actual-window-size' = ActualWindowSize; and re-enter the current state.

UnexpectedPDU_Received

If an unexpected PDU (BACnet-SegmentACK-PDU whose 'server' parameter is FALSE) is received from the network layer,

then issue an N-UNITDATA.request with 'data_expecting_reply' = FALSE to transmit a BACnet-Abort-PDU with 'server' = TRUE; send ABORT.indication to the local application program; and enter the IDLE state.

Timeout

If RequestTimer becomes greater than $\mathbf{T_{out}}$,

then issue an N-UNITDATA.request with 'data_expecting_reply' = FALSE to transmit a BACnet-Abort-PDU with 'server' = TRUE; send ABORT.indication to the local application program; and enter the IDLE state.

5.4.5.4 SEGMENTED_RESPONSE

In the SEGMENTED_RESPONSE state, the device waits for a BACnet-SegmentACK-PDU for a segment or segments of a BACnet-ComplexACK-PDU.

DuplicateACK_Received

If a BACnet-SegmentACK-PDU whose 'server' parameter is FALSE is received from the network layer and InWindow('sequence-number' parameter of the BACnet-SegmentACK-PDU, InitialSequenceNumber) returns a value of FALSE,

then restart SegmentTimer and enter the SEGMENTED_RESPONSE state to await an acknowledgment or timeout.

NewACK_Received

If a BACnet-SegmentACK-PDU whose 'server' parameter is FALSE is received from the network layer and InWindow('sequence-number' parameter of the BACnet-SegmentACK-PDU, InitialSequenceNumber) returns a value of TRUE and there is at least one segment remaining to send,

then set InitialSequenceNumber equal to the 'sequence-number' parameter of the BACnet-SegmentACK-PDU plus one, modulo 256; set ActualWindowSize equal to the 'actual-window-size' parameter of the BACnet-SegmentACK-PDU; restart SegmentTimer; set SegmentRetryCount to zero; call FillWindow(InitialSequenceNumber) to issue an N-UNITDATA.request with 'data_expecting_reply' = TRUE to transmit one or more BACnet-ComplexACK-PDUs containing the next ActualWindowSize segments of the message; and enter the SEGMENTED_RESPONSE state to await an acknowledgment.

FinalACK_Received

If a BACnet-SegmentACK-PDU whose 'server' parameter is FALSE is received from the network layer and InWindow('sequence-number' parameter of the BACnet-SegmentACK-PDU, InitialSequenceNumber) returns a value of TRUE and there are no more segments to send,

then stop SegmentTimer and enter the IDLE state.

Timeout

If SegmentTimer becomes greater than $\mathbf{T}_{seg}$ and SegmentRetryCount is less than Number_Of_APDU_Retries,

then increment SegmentRetryCount; restart SegmentTimer; call FillWindow(InitialSequenceNumber) to reissue an N-UNITDATA.request with 'data_expecting_reply' = TRUE to transmit one or more BACnet-ComplexACK-PDUs containing the next ActualWindowSize segments of the message; and enter the SEGMENTED_RESPONSE state to await an acknowledgment.

FinalTimeout

If SegmentTimer becomes greater than $\mathbf{T}_{seg}$ and SegmentRetryCount is greater than or equal to Number_Of_APDU_Retries,

then stop the SegmentTimer, and enter the IDLE state.

AbortPDU_Received

If a BACnet-Abort-PDU whose 'server' parameter is FALSE is received from the network layer,

then stop SegmentTimer; send ABORT.indication to the local application program; and enter the IDLE state.

UnexpectedPDU_Received

If an unexpected PDU (BACnet-Confirmed-Request-PDU) is received from the network layer,

then stop SegmentTimer; issue an N-UNITDATA.request with 'data_expecting_reply' = FALSE to transmit a BACnet-Abort-PDU with 'server' = TRUE; and enter the IDLE state.

SendAbort

If ABORT.request is received from the local application program,

then stop SegmentTimer; issue an N-UNITDATA.request with 'data_expecting_reply' = FALSE to transmit a BACnet-Abort-PDU with 'server' = TRUE; and enter the IDLE state.

5.5 Application Protocol Time Sequence Diagrams

The flow sequence of service primitives can be represented by time-sequence diagrams. Each diagram is partitioned into three or four fields. The field labeled "Provider" represents the service-provider and the two fields labeled "User" represent the two service-users. The fourth field, if present, represents an application program. For the application layer, the vertical lines between user and provider represent the interface between the BACnet User Element and the BACnet ASE. For lower layers these vertical lines represent the service-access-points between the service-users and the service-provider. Moving from top to bottom in the diagram represents the passage of time. Arrows, placed in the areas representing the service-user, indicate the main flow of information during the execution of an interaction described by a service-primitive (i.e., to or from the service-user). Figures 5-4 through 5-13 illustrate the various sequences of application service primitives defined in BACnet.

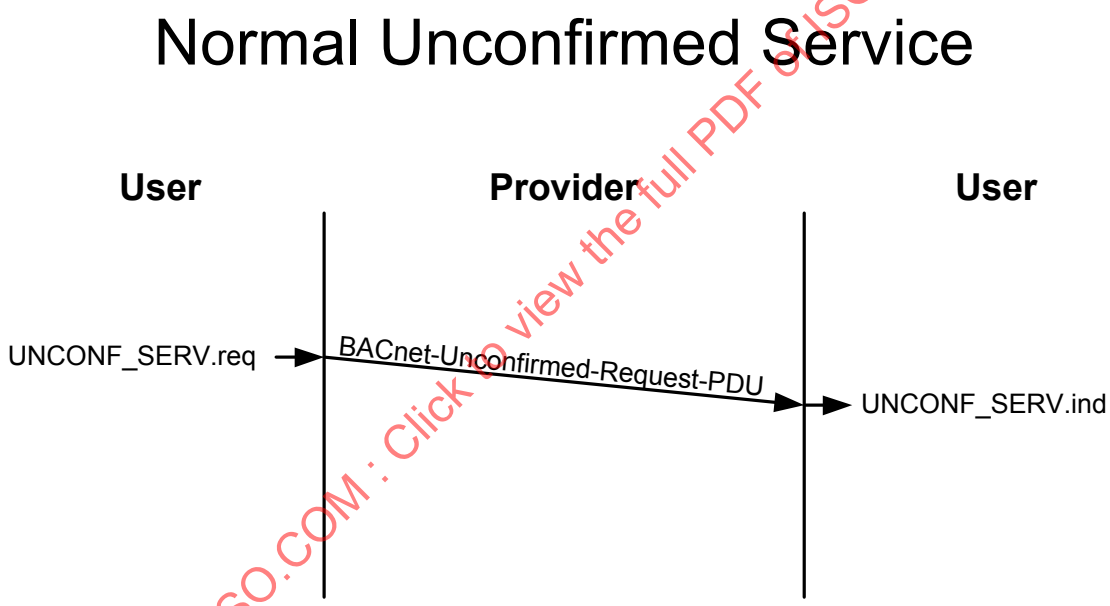


Figure 5-4. Time sequence diagram for a normal unconfirmed service.

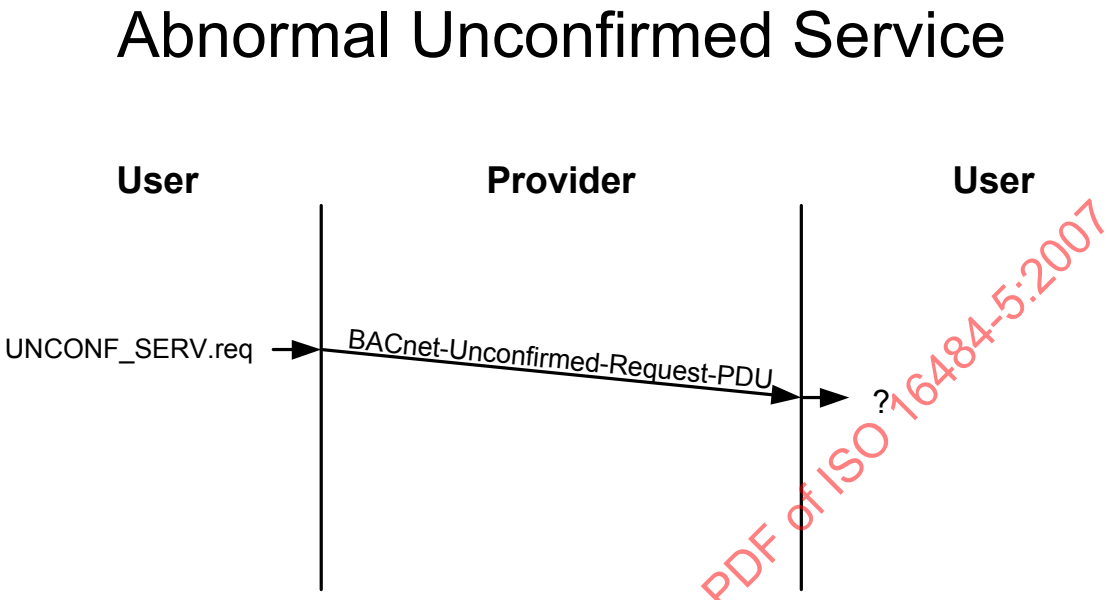


Figure 5-5. Time sequence diagram for an abnormal unconfirmed service. Unconfirmed service requests that are in some way flawed are ignored by the receiving user as indicated by the symbol "?".

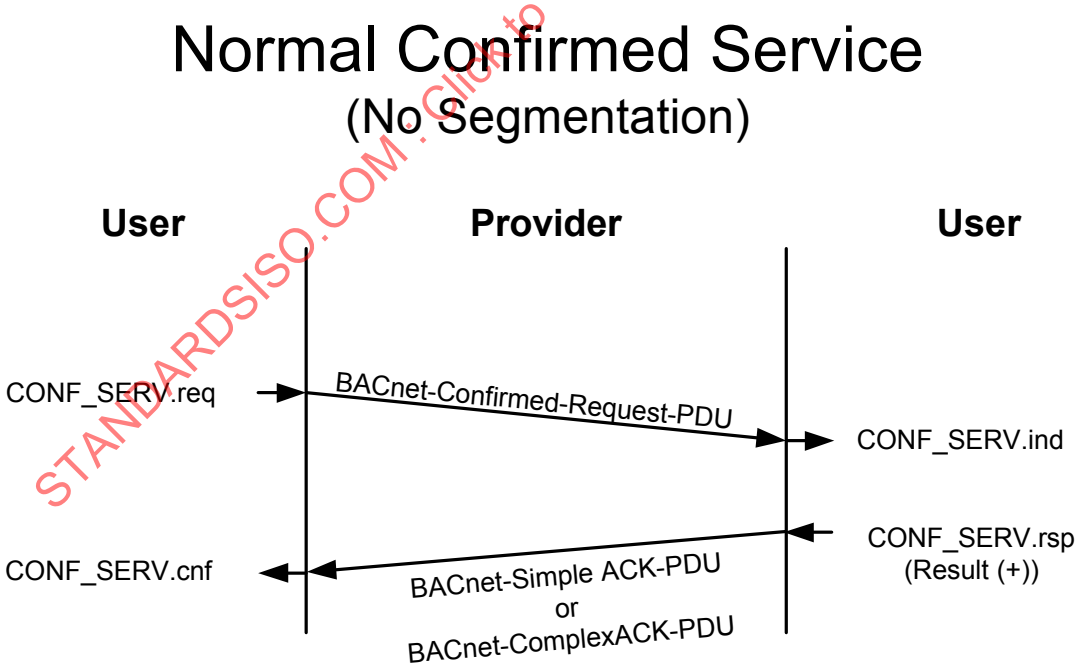


Figure 5-6. Time sequence diagram for normal confirmed services.

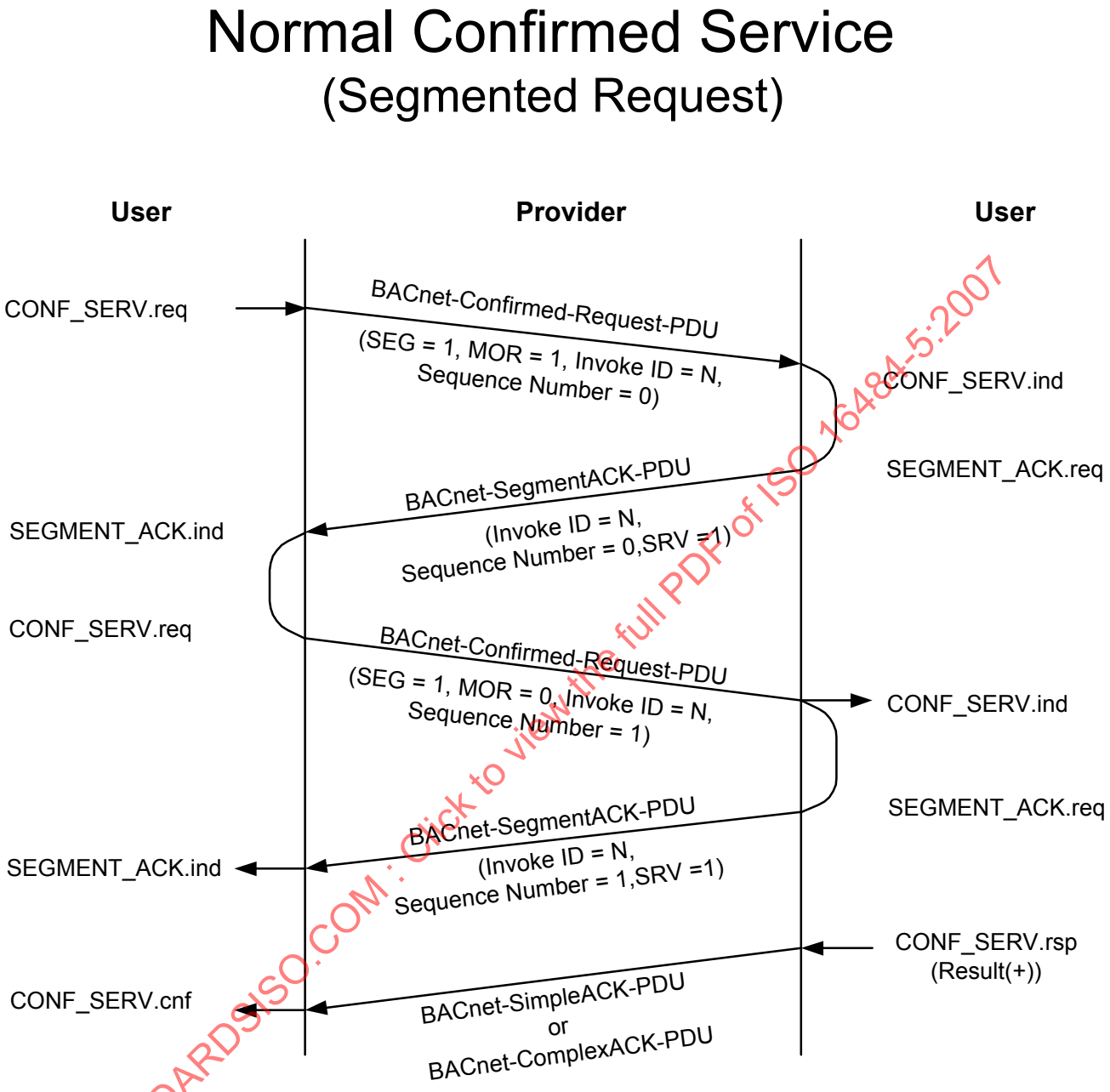


Figure 5-7. Time sequence diagram for a normal confirmed service with a segmented request.

Figure 5-7 illustrates two separate, interleaved exchanges of service primitives. One exchange is the usual confirmed service request, indication, response, and confirm sequence. Because the request is segmented it takes several CONF_SERV.request primitives to convey the entire request. The segment acknowledge service primitives, which are an independent exchange, are used to signal the client that the server is ready for the next segment.

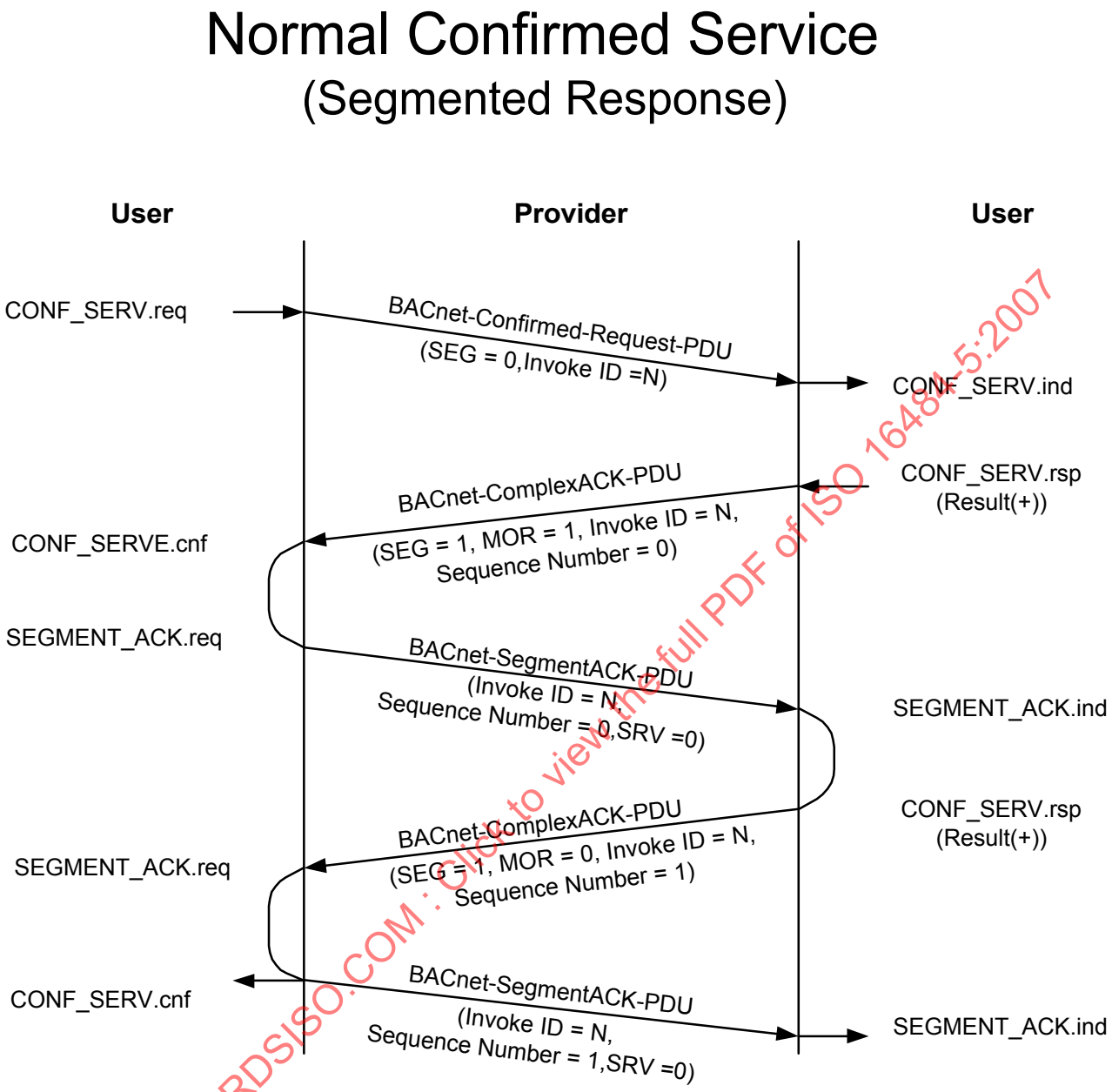


Figure 5-8. Time sequence diagram for a normal confirmed service with segmented response.

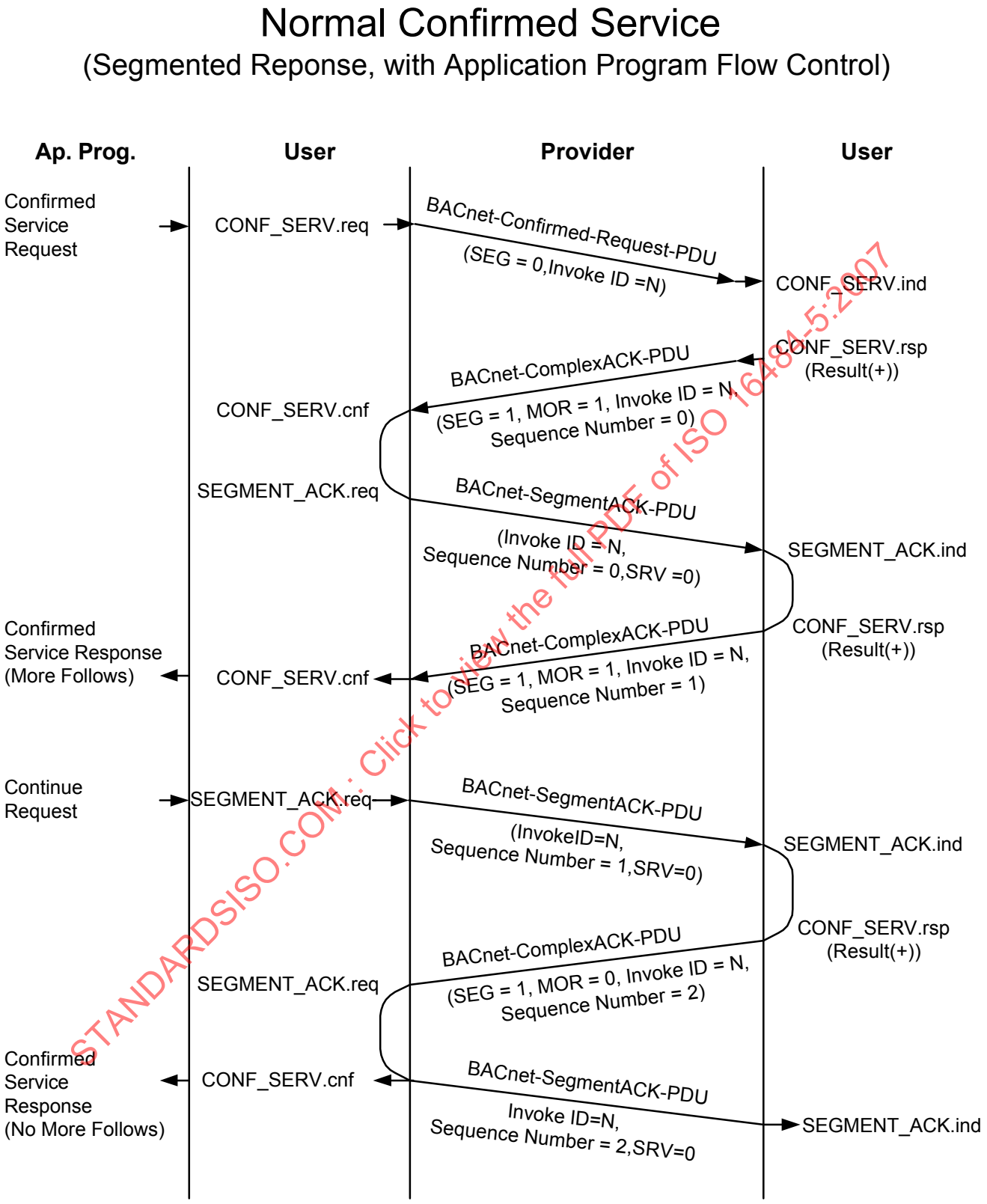


Figure 5-9. Time sequence diagram for a normal confirmed service with application flow control.

Normal Confirmed Service

(Segmented Response, with Application Program Flow Control and Requester Abort)

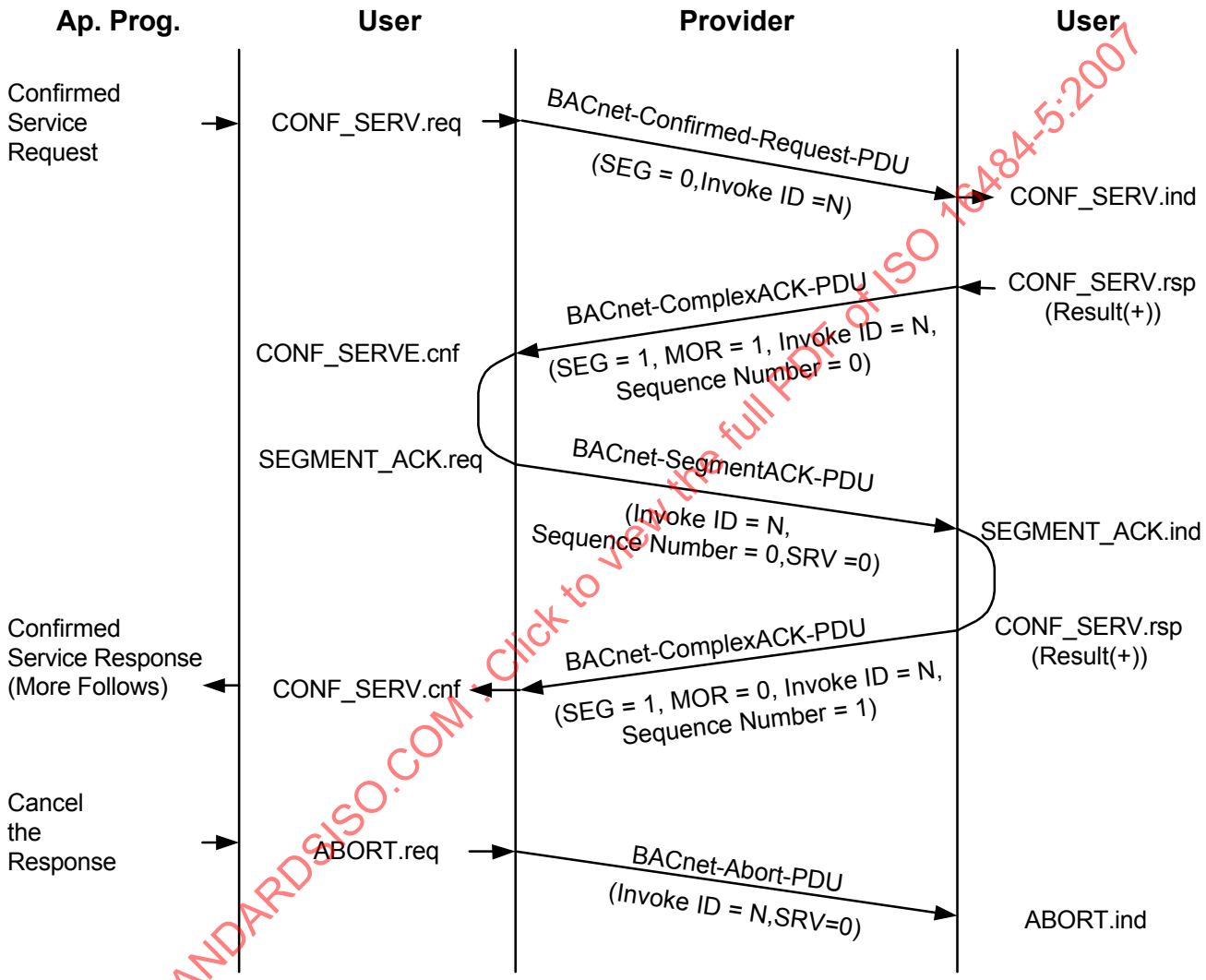


Figure 5-10. Time sequence diagram for a normal confirmed service with segmented response, application program flow control, and response cancellation.

Abnormal Confirmed Service

(Segmented Response and Requester Abort)

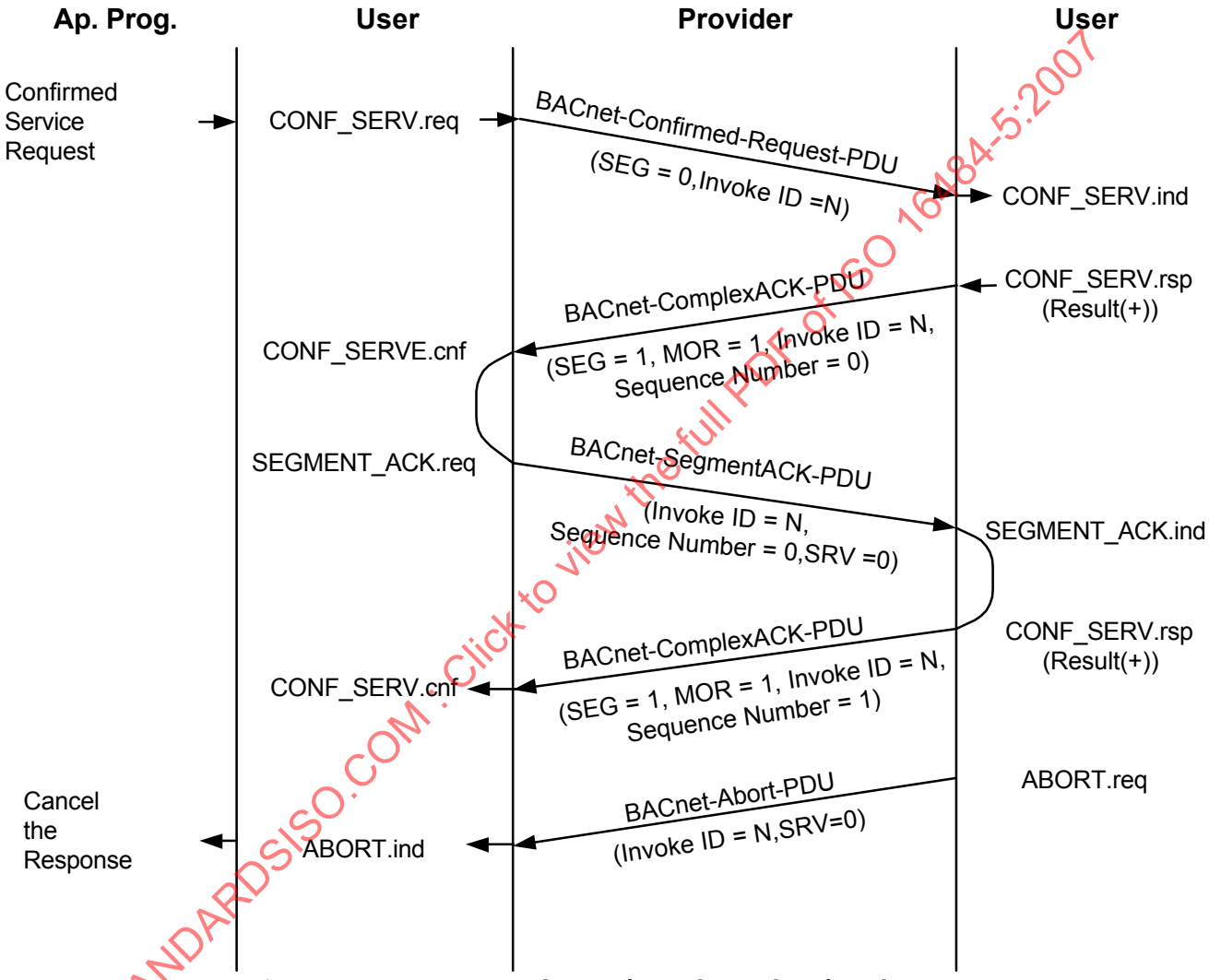


Figure 5-11. Time sequence diagram for an abnormal confirmed service.

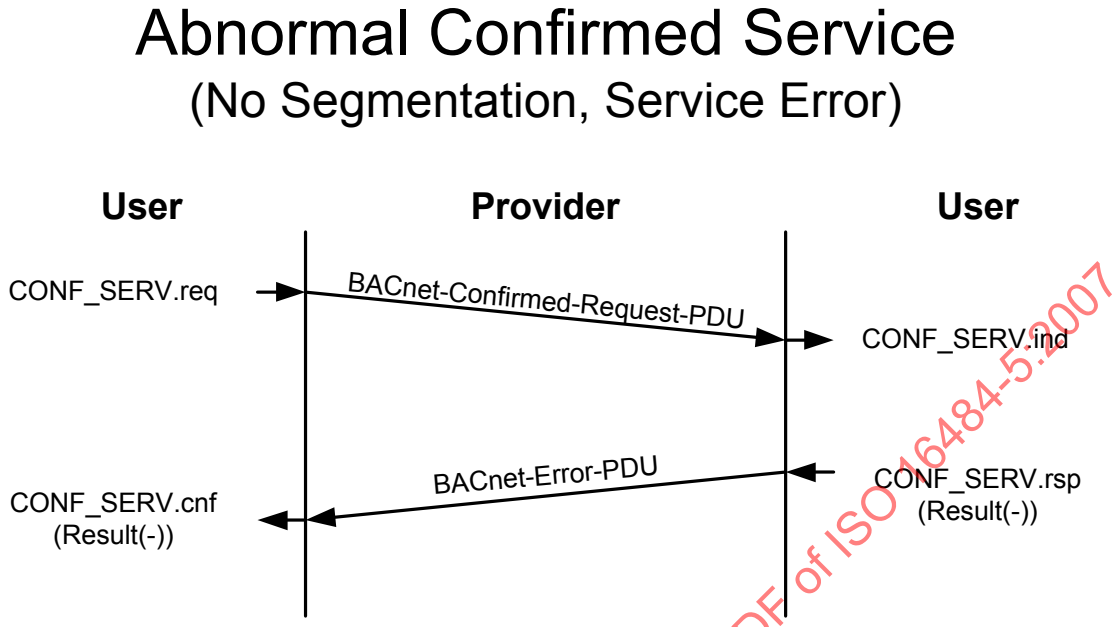


Figure 5-12. Time sequence diagram for an abnormal confirmed service.

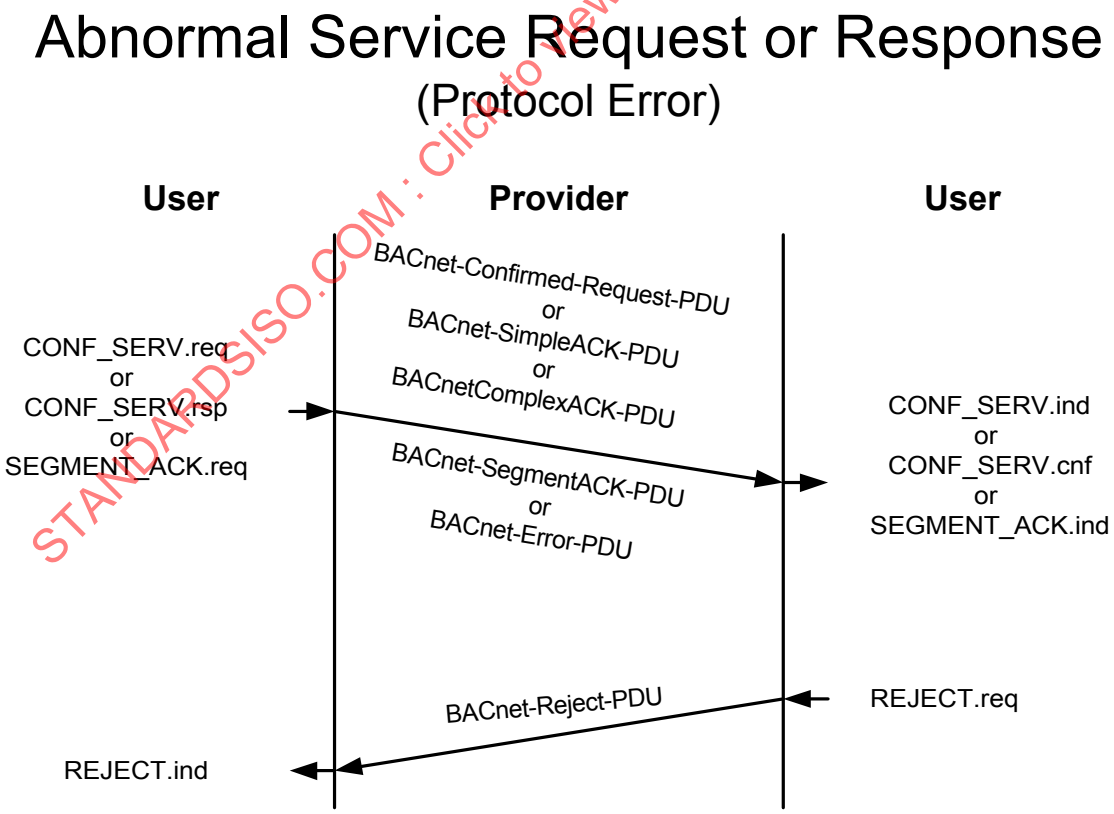


Figure 5-13. Time sequence diagram for an abnormal service request or response with a protocol error.

5.6 Application Layer Service Conventions

This standard uses the descriptive conventions contained in the OSI technical report on service conventions, ISO TR 8509. The OSI conventions define the interactions between a protocol service user and a protocol service provider. Information passed between the protocol service user and the protocol service provider is represented abstractly as an exchange of "service primitives." The service primitives are an abstraction of the functional specification and the user-layer interaction. The abstract definition does not contain local detail of the user/provider interaction. Each primitive has a set of zero or more parameters, representing data elements that are passed to qualify the functions invoked by the primitive. Parameters indicate information available in a user/provider interaction; in any particular interface, some parameters may be explicitly stated (even though not explicitly defined in the primitive) or implicitly associated with the service access point. Similarly, in any particular protocol specification, functions corresponding to a service primitive may be explicitly defined or implicitly available.

Clauses 13 through 17 and 24 use a tabular format to describe the component parameters of the BACnet service primitives. Each table consists of five columns, containing the name of the service parameter and a column each for the request ("Req"), indication ("Ind"), response ("Rsp"), and confirm ("Cnf") primitives. The "Rsp" and "Cnf" columns are absent for unconfirmed services. Each row of the table contains one parameter or subparameter. Under the appropriate service primitive columns, a code is used to specify the type of use of the parameter on the primitive specified in the vertical column. These codes follow the conventions suggested in the ISO technical report on conventions, ISO TR 8509, namely:

M - parameter is Mandatory for the primitive.
U - parameter is a User option and may not be provided.
C - parameter is Conditional upon other parameters.
S - parameter is a Selection from a collection of two or more possible parameters. The parameters that make up this collection are indicated in the table as follows:

(a) each parameter in the collection is specified with the code "S";

(b) the name of each parameter in the collection is at the same table indentation from the beginning of the parameter column in the table; and

(c) either

1. each parameter is at the leftmost (outer) indentation in the table or

2. each parameter is part of the same parameter group. A parameter group is a collection of parameters where each member has a common parent parameter. The parent parameter for any group member is the first parameter above the member that is not indented as far as that member. In the following example, ParameterA and ParameterB form a parameter group:

```
ParameterX
        ParameterA
        ParameterB
ParameterY
        ParameterC
```

Informally, for parameters involved in a selection, the indentation in the service tables signifies which parameters are involved in a selection. All parameters at the same level of indentation under a common "higher level" parameter are part of the same selection.

The code "(=)" following one of the codes M, U, C, or S indicates that the parameter is semantically equivalent to the parameter in the service primitive to its immediate left in the table. For instance, an "M(=)" code in the indication service primitive column and "M" in the request service primitive column means that the parameter in the indication primitive is semantically equivalent to that in the request primitive.

Some parameters may contain subparameters. Subparameters are indicated by indenting them with respect to the parent parameter. The presence of subparameters is always dependent on the presence of the parent parameter. In the example above, ParameterA and ParameterB are subparameters of ParameterX and ParameterC is a subparameter of ParameterY. If ParameterX is optional and is not supplied in a service primitive, then the subparameters (ParameterA and ParameterB) shall not be supplied.

Some service parameters are named using a "List of ..." convention. Unless otherwise noted, all parameters whose name begins with "List of ..." specify a list of zero or more of the item specified after the "List of" keyword phrase.

6 THE NETWORK LAYER

The purpose of the BACnet network layer is to provide the means by which messages can be relayed from one BACnet network to another, regardless of the BACnet data link technology in use on that network. Whereas the data link layer provides the capability to address messages to a single device or broadcast them to all devices on the local network, the network layer allows messages to be directed to a single remote device, broadcast on a remote network, or broadcast globally to all devices on all networks. A BACnet Device is uniquely located by a network number and a MAC address.

Devices that interconnect two disparate BACnet LANs, e.g., ISO 8802-3 and ARCNET, and provide the relay function described in this clause are called "BACnet routers." Devices that interconnect two disparate BACnet networks through a point-to-point (PTP) connection (see Clause 10) are also BACnet routers. BACnet routers build and maintain their routing tables automatically using the network layer protocol messages defined in this clause. Network layer protocol messages facilitate both the auto-configuration of routers and the flow of messages to, and between, routers. BACnet routing capability may be implemented in stand-alone devices or, alternatively, in devices that carry out other building automation and control functions.

Some functions assigned to the network layer by the OSI Basic Reference Model are not required in BACnet. One such function involves selecting a communications path between source and destination machines based on an optimization algorithm. This is not required because BACnet internetworks shall be designed and installed with at most a single, active path between any two devices, a constraint that greatly reduces the complexity of the network layer. Another common network layer function is message segmentation and reassembly. To obviate the need for these capabilities at the network layer, BACnet imposes a limitation on the length of the NPDU in messages passed through a BACnet router. The maximum NPDU length shall not exceed the capability of any data link technology encountered along the path from source to destination. A list of the maximum NPDU lengths for BACnet data link technologies is given in Table 6-1.

Table 6-1. Maximum NPDU Lengths When Routing Through Different BACnet Data Link Layers

Data Link Technology	Maximum NPDU Length
ISO 8802-3 ("Ethernet"), as defined in Clause 7	1497 octets
ARCNET, as defined in Clause 8	501 octets
MS/TP, as defined in Clause 9	501 octets
Point-To-Point, as defined in Clause 10	501 octets
LonTalk, as defined in Clause 11	228 octets
BACnet/IP, as defined in Annex J	1497 octets

6.1 Network Layer Service Specification

Conceptually, the BACnet network layer provides an unacknowledged connectionless form of data unit transfer service to the application layer. The primitives associated with the interaction are the N-UNITDATA request and indication. These primitives provide parameters as follows:

N-UNITDATA.request (
destination_address,
data,
network_priority,
data_expecting_reply
)

N-UNITDATA.indication (
source_address,
destination_address,
data,
network_priority,
data_expecting_reply
)

The 'destination_address' and 'source_address' parameters provide the logical concatenation of 1) an optional network number, 2) the MAC address appropriate to the underlying LAN technology, and the 3) the link service access point. A network number of X'FFFF' indicates that the message is to be broadcast "globally" to all devices on all currently reachable networks. Currently reachable networks are those networks to which an active connection is already established within the BACnet internet. In particular, a global broadcast shall not trigger any attempts to establish PTP connections. The 'data' parameter is the network service data unit (NSDU) passed down from the application layer and is composed of a fully encoded BACnet APDU. The 'network_priority' is a numeric value used by the network layer in BACnet routers to determine any possible deviations from a first-in-first-out approach to managing the queue of messages awaiting relay. The data_expecting_reply parameter indicates whether (TRUE) or not (FALSE) a reply data unit is expected for the data unit being transferred.

Upon receipt of an N-UNITDATA.request primitive from the application layer, the network layer shall attempt to send an NSDU using the procedures described in this clause. Upon receipt of an NSDU from a peer network entity, a network entity shall either 1) send the NSDU to its destination on a directly connected network, 2) send the NSDU to the next BACnet router en route to its destination, and/or 3) if the destination address matches that of one of its own application entities, issue an N-UNITDATA.indication primitive to the appropriate entity in its own application layer to signal the arrival of the NSDU.

6.2 Network Layer PDU Structure

6.2.1 Protocol Version Number

Each NPDU shall begin with a single octet that indicates the version number of the BACnet protocol, encoded as an 8 bit unsigned integer. The present version number of the BACnet protocol is one (1).

6.2.2 Network Layer Protocol Control Information

The second octet in an NPDU shall be a control octet that indicates the presence or absence of particular NPCI fields. Figure 6-1 shows the order of the NPCI fields in an encoded NPDU. Use of the bits in the control octet is as follows.

Bit 7: 1 indicates that the NSDU conveys a network layer message. Message Type field is present.
0 indicates that the NSDU contains a BACnet APDU. Message Type field is absent.

Bit 6: Reserved. Shall be zero.

Bit 5: Destination specifier where:
0 = DNET, DLEN, DADR, and Hop Count absent
1 = DNET, DLEN, and Hop Count present
DLEN = 0 denotes broadcast MAC DADR and DADR field is absent
DLEN > 0 specifies length of DADR field

Bit 4: Reserved. Shall be zero.

Bit 3: Source specifier where:
0 = SNET, SLEN, and SADR absent
1 = SNET, SLEN, and SADR present
SLEN = 0 Invalid
SLEN > 0 specifies length of SADR field

Bit 2: The value of this bit corresponds to the data_expecting_reply parameter in the N-UNITDATA primitives.

1 indicates that a BACnet-Confirmed-Request-PDU, a segment of a BACnet-ComplexACK-PDU, or a network layer message expecting a reply is present.

0 indicates that other than a BACnet-Confirmed-Request-PDU, a segment of a BACnet-ComplexACK-PDU, or a network layer message expecting a reply is present.

Bits 1,0: Network priority where:
B'11' = Life Safety message
B'10' = Critical Equipment message
B'01' = Urgent message
B '00' = Normal message

In this standard:

DNET = 2-octet ultimate destination network number.
DLEN = 1-octet length of ultimate destination MAC layer address (A value of 0 indicates a broadcast on the destination network.)
DADR = Ultimate destination MAC layer address.
DA = Local network destination MAC layer address.
SNET = 2-octet original source network number.
SLEN = 1-octet length of original source MAC layer address.
SADR = Original source MAC layer address.
SA = Local network source MAC layer address.

Version	1 octet
Control	1 octet
DNET	2 octets
DLEN	1 octet
DADR	variable
SNET	2 octets
SLEN	1 octet
SADR	variable
Hop Count	1 octet
Message Type	1 octet
Vendor ID	2 octets
APDU	N octets

Figure 6-1. NPDU field format. Which fields are present is determined by the bits in the control octet.

Figures 6-2(a) - 6-2(e) provide examples of NPDUs containing APDUs for various combinations of addressing information.

Version = X'01'	1 octet
Control = X'04'	1 octet
APDU	N octets

Figure 6-2(a). Example of a typical "Local" BACnet NPDU for which a reply is expected.

Version = X'01'	1 octet
Control = X'24'	1 octet
DNET	2 octets
DLEN = M	1 octet
DADR	M octets
Hop Count	1 octet
APDU	N octets

Figure 6-2(b). Example of a typical "Remote" BACnet NPDU directed to a router. Network Priority is NORMAL and a reply is expected.

Version = X'01'	1 octet
Control = X'29'	1 octet
DNET	2 octets
DLEN	1 octet
DADR	1 octet
SNET	2 octets
SLEN = 6	1 octet
SADR	6 octets
Hop Count	1 octet
APDU	N octets

Figure 6-2(c). Example of a typical BACnet NPDU as passed between routers. Network Priority is URGENT, original MAC Address is 6 octets, and the ultimate Destination MAC Address is 1 octet.

Version =X'01'	1 octet
Control = X'28'	1 octet
DNET = X'FFFF'	2 octets
DLEN = 0	1 octet
SNET	2 octets
SLEN	1 octet
SADR	1 octet
Hop Count	1 octet
APDU	N octets

Figure 6-2(e). Example of a typical Broadcast message of NORMAL Network Priority as broadcast by a Router.

Version = X'01'	1 octet
Control = X'0B'	1 octet
SNET	2 octets
SLEN = 6	1 octet
SADR	6 octets
APDU	N octets

Figure 6-2(d). Example of a typical "Remote" BACnet NPDU as sent from a Router to its ultimate destination on a directly connected network. Network Priority is LIFE SAFETY.

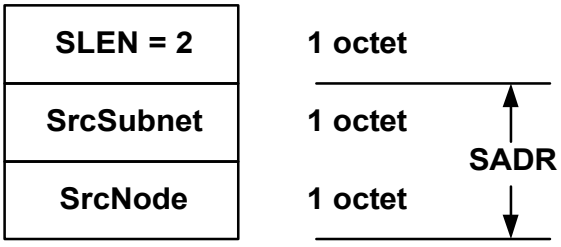


Figure 6-3. Encoding of the SLEN and SADR for NPDUs destined for LonTalk devices being routed through BACnet.

6.2.2.1 DNET, SNET, and Vendor ID Encoding

The multi-octet fields, DNET, SNET, and Vendor ID, shall be conveyed with the most significant octet first. Allowable network number values for DNET shall be from 1 to 65535 and for SNET from 1 to 65534.

6.2.2.2 DADR and SADR Encoding

The DADR and SADR fields are encoded as shown in Table 6-2, Figure 6-3, and Figure 6-4.

Table 6-2. BACnet DADR and SADR encoding rules based upon data link layer technology

BACnet Data Link Layer	DLEN	SLEN	Encoding Rules
ISO 8802-3 ("Ethernet"), as defined in Clause 7	6	6	Encoded as in their MAC layer representations
ARCNET, as defined in Clause 8	1	1	Encoded as in their MAC layer representations
MS/TP, as defined in Clause 9	1	1	Encoded as in their MAC layer representations
LonTalk domain wide broadcast	2	2	The encoding for the SADR is shown in Figure 6-3
LonTalk multicast	2	2	
LonTalk unicast	2	2	The encoding for the DADR is shown in Figure 6-4
LonTalk, unique Neuron_ID	7	2	

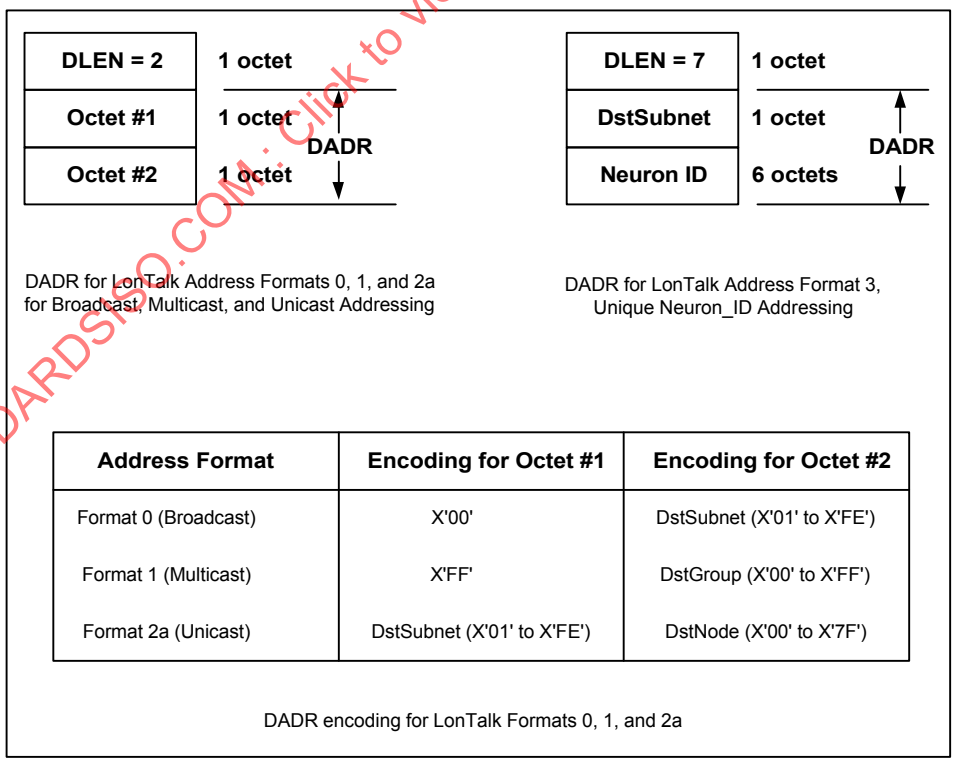


Figure 6-4. Encoding of the DLEN and DADR for NPDUs destined for LonTalk devices being routed through BACnet. The different LonTalk address formats are encoded as shown.

6.2.3 Hop Count

The Hop Count is a decrementing counter value used to ensure that a message cannot be routed in a circular path indefinitely. Such a circular path can only occur if the configuration rule that allows only a single path between any two BACnet nodes is violated. See 4.2.

The Hop Count field shall be present only if the message is destined for a remote network, i.e., if DNET is present. This is a one-octet field that is initialized to a value of X'FF'. Each router the message passes through shall decrement the Hop Count by at least one. If the Hop Count reaches a value of zero, the router shall discard the message and not forward it to the next router.

6.2.4 Network Layer Message Type

If Bit 7 of the control octet described in 6.2.2 is 1, a message type octet shall be present as shown in Figure 6-1. The following message types are indicated:

X'00': Who-Is-Router-To-Network
X'01': I-Am-Router-To-Network
X'02': I-Could-Be-Router-To-Network
X'03': Reject-Message-To-Network
X'04': Router-Busy-To-Network
X'05': Router-Available-To-Network
X'06': Initialize-Routing-Table
X'07': Initialize-Routing-Table-Ack
X'08': Establish-Connection-To-Network
X'09': Disconnect-Connection-To-Network
X'0A' to X'7F': Reserved for use by ASHRAE
X'80' to X'FF': Available for vendor proprietary messages

Figures 6-5 through 6-10 provide examples of NPDUs containing network layer messages.

6.2.5 Vendor Proprietary Network Layer Messages

If Bit 7 of the control octet is 1 and the Message Type field contains a value in the range X'80' – X'FF', then a Vendor ID field shall be present as shown in Figure 6-1. Otherwise, the Vendor ID shall be omitted. The Vendor ID is defined in Clause 23. The Vendor ID shall be encoded in two octets.

Version = X'01'	1 octet
Control = X'80'	1 octet
Message Type = X'00'	1 octet
DNET	2 octets (optional)

Figure 6-5. Example of a Who-Is-Router-To-Network message. If DNET is omitted, a router receiving this message shall return a list of all reachable DNETs.

Version = X'01'	1 octet
Control = X'80'	1 octet
Message Type = X'08'	1 octet
DNET	2 octets
Termination Time Value	1 octet

Figure 6-6. Example of an Establish-Connection-To-Network message directed to a local router.

Version = X'01'	1 octet
NPCI = X'80'	1 octet
Message Type = X'03'	1 octet
Rejection Reason	1 octet
DNET	2 octets

Figure 6-7. Example of a Reject-Message-To-Network DNET.

Version = X'01'	1 octet
Control = X'80'	1 octet
Message Type = X'05'	1 octet
List of DNETs	2N octets (optional)

Figure 6-8. Example of a Router-Available-To-Network for one or more DNETs. If the list of DNETs is not present, flow control is being eased on all networks normally reached through the router.

Version = X'01'	1 octet
Control = X'80'	1 octet
Message Type = X'04'	1 octet
List of DNETs	2N octets (optional)

Figure 6-9. Example of a Router-Busy-To-Network for one or more DNETs. If the list of DNETs is not present, flow control is being imposed on all networks normally reachable through the router.

Version = X'01'	1 octet
Control = X'80'	1 octet
Message Type = X'02'	1 octet
DNET	2 octets
Performance Index	1 octet

Figure 6-10. Example of an I-Could-Be-Router-To-Network Message.

6.2.6 Network Layer Messages Conveying Data

If there are data octets to be conveyed for the message type specified in 6.2.4, these data octets shall follow the message type octet in the manner prescribed for each message type.

6.3 Messages for Multiple Recipients

BACnet supports the transmission of messages to multiple recipients through the use of multicast and broadcast addresses. Multicasting results in a message being processed by a group of recipients. Broadcasting results in a message being processed by all of the BACnet Devices on the local network, a remote network, or all networks. The use of broadcast or multicast addressing for network layer protocol messages is described in 6.5. Of the BACnet APDUs, only the BACnet-Unconfirmed-Request-PDU may be transmitted using a multicast or broadcast address.

6.3.1 Multicast Messages

At present only ISO 8802-3 and LonTalk support multicast addresses. The method by which a BACnet Device is assigned to a specific multicast group shall be a local matter.

6.3.2 Broadcast Messages

Three forms of broadcast transmission are provided by BACnet: local, remote, and global. A local broadcast is received by all stations on the local network. A remote broadcast is received by all stations on a single remote network. A global broadcast is received by all stations on all networks comprising the BACnet internetwork.

A local broadcast makes use of the broadcast MAC address appropriate to the local network's LAN technology, i.e., X'FFFFFFFFFFFF' for ISO 8802-3, X'00' for ARCNET, X'FF' for MS/TP, or X'00' in the DstSubnet field of Address Format 0 in LonTalk.

A remote broadcast is made on behalf of the source device on a specific distant network by a router directly connected to that network. In this case, DNET shall specify the network number of the remote network and DLEN shall be set to zero.

A global broadcast, indicated by a DNET of X'FFFF', is sent to all networks through all routers. Upon receipt of a message with the global broadcast DNET network number, a router shall decrement the Hop Count. If the Hop Count is still greater than zero, then the router shall broadcast the message on all directly connected networks except the network of origin, using the broadcast MAC address appropriate for each destination network. If the Hop Count is zero, then the router shall discard the message. In order for the message to be disseminated globally, the originating device shall use a broadcast MAC address on the originating network so that all attached routers may receive the message and propagate it further.

If a router has one or more ports that represent PTP connections as defined in Clause 10, global broadcasts shall be processed as follows. If the PTP connection is currently established, that is, the Connection State Machine is in the Connected state (see 10.4.9), then the global broadcast message shall be transmitted through the PTP connection. If the PTP connection is not currently established, then no action shall be taken by the router to transmit the broadcast message through the PTP connection.

6.4 Network Layer Protocol Messages

This subclause describes the format and purpose of the ten BACnet network layer protocol messages. These messages provide the basis for router auto-configuration, router table maintenance, and network layer congestion control.

6.4.1 Who-Is-Router-To-Network

This message is indicated by a Message Type of X'00' optionally followed by a 2-octet network number. Who-Is-Router-To-Network is used by both routing and non-routing nodes to ascertain the next router to a specific destination network or, in the case of routers, as an aid in building an up-to-date routing table. See Figure 6-5.

6.4.2 I-Am-Router-To-Network

This message is indicated by a Message Type of X'01' followed by one or more 2-octet network numbers. It is used to indicate the network numbers of the networks accessible through the router generating the message. It shall always be transmitted with a broadcast MAC address.

6.4.3 I-Could-Be-Router-To-Network

This message is used to respond to a Who-Is-Router-To-Network message containing a specific 2-octet network number when the responding half-router has the capability of establishing a PTP connection that can be used to reach the desired network but this PTP connection is not currently established.

This message is indicated by a Message Type of X'02'. The complete format of the NPDU is shown in Figure 6-10. The 2-octet network number indicates the DNET that could be reached by this half-router. The 1-octet "Performance Index" is a locally determined number that gives an indication of the quality and performance of this proposed connection. A low value in this field indicates a high performance index. Typically, the Performance Index would be established at installation time and set relative to the performance of other PTP half-routers in the system.

6.4.4 Reject-Message-To-Network

This message is indicated by a Message Type of X'03' followed by an octet indicating the reason for the rejection and a 2-

octet network number (see Figure 6 -7). It is directed to the node that originated the message being rejected, as indicated by the source address information in that message. The rejection reason octet shall contain an unsigned integer with one of the following values:

0: Other error.

1: The router is not directly connected to DNET and cannot find a router to DNET on any directly connected network using Who-Is-Router-To-Network messages.

2: The router is busy and unable to accept messages for the specified DNET at the present time.

3: It is an unknown network layer message type.

4: The message is too long to be routed to this DNET.

6.4.5 Router-Busy-To-Network

This message is indicated by a Message Type of X'04' optionally followed by a list of 2-octet network numbers. It shall always be transmitted with a broadcast MAC address appropriate to the network on which it is broadcast. Router-Busy-To-Network is used by a router to curtail the receipt of messages for specific DNETs or all DNETs. See Figure 6-9.

6.4.6 Router-Available-To-Network

This message is indicated by a Message Type of X'05' optionally followed by a list of 2-octet network numbers. It shall always be transmitted with a broadcast MAC address. Router-Available-To-Network is used by a router to enable or re-enable the receipt of messages for a specific list of DNETs or all DNETs. See Figure 6-8.

6.4.7 Initialize-Routing-Table

This message is indicated by a Message Type of X'06'. It is used to initialize the routing table of a router or to query the contents of the current routing table.

The format of the data portion of the Initialize-Routing-Table message is shown in Figure 6-11.

The Number of Ports field of this NPDU indicates how many port mappings are being provided in this NPDU. This field permits routing tables to be incrementally updated as the network changes. Valid entries in this field are 0-255. Following this field are sets of data indicating the DNET directly connected to this port or accessible through a dial-up PTP connection, Port ID, Port Info Length, and, in the case Port Info Length is non-zero, Port Info. If an Initialize-Routing-Table message is sent with the Number of Ports equal to zero, the responding device shall return its complete routing table in an Initialize-Routing-Table-Ack message without updating its routing table. If the Port ID field has a value of zero, then all table entries for the specified DNET shall be purged from the table. If the Port ID field has a non-zero value, then the routing information for this DNET shall either replace any previous entry for this DNET in the routing table or, if no such entry exists, be appended to the routing table.

The Port Info Length is an unsigned integer indicating the length of the Port Info field.

Number of Ports	1 octet
Connected DNET	2 octets
Port ID	1 octet
Port Info Length	1 octet
Port Info	J octets
. . .	. . .
Connected DNET	2 octets
Port ID	1 octet
Port Info Length	1 octet
Port Info	K octets

Figure 6-11. Format of the data portion of an Initialize-Routing-Table or Initialize-Routing-Table-Ack.

The Port Info field, if present, shall contain an octet string. A typical use would be to convey modem control and dial information for accessing a remote network via a dial-up PTP connection.

The Initialize-Routing-Table message shall be transmitted with the DER = TRUE.

6.4.8 Initialize-Routing-Table-Ack

This message is indicated by a Message Type of X'07'. It is used to indicate that the routing table of a router has been changed or the table has been queried through the receipt of an Initialize-Routing-Table message with the Number of Ports field set equal to zero. The data portion of this message, returned only in response to a routing table query, conveys the routing table information, and it has the same format as the data portion of an Initialize-Routing-Table message. See 6.4.7 and Figure 6-11.

6.4.9 Establish-Connection-To-Network

This massage is used to instruct a half-router to establish a new PTP connection that creates a path to the indicated network.

This message is indicated by a Message Type of X'08'. The complete format of the NPDU is shown in Figure 6-6. The 2-octet network number indicates the DNET that should be connected to by this half-router. The 1-octet "Termination Time Value" specifies the time, in seconds, that the connection shall remain established in the absence of NPDUs being sent on this connection. A value of 0 indicates that the connection should be considered to be permanent. See 6.7.1.4.

6.4.10 Disconnect-Connection-To-Network

This message is indicated by a Message Type of X'09' followed by a 2-octet network number. This message is used to instruct a router to disconnect an established PTP connection. The disconnection process shall follow the procedures described in Clause 10.

6.5 Network Layer Procedures

This subclause describes the network layer procedures to be followed by BACnet router and non-router nodes for both local and remote data transfer. "Local" means that the source and destination devices are on the same BACnet network. "Remote" means that the source and destination devices are on different BACnet networks. The source and destination networks are interconnected by zero or more intervening networks joined by BACnet routers to form a BACnet internetwork. See Figure 4-3.

6.5.1 Network Layer Procedures for the Transmission of Local Traffic

Upon receipt of an N-UNITDATA.request primitive, the network entity (NE) shall inspect the DNET portion of the 'destination_address' parameter. The absence of DNET indicates that the destination device resides on the same BACnet network as the device issuing this transmission request. The value of the 'network_priority' parameter shall be included in the NPCI control octet although its use by receiving non-router entities is unspecified. The NE shall prepare a control NPCI octet indicating the absence of DNET, DADR, HOP COUNT, SNET, and SADR, concatenate it with the 'data' parameter conveyed in the N-UNITDATA.request primitive, and issue a DL-UNITDATA data link request primitive. The concatenation of the NPCI and the NSDU (the 'data' parameter from the N-UNITDATA.request), the NPDU, is passed as the 'data' parameter of the data link primitive.

6.5.2 Network Layer Procedures for the Receipt of Local Traffic

Upon receipt of an NPDU from the data link layer (conveyed by the 'data' parameter of the DL-UNITDATA data link indication primitive) whose first octet indicates BACnet version one, the destination NE shall interpret the second octet of the NPDU as control NPCI. If bit 7 of the control NPCI indicates that the message contains an APDU, then the procedure in 6.5.2.1 is followed. Otherwise, a network layer message is being conveyed and the procedure in 6.5.2.2 applies.

6.5.2.1 Receipt of Local APDUs

If the control NPCI octet indicates the absence of a DNET field or a DNET field is present and contains the global broadcast address X'FFFF', the NE shall attempt to locate a BACnet application entity. If a BACnet application entity is found, the NE shall issue an N-UNITDATA.indication primitive with the portion of the data link data following the NPCI as the 'data' parameter. If the application entity is not found and the NE resides in a non-routing node, the data link data shall be discarded. If the DNET is present and not equal to the global broadcast address X'FFFF' and the NE resides in a non-routing node, the data link data shall likewise be discarded and no further action taken. If the DNET is present and the NE resides in a BACnet router, the NE shall take the actions specified in 6.5.4.

6.5.2.2 Receipt of Local Network Layer Messages

If the control NPCI octet indicates the absence of a DNET field or a DNET field is present and contains the global broadcast address X'FFFF', the NE shall attempt to interpret the network layer message. If the DNET field is present and the NE resides in a routing node and the network layer message can be interpreted, then the NE shall take the actions specified in 6.5.4. If the message cannot be interpreted, a Reject-Message-To-Network shall be returned to the device that sent the message.

If the DNET is present and not equal to the global broadcast address X'FFFF' and the NE resides in a non-routing node, the data link data shall be discarded and no further action taken. If the DNET is present and the NE resides in a BACnet router, the NE shall take the actions specified in 6.5.4.

6.5.3 Network Layer Procedures for the Transmission of Remote Traffic

Upon receipt of an N-UNITDATA.request primitive, the NE shall inspect the DNET portion of the 'destination_address' parameter. The presence of a DNET signifies that the destination device resides on a different BACnet network than the device issuing this transmission. A DNET value of X'FFFF' signifies a global broadcast and indicates that the message is to be directed to all local routers via a broadcast message on the local network. The NE shall prepare an NPCI control octet indicating the presence of DNET, DADR, and Hop Count but the absence of SNET and SADR. The NE shall also fill in the network priority field using the supplied parameter. The resulting control, priority, and address information shall then be concatenated with the 'data' parameter conveyed in the N-UNITDATA.request primitive and issued as a DL-UNITDATA data link request primitive. The concatenation of the NPCI and the 'data' parameter from the N-UNITDATA.request (the NSDU), the NPDU, is passed as the 'data' parameter of the data link primitive. The DA portion of the 'destination_address' parameter passed to the data link layer shall be the MAC address of the BACnet router corresponding to the DNET parameter or the appropriate broadcast DA if the address of the router is initially unknown. The broadcast DA is also to be used if the DNET global broadcast network number is present.

Note that four methods exist for establishing the address of a BACnet router for a particular DNET: 1) the address may be established manually at the time a device is configured, 2) the address may be learned by issuing a Who-Is request and noting the SA associated with the subsequent I-Am message (assuming the device specified in the Who-Is is located on a remote DNET and the I-Am message was handled by a router on the local network), 3) by using the network layer message Who-Is-Router-To-Network, and 4) by using the local broadcast MAC address in the initial transmission to a device on a remote DNET and noting the SA associated with any subsequent responses from the remote device. Which method is used shall be a local matter.

6.5.4 Network Layer Procedures for the Receipt of Remote Traffic

Upon receipt of an NPDU from the data link layer (conveyed by the 'data' parameter of the DL-UNITDATA indication primitive) whose first octet indicates BACnet version one, the NE shall interpret the second octet of the NPDU as control NPCI. If the NPCI control octet indicates the presence of a DNET field whose value is not X'FFFF' and the NE resides in a BACnet device that is not a router, the message shall be discarded. If the NPCI control octet indicates the presence of a DNET field and the NE resides in a BACnet router, it shall place the NPDU in its message queue (or queues, if separate queues are maintained for each DNET), arranged in order by priority. Within each priority, the messages shall be arranged in first-in-first-out order. If the NPCI control octet indicates that the NPDU contains a network layer message, the NE shall, in addition, inspect the Message Type field. If this field indicates the presence of a Reject-Message-To-Network message, the NE shall carry out the processing specified in 6.6.3.5. If the SNET and SADR fields are present, the message has arrived from a peer router. If the SNET and SADR fields are absent, the message originated on a network directly connected to the router. In the latter case, the router shall add the SNET and SADR to the NPCI based on the router's knowledge of the network number of the network from which the message arrived, alleviating the requirement that the originating station know its own network number. The SADR field shall be set equal to the SA of the incoming NPDU.

Three possibilities exist: either the router is directly connected to the network referred to by DNET, the message must be relayed to another router for further transmission, or a global broadcast is required. In the first case, DNET, DADR, and Hop Count shall be removed from the NPCI and the message shall be sent directly to the destination device with DA set equal to DADR. The control octet shall be adjusted accordingly to indicate only the presence of SNET and SADR. In the second case, the Hop Count shall be decremented. If the Hop Count is still greater than zero, the message shall be sent to the next router on the path to the destination network. If the next router is unknown, an attempt shall be made to identify it using a Who-Is-Router-To-Network message. If the Hop Count is zero, then the message shall be discarded. If the DNET global broadcast network number is present and the Hop Count is greater than zero, the router shall broadcast the message on each network to

which the router is directly connected, except the network of origin, using the broadcast address appropriate to each data link. If the DNET global broadcast network number is present and the Hop Count is zero, then the message shall be discarded.

6.6 BACnet Routers

BACnet routers are devices that interconnect two or more BACnet networks to form a BACnet internetwork. A router may, or may not, provide BACnet application layer functionality. BACnet routers make use of BACnet network layer protocol messages to maintain their routing tables. Routers perform the routing tasks described in 6.5. See Figure 6-12 for a flow chart of router operation.

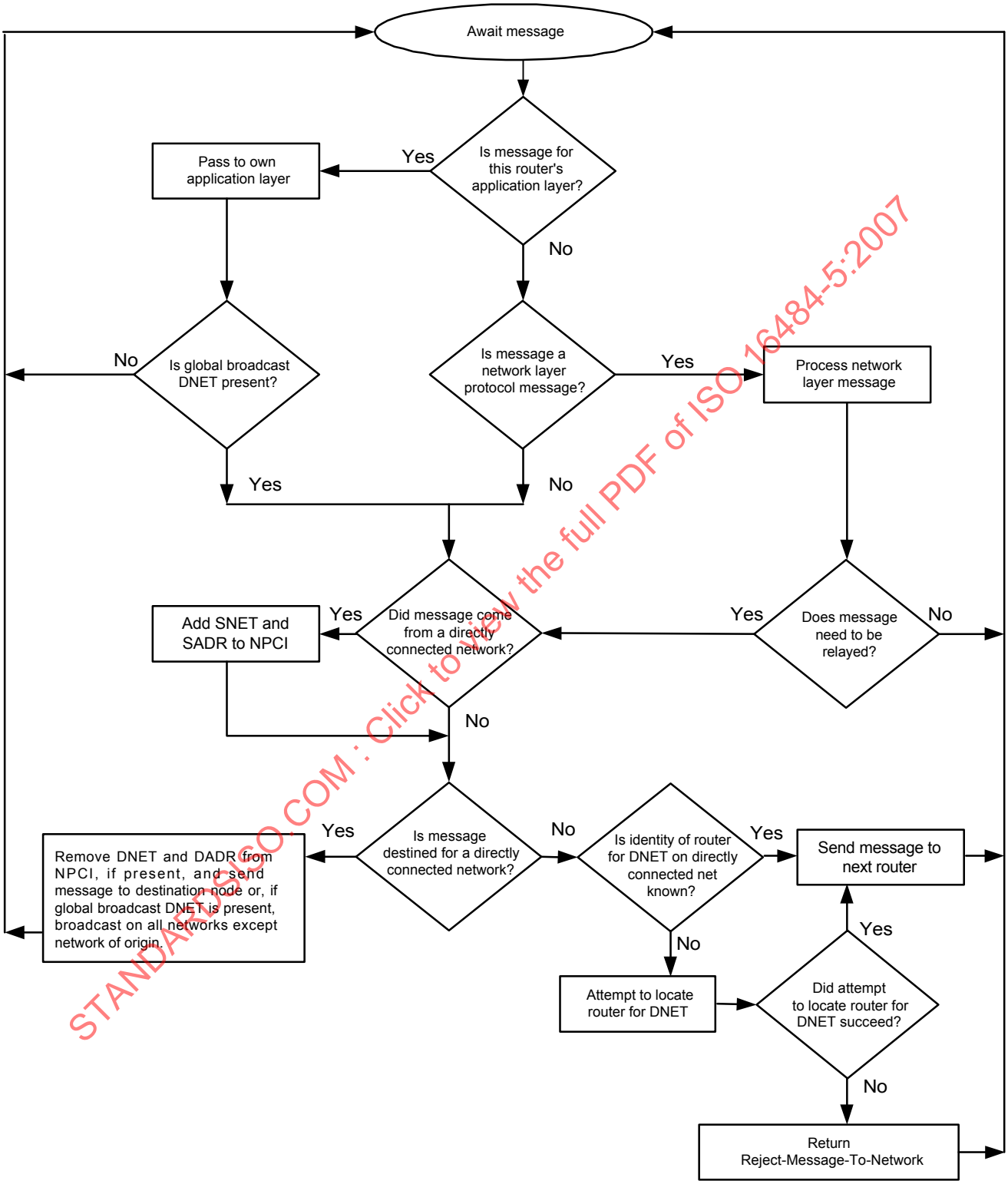


Figure 6-12. BACnet message routing.

6.6.1 Routing Tables

By definition, a router is a device that is connected to at least two BACnet networks. Each attachment is through a "port." A "routing table" consists of the following information for each port:

(a) the MAC address of the port's connection to its network;

(b) the 2-octet network number of the directly connected network;

(c) a list of network numbers reachable through the port along with the MAC address of the next router on the path to each network number and the reachability status of each such network.

The "reachability status" is an implementation-dependent value that indicates whether the associated network is able to receive traffic. The reachability status shall be able to distinguish, at a minimum, between "permanent" failures of a route, such as might result from the failure of a router, and "temporary" unreachability due to the imposition of a congestion control restriction.

6.6.2 Start-up Procedures

Upon start-up, each router shall broadcast out each port an I-Am-Router-To-Network message containing the network numbers of each accessible network except the networks reachable via the network on which the broadcast is being made. This enables routers to build or update their routing table entries for each of the network numbers contained in the message.

6.6.3 Router Operation

This subclause describes the operation of BACnet routers.

6.6.3.1 BACnet NPDUs - General

If a BACnet NPDU is received with NPCI indicating that the message should be relayed by virtue of the presence of a non-broadcast DNET, the router shall search its routing table for the indicated network number. Normal routing procedures are described in 6.5. If, however, the network number cannot be found in the routing table or through the use of the Who-Is-Router-To-Network message, the router shall generate a Reject-Message-To-Network message and send it to the node that originated the BACnet NPDU. If the NPCI indicates either a remote or global broadcast, the message shall be processed as described in 6.3.2.

6.6.3.2 Who-Is-Router-To-Network

This message may be generated by a non-routing BACnet node or by a BACnet router. If the message is broadcast with a specific network number, one I-Am-Router-To-Network message should be returned at most, originating at the router on the local network that is the next router to the specified destination network. If the 2-octet network number is omitted, each responding router shall reply with an I-Am-Router-To-Network message containing all networks reachable through it, including those that may be temporarily unreachable due to the imposition of a congestion control restriction and excluding the networks reachable through the port from which the Who-Is-Router-To-Network message was received. Who-Is-Router-To-Network will generally be broadcast but may be directed to a specific router to learn the contents of its router table. In the event a router receives multiple I-Am-Router-To-Network messages pertaining to the same network, the router shall assume that each new I-Am-Router-To-Network message represents a modification in the system configuration and shall update its routing information. If the router has an established PTP connection (see Clause 10) that conflicts with this new information, the PTP connection shall be terminated using the disconnect procedures defined in Clause 10. Thus the last message received shall take precedence over all previous messages.

When a router receives a Who-Is-Router-To-Network message specifying a particular network number, it shall search its routing table for the network number contained in the message. If the specified network number is found in its table and the port through which it is reachable is not the port from which the Who-Is-Router-To-Network message was received, the router shall construct an I-Am-Router-To-Network message containing the specified network number and send it to the node that generated the request using a broadcast MAC address, thus allowing other nodes on this network to take advantage of the routing information.

If the network number is not found in the routing table, the router shall attempt to discover the next router on the path to the indicated destination network by generating a Who-Is-Router-To-Network message containing the specified destination network number and broadcasting it out all its ports other than the one from which the Who-Is-Router-To-Network message

arrived. Two cases are possible. In case one the received Who-Is-Router-To-Network message was from the originating device. For this case, the router shall add SNET and SADR fields before broadcasting the subsequent Who-Is-Router-To-Network. This permits an I-Could-Be-Router-To-Network message to be directed to the originating device. The second case is that the received Who-Is-Router-To-Network message came from another router and it already contains SNET and SADR fields. For this case, the SNET and SADR shall be retained in the newly generated Who-Is-Router-To-Network message.

If the Who-Is-Router-To-Network message does not specify a particular destination network number, the router shall construct an I-Am-Router-To-Network message containing a list of all the networks it is able to reach through other than the port from which the Who-Is-Router-To-Network message was received and transmit it in the same manner as described above. The message shall list all networks not flagged as permanently unreachable, including those that are temporarily unreachable due to the imposition of congestion control restrictions. Networks that may be reachable through a PTP connection shall be listed only if the connection is currently established.

6.6.3.3 I-Am-Router-To-Network

At router start-up, each router shall broadcast locally an I-Am-Router-To-Network message on each directly connected network as specified in 6.6.2. Each such message shall list each accessible network number except the number of the network on which the broadcast is being made. This broadcast allows other routers to update their routing tables whenever a new router joins the internetwork. In addition, an I-Am-Router-To-Network message shall be broadcast locally upon the receipt of a Who-Is-Router-To-Network message containing a network number matching a network number contained in the router's routing table, provided that the port through which it is reachable is not the port from which the Who-Is-Router-To-Network message was received.

If one or more of the reachable networks listed in the I-Am-Router-To-Network message is reached through a directly connected PTP connection, transmitting the I-Am-Router-To-Network message shall start or restart a connection termination delay timer. The PTP connection shall not be terminated before this delay timer expires. The connection termination delay timer shall be configurable with a default value of sixty seconds.

Upon receipt of an I-Am-Router-To-Network message, the router shall search its routing table for entries corresponding to each network number contained in the message. If no entry is found for a particular network number, a new entry shall be created. If an entry is found but the MAC address or port of the next router on the path to the indicated network differs from that found in the table, the MAC address in the table shall be replaced with that of the router originating the I-Am-Router-To-Network message. This ensures that all routers will have the most current information in their tables. Whether the router table was updated or not, the router shall then generate an I-Am-Router-To-Network message for all the network numbers contained in the received I-Am-Router-To-Network message and broadcast the new message, using the local broadcast MAC address, out all ports other than the one from which the previous message was received.

6.6.3.4 I-Could-Be-Router-To-Network

This message is generated by a half-router in response to a Who-Is-Router-To-Network message containing a specific 2-octet network number when the responding half-router has the capability of establishing a PTP connection that can be used to reach the desired network, but this PTP connection is not currently established. In the event that a Who-Is-Router-To-Network message is received in which the 2-octet network number field is absent, such as is used to determine lists of networks reachable through active routers, the I-Could-Be-Router-To-Network message shall not be returned. The I-Could-Be-Router-To-Network message shall be directed to the device that originated the Who-Is-Router-To-Network message. The procedures to be used to establish a PTP connection are described in 6.7 and Clause 10.

6.6.3.5 Reject-Message-To-Network

Reject-Message-To-Network is generated by a router when it receives a message that it is unable to relay to the DNET specified in the NPCI or if it receives an unknown network layer message directed specifically to that router. The reasons for rejecting the message are set forth in 6.4.4.

When a router receives a Reject-Message-To-Network message with a rejection reason octet containing a value of 1or 2, it shall search its routing table for the network number specified in the Reject-Message-To-Network message. If the network number is found, the status information for this network number shall be updated to indicate that the network is permanently unreachable if the reject reason was 1 or unreachable due to flow control if the reject reason was 2. In addition, regardless of the contents of the rejection reason octet, the router shall relay the message in the normal manner to the originating node

specified in the NPCI using the procedures of 6.5. A rejection reason of 1 is to be considered a serious error condition and should be reported to a local or remote network management entity. The nature of this reporting procedure is a local matter.

6.6.3.6 Router-Busy-To-Network

If a router wishes to curtail the receipt of messages for specific DNETs or all DNETs, it shall generate a Router-Busy-To-Network message.

If a router temporarily wishes to receive no more traffic for one or more specific DNETs, it shall broadcast a Router-Busy-To-Network message with a list of the 2-octet network numbers corresponding to these DNETs. If the 2-octet network numbers are omitted, it means the router wishes to stop the flow of messages to all the networks it normally serves.

Each router receiving a Router-Busy-To-Network message shall update its routing table to indicate that the specified DNETs are not reachable, set or reset a 30-second timer for this status, and broadcast a Router-Busy-To-Network message out each port other than the one on which it was received so that all routers may learn of the congestion control restriction. Normally, a Router-Busy-To-Network message should be followed in a short time by a Router-Available-To-Network message indicating that the congestion control restriction has been lifted. In the event that this does not happen within a timeout period of 30 seconds or if a node that has recently joined the network did not receive a previous Router-Busy-To-Network message, it may attempt a transmission to the "busy" router. If the router is able to accept the message, it shall do so and, at its discretion, again broadcast a Router-Busy-To-Network message for the benefit of this node and any others that may not have received the previous transmission. If the router is unable to accept the message, it shall immediately return a Reject-Message-To-Network to the sender. It may then also broadcast another Router-Busy-To-Network message for the reasons cited above.

6.6.3.7 Router-Available-To-Network

When a router wishes to re-enable the receipt of messages for a specific list of DNETs, or all DNETs, previously curtailed by a Router-Busy-To-Network message, it shall broadcast a Router-Available-To-Network message. If the message is broadcast with a list of 2-octet network numbers, it means that the router is now able to receive traffic for these specific DNETs. If the 2-octet network numbers are omitted, the router wishes to re-enable the flow of messages to all the networks it serves.

Each router receiving a Router-Available-To-Network message shall update its routing table to indicate that the specified DNETs are now reachable and broadcast a Router-Available-To-Network message out each port other than the one on which it arrived so that all routers may learn of the lifting of the congestion control restriction.

6.6.3.8 Initialize-Routing-Table

The Initialize-Routing-Table message is generated by any node that has been programmed to provide the initial routing table information to one or more BACnet routers or wishes to query the contents of the current routing tables. The establishment of the contents of the routing table and the circumstances under which Initialize-Routing-Table messages are generated are local matters. In addition, an Initialize-Routing-Table message with Number of Ports set equal to zero shall cause the responding device to return its complete routing table in an Initialize-Routing-Table-Ack message without updating its routing table.

When a router receives this message containing a routing table, indicated by a non-zero value in the Number of Ports field, it shall update its current port-to-network-number mappings for each network specified in the NPDU with the information contained in the NPDU and return an Initialize-Routing-Table-Ack message without any routing table data to the source. When a router receives this message in the form of a routing table query, indicated by a zero value in the Number of Ports field, it shall return an Initialize-Routing-Table-Ack message to the source containing a complete copy of its routing table as described in 6.6.3.9.

6.6.3.9 Initialize-Routing-Table-Ack

This message is sent by a router after the reception and servicing of an Initialize-Router-Table message. If the router is acknowledging a table update message, signified by a non-zero value in the Number of Ports field, it shall return an Initialize-Routing-Table-Ack without data. If the router is acknowledging a table query message, indicated by a zero value in the Number of Ports field, it shall return a complete copy of its routing table. If a complete copy of the table cannot be returned in a single acknowledgment, the router shall send multiple acknowledgments, each containing a portion of the routing table until the entire table has been sent.

6.6.3.10 Establish-Connection-To-Network

Upon receipt of an Establish-Connection-To-Network message, a half-router shall attempt to establish a PTP connection using the procedures described in 6.7 and Clause 10.

6.6.3.11 Disconnect-Connection-To-Network

Upon receipt of a Disconnect-Connection-To-Network message, a half-router shall terminate an established PTP connection using the procedures described in 6.7 and Clause 10.

6.6.4 Router Congestion Control

Routers may wish to temporarily suspend the receipt of messages destined for a specific network or, possibly, all networks. Normally, this would be the result of impending buffer overflow in the router itself but could also occur because of a buffer problem with a downstream router on the path to a particular network. The messages used to impose and remove congestion control restrictions are Router-Busy-To-Network and Router-Available-To-Network. The algorithm for determining that congestion control should be imposed or removed is not specified in this standard but would most likely involve such factors as the percentage of buffer space currently occupied and, possibly, the rate at which new messages have been arriving at the router.

6.7 Point-To-Point Half-Routers

In BACnet networks that are interconnected across PTP connections (as defined in Clause 10), the procedures for half-router establishment and synchronization are different from those for normal routers. This is due to two unique characteristics of this type of connection. First, since a PTP connection may be established over a wide area network, such as the public telephone network, it is sometimes advantageous to limit the duration of these connections. This causes temporary half-router connections that must be controlled by BACnet. Secondly, PTP connections are always established between two half-routers that together form a single router. A diagram of this router architecture is shown in Figure 6-13. When a connection is established, both half-routers also need to update their routing tables to reflect any new or updated routing information stored by the partner half-router.

To control the link establishment, link termination, and route-learning functions of a PTP half-router, BACnet has defined five network layer messages. The I-Could-Be-Router-To-Network message announces that a half-router has the capability to connect to a requested network but does not have an active connection. The Establish-Connection-To-Network message requests that a connection be established. The Disconnect-Connection-To-Network message requests that an active connection be disconnected. Routing table initialization may be performed using the Initialize-Routing-Table and Initialize-Routing-Table-ACK messages. Thereafter, the half-router maintains its table using the same procedures as other active routers regardless of whether any active PTP connections exist.

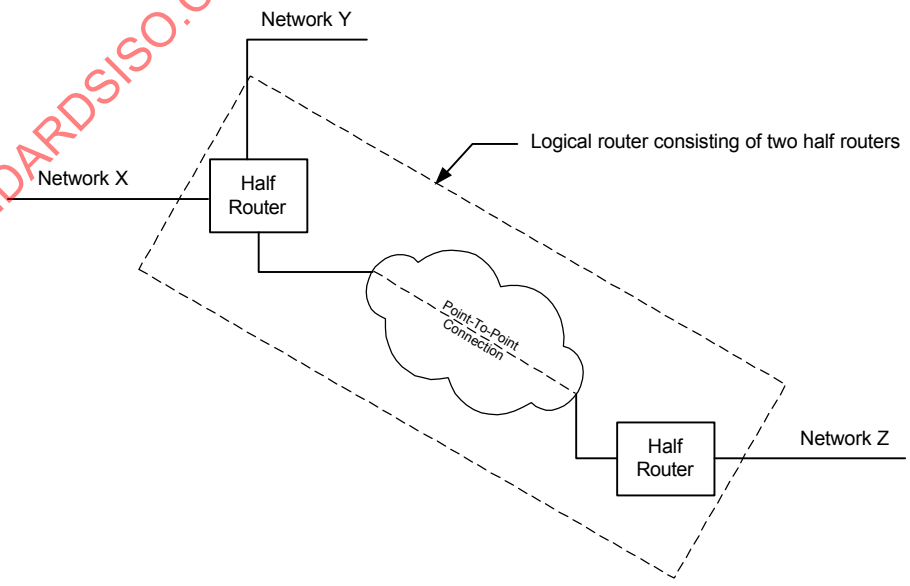


Figure 6-13. Upon a PTP connection, two half-routers combine to become a router.

6.7.1 Procedures for Establishing a New PTP Connection Between Two Half-Routers

As specified in 6.5.3, one of the methods of establishing the address of a BACnet router for a particular DNET is by having the initiating Network Entity (NE) send a Who-Is-Router-To-Network message for the DNET. A router that has an active connection to the DNET, either directly connected or over an established PTP connection, shall respond with an I-Am-Router-To-Network message. A half-router that does not have an active connection but could initiate a PTP connection to the requested DNET shall respond with an I-Could-Be-Router-To-Network message.

6.7.1.1 Initiating Network Entity (NE) Procedure

To determine a new route, the NE shall issue a Who-Is-Router-To-Network message for the unknown DNET. After a locally specified time period, the initiating NE shall determine the most suitable half-router to the DNET. The algorithm for selecting the most suitable half-router is a local matter. As an aid to help select the most suitable half-router, each I-Could-Be-Router-To-Network message has a 1-octet field to indicate the expected performance of the half router's PTP connection. Upon selection of a suitable half-router, the initiating NE shall send an Establish-Connection-To-Network message to this half-router and wait to receive an I-Am-Router-To-Network message for this DNET. When the I-Am-Router-To-Network message is received, the initiating NE may send NPDUs for this DNET. If, after a locally specified time period, an I-Am-Router-To-Network message is not received by the initiating NE, the initiating NE shall send a Disconnect-Connection-To-Network to the selected half-router and may try this procedure again to find another half-router.

6.7.1.2 Initiating Half-Router Procedure

Upon receipt of an Establish-Connection-To-Network message, a half-router shall try to establish the requested connection. If the connection is established, the initiating half-router shall forward the Establish-Connection-To-Network message to the answering half-router, synchronize its routing table with the routing table of the answering half-router partner using the procedures in 6.7.3, broadcast an I-Am-Router-To-Network message containing all of the DNETs accessible through the answering half-router to all directly connected networks, and start an activity timer (T_{active}).

When the connection is established, the initiating half-router shall adjust its routing table to indicate that any DNETs accessible from the answering half-router have a "reachability status" that is "reachable" and continue with other normal operations.

If the connection cannot be established, the initiating half-router shall adjust its routing table to indicate that any DNETs accessible from the answering half-router have a "reachability status" that is "temporarily unreachable" and continue with other normal operations. If a Disconnect-Connection-To-Network message is received, the half-router shall ignore the message.

If the connection is in the process of being established and a Disconnect-Connection-To-Network message is received, the half-router shall immediately end the connection establishment procedure.

If the connection is in the process of being established and a Who-Is-Router-To-Network message is received for a DNET accessible through the PTP connection, the half-router shall respond with an I-Am-Router-To-Network, followed by a Router-Busy-To-Network message. If a message is received for the DNET before the connection is established, it shall be rejected with a rejection reason = 2. When the connection is established, the initiating half-router shall issue a Router-Available-To-Network message. If the connection cannot be established, the half-router shall issue a Router-Available-To-Network message but reject, with a rejection reason = 1, any message received for the DNET. The reason for rejecting the message is to expedite error handling.

If the connection is in the process of being established and a I-Am-Router-To-Network message is received for the DNET to which the initiating half-router is attempting a connection, the connection establishment shall be immediately terminated.

If the connection is in the process of being established and an Establish-Connection-To-Network message to the same DNET is received, the Termination Time Value shall be evaluated. If the new Termination Time Value is greater than the Termination Time Value of the original Establish-Connection-To-Network message, the new Termination Time Value shall be used by the Activity Timer. The connection process shall then proceed normally.

6.7.1.3 Answering Half-Router Procedure

Upon connection establishment from a PTP half-router, the answering half-router shall set an activity timer (T_{active}) based upon a received Establish-Connection-To-Network message from the initiating half-router, synchronize its routing table with

the routing table of the initiating half-router partner using the procedures in 6.7.3, and broadcast an I-Am-Router-To-Network message containing all of the DNETs accessible through the initiating half-router to all directly connected networks. If the connection is terminated, the answering half-router's routing table shall be adjusted to indicate that any DNET accessible from the initiating half-router has a "reachability status" that is "temporarily unreachable," if the answering half-router is able to re-establish the connection or "permanently unreachable" if the answering half-router is unable to re-establish the connection. If the connection is established, the answering half-router's routing table shall be adjusted to indicate that any DNET accessible from the initiating half-router has a "reachability status" that is "reachable."

6.7.1.4 Activity Timer (T_{active})

The activity timer (T_{active}) is the time that a half-router shall wait for the absence of any messages being routed over its PTP connection before it attempts to automatically disconnect the connection. This timer shall be set to the Termination Time Value field from the Establish-Connection-To-Network message. If the Termination Time Value is set to zero, the activity time shall be considered infinite.

6.7.1.4.1 Initiating Half-Router Procedure

Upon receipt of an Establish-Connection-To-Network message, an initiating half-router shall set the activity timer (T_{active}) to the Termination Time Value field from the Establish-Connection-To-Network message. If the Termination Time Value is set to zero, the activity time shall be considered infinite.

6.7.1.4.2 Answering Half-Router Procedure

Upon receipt of an Establish-Connection-To-Network message from the initiating half-router, the answering half-router shall set the activity timer (T_{active}) to the Termination Time Value field from the Establish-Connection-To-Network message. If the Termination Time Value is set to zero, the activity time shall be considered infinite.

6.7.2 Procedures for Disconnecting a PTP Connection in a Half-Router

There are three bases for disconnecting a PTP connection established by a half-router. The first is by a Network Entity (NE) initiating a Disconnect-Connection-To-Network message. The second is by a timer expiration indicating that the connection has been inactive for an abnormal period of time. The third basis for disconnecting a connection is to compensate for a configuration error. The specification of this procedure is given in 6.7.4.

6.7.2.1 Active Disconnection of a PTP Connection

6.7.2.1.1 Initiating Network Entity (NE) Procedure

If the initiating NE determines that the half-router connection is no longer needed, it may send a Disconnect-Connection-To-Network message to the half-router. The routing table entry for this DNET shall be immediately set to "disconnected."

6.7.2.1.2 Initiating/Answering Half-Router Procedure

Upon receipt of a Disconnect-Connection-To-Network message, a half-router shall disconnect the PTP connection as specified in Clause 10. When the connection is terminated, the half-router shall adjust its routing table to indicate that any DNETs accessible from the previously connected half-router have a "reachability status" that is "temporarily unreachable" if the half-router is able to re-establish the connection, or "permanently unreachable" if the half-router is unable to re-establish the connection.

6.7.2.2 Timed Disconnection of a PTP Connection

If the activity timer (T_{active}) expires, a half-router shall disconnect the PTP connection as specified in Clause 10. When the connection is terminated, the half-router shall adjust its routing table to indicate that any DNETs accessible from the previously connected half-router have a "reachability status" that is "temporarily unreachable" if the half-router is able to re-establish the connection, or "permanently unreachable" if the half-router is unable to re-establish the connection.

6.7.2.3 Restarting of the Activity Timer (T_{active})

The Activity Timer (T_{active}) in each half-router shall be restarted to its original value contained in the initiating Establish-Connection-To-Network whenever an NPDU is transferred over the PTP link.

6.7.3 Procedures for Synchronizing Half-Router Routing Tables

Upon the establishment of a PTP connection between two half-routers, the routing tables of the half-routers shall be synchronized. This is accomplished using the I-Am-Router-To-Network message.

Upon connection establishment, the two half-routers shall exchange I-Am-Router-To-Network messages. Each message shall contain all of the reachable (before the connection was established) DNETs connected through this router. In the event that a duplicate network connection is discovered by the procedure specified in 6.7.4.2, synchronization of routing tables shall fail, causing the routing table entry for any DNET accessible from the peer half-router to have a reachability status of "temporarily unreachable."

6.7.4 Error Recovery Procedures

6.7.4.1 Recovering from Routing Requests to Unconnected Networks

Since PTP connections may be temporary in nature, there is a possibility that a half-router may receive a message bound for a DNET connection that has been disconnected. If another route is not in place through a different port than the one from which the message was received, this is considered an error. To recover from this situation, the receiving half-router shall reject this message with a Reject-Message-To-Network message using rejection reason = 1. The initiating Network Entity shall recover from this error by initiating the procedure for establishing a new PTP connection through a half-router as described in 6.7.1.

6.7.4.1.1 Disconnected Half-Router Procedure

Upon receipt of a message that is requested to be routed across a PTP connection that is disconnected, the half-router shall determine if another route is in place. If no other route is in place or if the next hop of this route is identical to the path from which the message was received, the half-router shall issue a Reject-Message-To-Network for this message with a rejection reason = 1 and discard the message. If another acceptable route is in place, the message shall be forwarded on this route.

6.7.4.1.2 Initiating Network Entity (NE) Procedure

If the initiating NE receives a Reject-Message-To-Network, it shall attempt to determine a new route to the DNET after waiting for a random back-off period. The random back-off period, in seconds, is determined by the initiating NE through the generation of a random number of either 0 or 1 and then multiplying this number by 40. The initiating NE shall not try to re-establish the network connection until the back-off period has expired. If during the back-off period the initiating NE learns of a valid route to the required DNET, the initiating NE shall use this path and consider the network connection re-established. Upon expiration of the back-off period, if the network connection has not been re-established, the initiating NE shall attempt to determine a new route to the DNET using the procedure for establishing a new PTP connection through a half-router as described in 6.7.1.

6.7.4.2 Recovering from Duplicate Network Connections

In the unlikely event that two or more PTP connections are made to single DNET, at least one of the connections shall be terminated and the routing tables in all routers shall be made consistent. The procedure to ensure that no loop exists consists of having every half-router examine each received I-Am-Router-To-Network message for another path to any of the half-router's directly connected networks. The existence of a second path to a directly connected network indicates that a loop is formed. If a loop is detected, the half-router shall disconnect its PTP connection thereby breaking the loop.

6.7.4.2.1 Half-Router Procedure for Receipt of Conflicting I-Am-Router-To-Network Messages

If during the initialization or lifetime of a PTP connection a half-router hears an I-Am-Router-To-Network message from the PTP connection containing a DNET to one of the half-router's directly connected networks, the half-router shall immediately terminate the connection.

6.7.4.2.2 Half-Router Procedure for Initiation of I-Am-Router-To-Network Messages

As an added safety measure to ensure that duplicate paths are discovered in a timely manner, a half-router shall broadcast one or more I-Am-Router-To-Network message(s) once every five minutes when a PTP connection is in place. The DNETs in this message shall be all of the DNETs accessible through the PTP connection. In the event this list of DNETs would exceed the maximum NPDU length of the network being utilized, the list shall be divided into segments that fit on the network and sent in consecutive I-Am-Router-To-Network messages.

6.7.4.2.3 Half-Router Procedure for Decrementing the Hop Count

To reduce the number of circularly routed messages in a misconfigured system, BACnet NPDUs contain a hop count that limits the number of routers that shall forward the NPDU. In routers that provide the capability to configure the amount that

the Hop Count field shall be decremented when an NPDU is forwarded, a network administrator may optimize the damping of looping messages. One method to do this is to find the path in the network that requires the maximum number of router hops. The amount to decrement the NPDU Hop Count field in every router on the network is then calculated as the integer division of 255/(maximum number of router hops). On a PTP connection, the half of the router that forwards the NPDU onto a non-PTP network shall decrement the Hop Count field.

7 DATA LINK/PHYSICAL LAYERS: ISO 8802-3 ("Ethernet") LAN

This clause describes the transport of BACnet LSDUs using the services of the data link and physical mechanisms described in International Standards ISO 8802-2: *Information processing systems- Local area networks- Part 2: Logical link control* and ISO/IEC 8802-3: *Information processing systems- Local area networks- Part 3: Carrier sense multiple access with collision detection (CSMA/CD) access method and physical layer specifications*. The clauses of ISO 8802-2 pertaining to Class I LLC and Type 1 Unacknowledged Connectionless-Mode Service as well as all of ISO/IEC 8802-3, as amended and extended by the International Organization for Standardization, are deemed to be included in this standard by reference.

7.1 The Use of ISO 8802-2 Logical Link Control (LLC)

Standard BACnet networks may pass BACnet link service data units (LSDUs) using the data link services of ISO 8802-2 Logical Link Control (LLC). A BACnet LSDU consists of an NPDU constructed as described in Clause 6. BACnet devices using ISO 8802-3 LAN technology shall conform to the requirements of LLC Class I, subject to the constraints specified in this clause. Class I LLC consists of Type 1 LLC - Unacknowledged Connectionless-Mode service. LLC parameters shall be conveyed using the DL-UNITDATA primitives as described in the referenced standards.

All BACnet devices conforming to this section shall be capable of accepting properly formed Unnumbered Information (UI) commands and responding to XID Exchange Identification and TEST commands.

7.2 Parameters Required by the LLC Primitives

The DL-UNITDATA primitive requires source address, destination address, data, and priority parameters. The source and destination addresses each consist of the logical concatenation of a medium access control (MAC) address and a link service access point (LSAP). The MAC address is a 6-octet value determined by the network interface hardware. The LSAP is the single-octet value X'82' and is used to indicate that an LSDU contains BACnet data. The data parameter is the NPDU from the network layer. Since the ISO 8802-3 MAC layer only operates at a single priority with only one class of service, the value of the priority parameter is not specified in this standard.

7.3 Parameters Required by the MAC Primitives

The ISO/IEC 8802-3 MAC layer primitives are the MA-DATA.request and MA-DATA.indication. These convey the encoded LLC data using the source and destination MAC addresses described above. Again, since only one class of service is provided, the value of the 'service_class' parameter is unspecified. See Figure 7-1.

7.4 Physical Media

The physical media specified by ISO 8802-3 and subsequent addenda are equally acceptable.

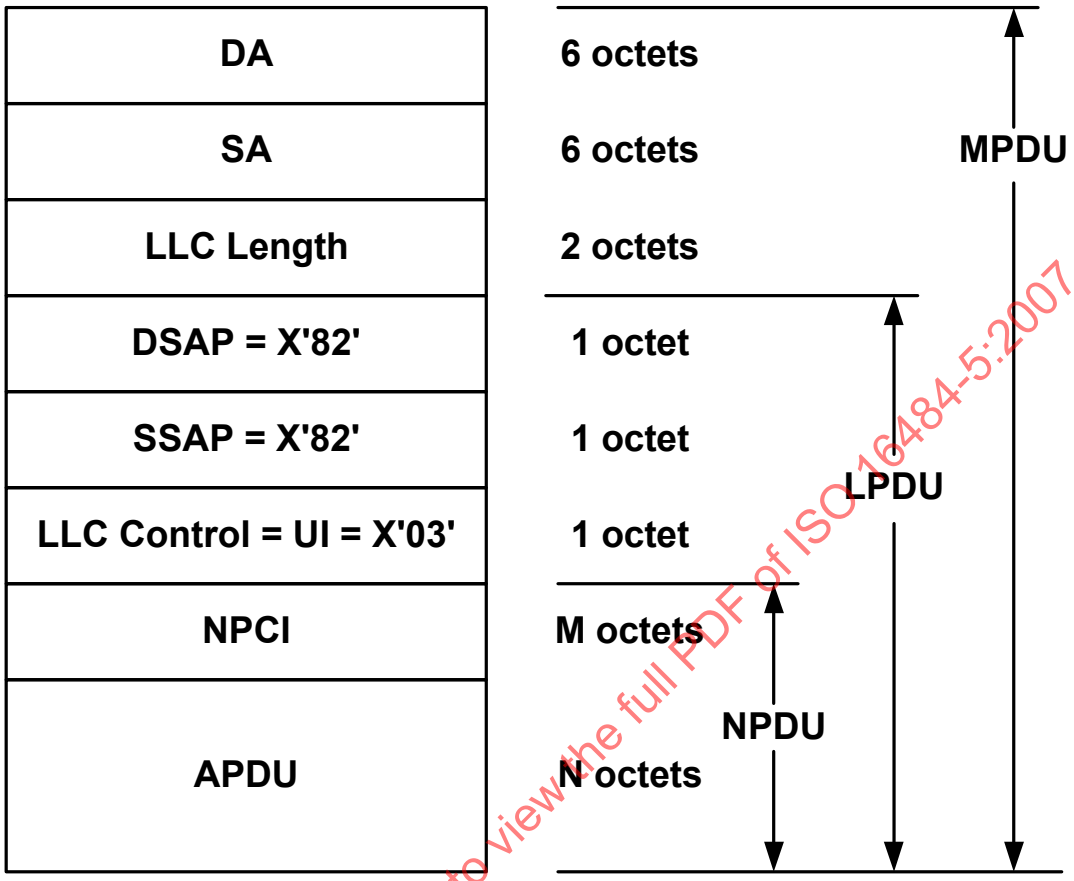


Figure 7-1. Format of an MPDU on an ISO 8802-3 LAN.

8 DATA LINK/PHYSICAL LAYERS: ARCNET LAN

This clause describes the transport of BACnet LSDUs using the services of the data link and physical mechanisms described in ATA/ANSI 878.1, *ARCNET Local Area Network Standard*. The *ARCNET Local Area Network Standard*, as amended and extended by the ARCNET Trade Association, is deemed to be included in this standard by reference.

8.1 The Use of ISO 8802-2 Logical Link Control (LLC)

Standard BACnet networks may pass BACnet link service data units (LSDUs) using the data link services of ISO 8802-2 LLC. A BACnet LSDU consists of an NPDU constructed as described in Clause 6. BACnet devices using ARCNET LAN technology shall conform to the requirements of LLC Class I, subject to the constraints specified in this clause. Class I LLC service consists of Type 1 LLC - Unacknowledged Connectionless-Mode service. LLC parameters shall be conveyed using the DL-UNITDATA primitives as described in the referenced standards.

The mapping of these primitives onto the ARCNET MAC layer primitives is described in 8.3.

All BACnet devices conforming to this section shall be capable of accepting and responding to XID Exchange Identification and TEST commands.

8.2 Parameters Required by the LLC Primitives

The DL-UNITDATA primitive requires source address, destination address, data, and priority parameters. The source and destination addresses each consist of the logical concatenation of a medium access control (MAC) address, link service access point (LSAP), and a system code (SC). The MAC address is a 1-octet value determined by the network interface hardware; the LSAP used to indicate that an LSDU contains BACnet data is the single octet value X'82'; and the SC used to indicate a BACnet frame is the single-octet value X'CD'. The data parameter is the NPDU from the network layer. Since the ARCNET MAC sublayer only operates at a single priority with only one class of service, the value of the priority parameter is not specified in this standard.

8.3 Mapping the LLC Services to the ARCNET MAC Layer

The Type 1 Unacknowledged Connectionless LLC service shall map directly onto the ARCNET MA_DATA request primitive. Although a successful transmission results in an acknowledgment from the destination MAC sublayer, no indication is expected, or provided, to the LLC sublayer.

ARCNET does not permit MSDUs of length 253, 254, or 255 octets. A BACnet LPDU of length 0 to 252 octets shall be conveyed as the entire MSDU of an ARCNET MPDU (frame) with a single Information length (IL) octet. A BACnet LPDU of length 253 to 504 octets shall be conveyed as the initial octets of the MSDU of an ARCNET MPDU with two Information length (IL) octets. In this case, the LPDU shall be followed by three octets of unspecified value, such that the net length of the MSDU is 256 to 507 octets. When an ARCNET MPDU with two Information length octets is received, the final 3 octets of the MSDU shall be ignored.

An LPDU longer than 504 octets cannot be conveyed via ARCNET.

8.4 Parameters Required by the MAC Primitives

The ARCNET MAC layer primitives are MA-DATA.request, MA-DATA.indication, and MA-DATA.confirmation. These convey the encoded LLC data (MSDU) using the source and destination MAC addresses described above in conjunction with the BACnet system code. See Figure 8-1.

8.5 Physical Media

The physical media specified by the ARCNET standard are equally acceptable.

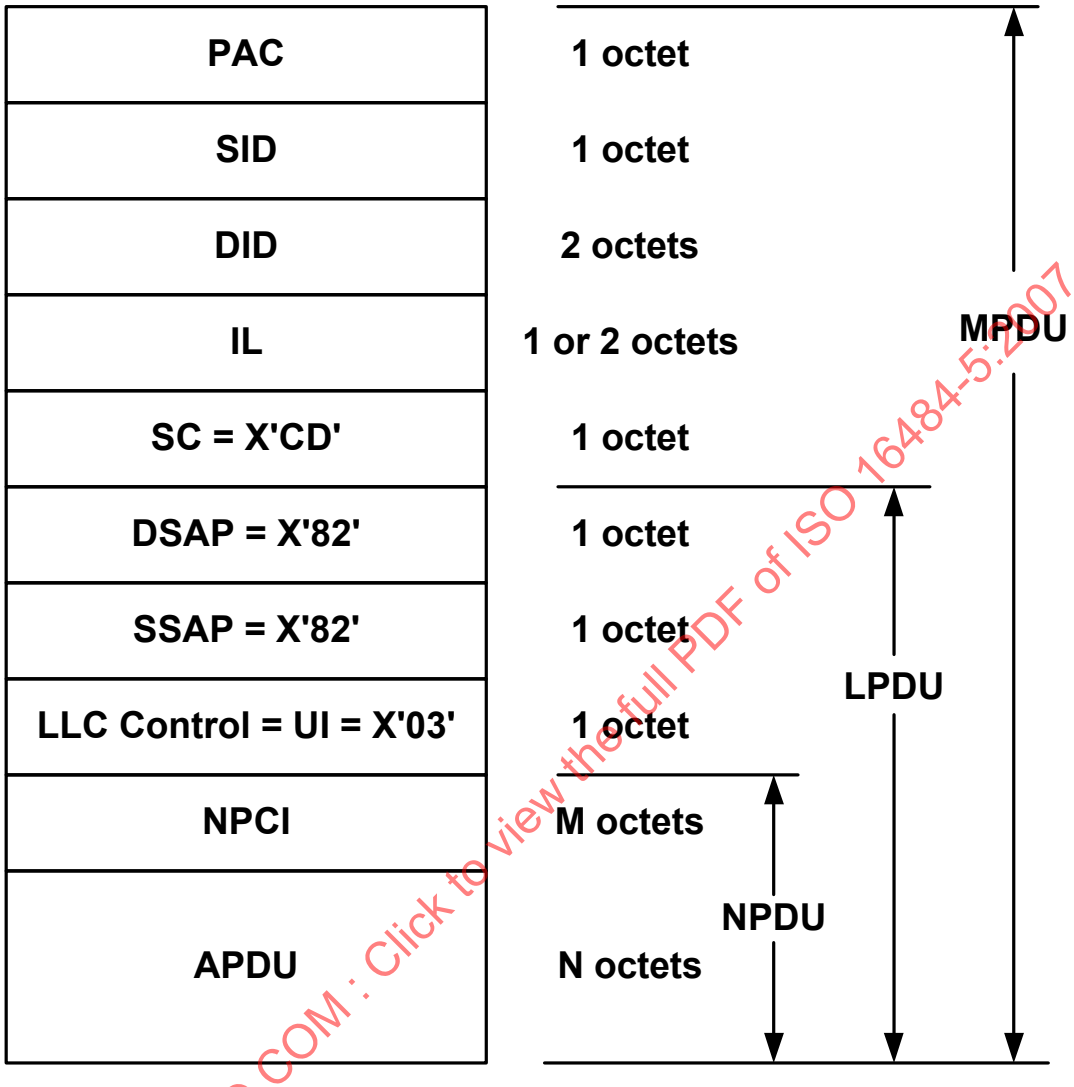


Figure 8-1. Format of an MPDU on an ARCNET LAN.

9 DATA LINK/PHYSICAL LAYERS: MASTER-SLAVE/TOKEN PASSING (MS/TP) LAN

This clause describes a Master-Slave/Token-Passing (MS/TP) data link protocol, which provides the same services to the network layer as ISO 8802-2 Logical Link Control. It uses services provided by the EIA-485 physical layer. Relevant clauses of EIA-485 are deemed to be included in this standard by reference. The following hardware is assumed:

(a) A UART (Universal Asynchronous Receiver/Transmitter) capable of transmitting and receiving eight data bits with one stop bit and no parity.

(b) An EIA-485 transceiver whose driver may be disabled.

(c) A timer with a resolution of five milliseconds or less.

9.1 Service Specification

MS/TP is not intended to be a general purpose LAN under ISO 8802-2. Instead, MS/TP includes a data link layer sufficient to provide to the BACnet network layer the same services as are offered by ISO 8802-2 Type 1.

This subclause describes the primitives and parameters associated with the provided services. The parameters are described in an abstract sense, which does not constrain the implementation method. Primitives and their parameters are described in a form that echoes their specification in ISO 8802-2. This is intended to provide a consistent interface to the BACnet network layer.

9.1.1 DL-UNITDATA.request

9.1.1.1 Function

This primitive is the service request primitive for the unacknowledged connectionless-mode data transfer service.

9.1.1.2 Semantics of the Service Primitive

The primitive shall provide parameters as follows:

```
DL-UNITDATA.request (
                        source_address,
                        destination_address,
                        data,
                        priority,
                        data_expecting_reply
                        )
```

Each source and destination address consists of the logical concatenation of a medium access control (MAC) address and a link service access point (LSAP). For the case of MS/TP devices, since MS/TP supports only the BACnet network layer, the LSAP is omitted and these parameters consist of only the device MAC address.

The 'data' parameter specifies the link service data unit (LSDU) to be transferred by the MS/TP entity.

The 'priority' parameter specifies the priority desired for the data unit transfer. The priority parameter is ignored by MS/TP.

The 'data_expecting_reply' parameter specifies whether or not the data unit to be transferred expects a reply.

9.1.1.3 When Generated

This primitive is passed from the network layer to the MS/TP entity to request that a network protocol data unit (NPDU) be sent to one or more remote LSAPs using unacknowledged connectionless-mode procedures.

9.1.1.4 Effect on Receipt

Receipt of this primitive causes the MS/TP entity to attempt to send the NPDU using unacknowledged connectionless-mode procedures.

9.1.2 DL-UNITDATA.indication

9.1.2.1 Function

This primitive is the service indication primitive for the unacknowledged connectionless-mode data transfer service.

9.1.2.2 Semantics of the Service Primitive

DL-UNITDATA.indication (
source_address,
destination_address,
data,
priority,
data_expecting_reply
)

Each source and destination address consists of the logical concatenation of a medium access control (MAC) address and a link service access point (LSAP). For the case of MS/TP devices, since MS/TP supports only the BACnet network layer, the LSAP is omitted and these parameters consist of only the device MAC address.

The 'data' parameter specifies the link service data unit that has been received by the MS/TP entity.

The 'priority' parameter specifies the priority desired for the data unit transfer. The priority parameter is ignored by MS/TP.

The 'data_expecting_reply' parameter specifies whether or not the data unit that has been received expects a reply.

9.1.2.3 When Generated

This primitive is passed from the MS/TP entity to the network layer to indicate the arrival of an NPDU from the specified remote entity.

9.1.2.4 Effect on Receipt

The effect of receipt of this primitive by the network layer is unspecified.

9.1.3 Test_Request and Test_Response

ISO 8802-2 Type 1 defines XID and TEST PDUs and procedures but does not define an interface to invoke them from the network layer. Test_Request and Test_Response PDUs and procedures have been defined for MS/TP to accomplish the same functions. Because MS/TP supports only the equivalent of a single LSAP, these PDUs are sufficient to implement the relevant aspects of XID as well.

The response with Test_Response to a received Test_Request PDU is mandatory for all MS/TP nodes. The origination of a Test_Request PDU is optional.

9.1.3.1 Use of Test_Request and Test_Response for ISO 8802-2 TEST Functions

The TEST function provides a facility to conduct loopback tests of the MS/TP to MS/TP transmission path. Successful completion of the test consists of sending a Test_Request PDU with a particular information field to the designated destination and receiving, in return, the identical information field in a Test_Response PDU.

If a receiving node can successfully receive and return the information field, it shall do so. If it cannot receive and return the entire information field but can detect the reception of a valid Test_Request frame (for example, by computing the CRC on octets as they are received), then the receiving node shall discard the information field and return a Test_Response containing no information field. If the receiving node cannot detect the valid reception of frames with overlength information fields, then no response shall be returned.

9.1.3.2 Use of Test_Request and Test_Response for ISO 8802-2 XID Functions

ISO 8802-2 describes seven possible uses of XID:

(a) XID can be used with a null DSAP and null SSAP as an "Are You There" test. Since MS/TP supports only the equivalent of a single LSAP, the Test_Request PDU with no data can perform this function.

(b) XID can be used with a group or global DSAP to identify group members or all active stations. Since MS/TP supports only the equivalent of a single LSAP, the Test_Request PDU with no data can perform this function.

(c) XID can be used for a duplicate address check. This function is not applicable to MS/TP. EIA-485 token bus networks such as MS/TP will generally not achieve reliable operation if multiple nodes exist with the same address, since collisions will occur during token passing.

(d) Class II LLCs may use XID to determine window size. MS/TP does not support Class II operation.

(e) XID may be used to identify the class of each LLC. Since MS/TP supports only Class I operation, this is a trivial operation.

(f) XID may be used to identify the service types supported by each LSAP. Since MS/TP supports only Class I operation, this is a trivial operation.

(g) An LLC can announce its presence by broadcasting an XID with global DSAP. Since MS/TP supports only one LSAP, the equivalent may be accomplished by broadcasting a Test_Response PDU.

9.2 Physical Layer

9.2.1 Medium

An MS/TP EIA-485 network shall use shielded, twisted-pair cable with characteristic impedance between 100 and 130 ohms. Distributed capacitance between conductors shall be less than 100 pF per meter (30 pF per foot). Distributed capacitance between conductors and shield shall be less that 200 pF per meter (60 pF per foot). Foil or braided shields are acceptable. The maximum recommended length of an MS/TP segment is 1200 meters (4000 feet) with AWG 18 (0.82 mm^2 conductor area) cable. The use of greater distances and/or different wire gauges shall comply with the electrical specifications of EIA-485.

9.2.2 Connections and Terminations

The maximum number of nodes per segment shall be 32 (as specified by the EIA-485 standard). Additional nodes may be accommodated by the use of repeaters, as described in 9.9.

Because MS/TP uses NRZ encoding, the polarity of the connection to the cable is important. The non-inverting input of the EIA-485 transceiver is designated in this specification as "plus" or "+" and the inverting input as "minus" or "-". It is recommended, but not required, that the black or red insulated wire of the twisted pair be designated as "plus" and the white, clear, or green insulated wire be designated as "minus." The method of connection between the interface and the cable is not part of this specification.

An MS/TP EIA-485 network shall have no T connections. A termination resistance of 120 ohms plus or minus 5% shall be connected at each of the two ends of the segment medium. No other termination resistors are allowed at intermediate nodes.

Each MS/TP segment shall be provided with network bias resistors, connected as shown in Figure 9-1, such that an undriven communications line will be held in a guaranteed logical one state. The bias provides a reliable way for stations to detect the presence or absence of signals on the line. An unbiased line will take an indeterminate state in the absence of any driving node. Under some conditions, noise or cross-talk might result in some nodes receiving spurious octets from the undriven idle line.

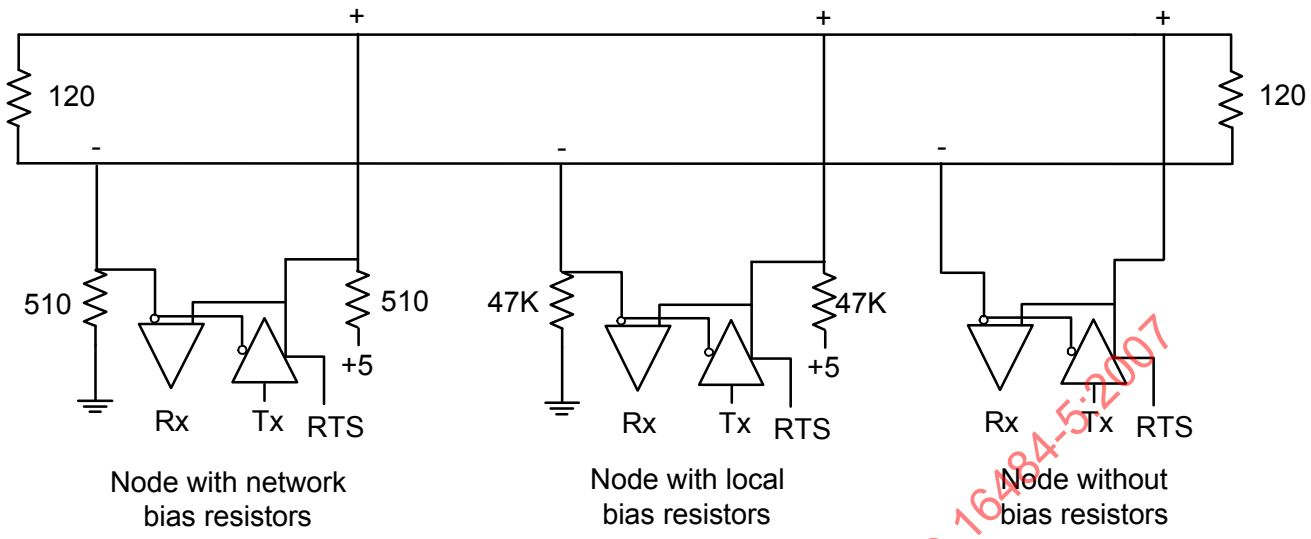


Figure 9-1. EIA-485 network showing three types of nodes.

At least one set, and no more than two sets, of network bias resistors shall exist for each segment. Each set of network bias resistors shall consist of two resistors, each having a value of 510 ohms, plus or minus 5%, connected as shown in Figure 9-1. If two sets of network bias resistors are provided, they shall be placed at two distinct nodes, preferably at the ends of the segment, so that proper bias levels can be maintained even if one of the bias nodes loses power. Other nodes may be provided with local bias resistors as long as each local bias resistor value is 47K ohms or greater. The use of local bias resistors is optional.

For any physical segment that runs between buildings there shall be at least 1500 V of electrical isolation between the EIA-485 signal conductors and the digital ground of any node on that physical segment.

The shield shall be grounded at one end only to prevent ground currents from being created.

9.2.3 Timing

Octets shall be transmitted using non-return to zero (NRZ) encoding with one start bit, eight data bits, no parity, and one stop bit. The start bit shall have a value of zero, while the stop bit shall have a value of one. The data bits shall be transmitted with the least significant bit first. This is illustrated in Figure 9-2.

Although asynchronous framing is used, there shall be no more than $\mathbf{T}_{\mathbf{frame\_gap}}$ of idle line (logical ones or stop bits) between any two octets of a frame.

The standard baud rate shall be 9600, plus or minus 1%. Any or all of the additional baud rates 19200, 38400, and 76800 may be supported at the vendor's option, but the 9600 baud shall be selectable.

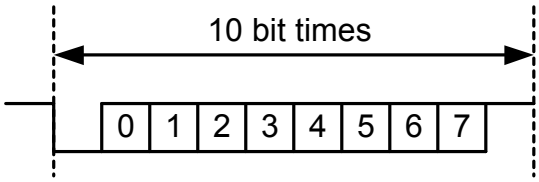


Figure 9-2. Octet framing.

Transmitter enable: A node shall enable its EIA-485 driver before it generates the leading edge of the first start bit of a frame. The node shall drive the line to the logical one state during the time between the enable and the leading edge of the first start bit of a frame.

Transmitter disable: A node shall not disable its EIA-485 driver until the stop bit of the final octet of a frame has been generated. The node shall disable its EIA-485 driver within $\mathbf{T}_{\mathbf{postdrive}}$ after the beginning of the stop bit of the final octet of a frame in order that it not interfere with any subsequent frame transmitted by another node. This specification allows, but does not encourage, the use of a "padding" octet after the final octet of a frame in order to facilitate the use of common UART transmit interrupts for driver disable control. If a "padding" octet is used, its value shall be X'FF'. The "padding" octet is not considered part of the frame, that is, it shall be included within $\mathbf{T}_{\mathbf{postdrive}}$.

Receive to Transmit turn-around: A node shall not enable its EIA-485 driver for at least $\mathbf{T}_{\mathbf{turnaround}}$ after the node receives the final stop bit of any octet.

9.3 MS/TP Frame Format

All frames are of the following format:

Preamble	two octet preamble: X'55', X'FF'
Frame Type	one octet
Destination Address	one octet address
Source Address	one octet address
Length	two octets, most significant octet first
Header CRC	one octet
Data	(present only if Length is non-zero)
Data CRC	(present only if Length is non-zero) two octets, least significant octet first
(pad)	(optional) at most one octet of padding: X'FF'

The Frame Type is used to distinguish between different types of MAC frames. Defined types are:

00	Token
01	Poll For Master
02	Reply To Poll For Master
03	Test_Request
04	Test_Response
05	BACnet Data Expecting Reply
06	BACnet Data Not Expecting Reply
07	Reply Postponed

Frame Types 8 through 127 are reserved by ASHRAE. Frame Types 128 through 255 are available to vendors for proprietary (non-BACnet) frames. Use of proprietary frames might allow a Brand-X controller, for example, to send proprietary frames to other Brand-X controllers that do not implement BACnet while using the same medium to send BACnet frames to a Brand-Y panel that does implement BACnet. Token, Poll For Master, and Reply To Poll For Master frames shall be understood by both proprietary and BACnet master nodes.

The Destination and Source Addresses are one octet each. A Destination Address of 255 (X'FF') denotes broadcast. A Source Address of 255 is not allowed. Addresses 0 to 127 are valid for both master and slave nodes. Addresses 128 to 254 are valid only for slave nodes.

The Length field specifies the length in octets of the Data field.

The Data and Data CRC fields are conditional on the Frame Type and the Length, as specified in the description of each Frame Type. If the Length field is zero, that is, if both length octets are zero, then the Data and Data CRC fields shall not be present.

The length of the Data field shall be between 0 and 501 octets.

Subclause 9.6 and Annex G describe in detail the generation and checking of the Header and Data CRC octets.

9.3.1 Frame Type 00: Token

The Token frame is used to pass network mastership to the destination node. The use of the Token frame is described in detail in 9.5.

There are no data octets in Token frames.

9.3.2 Frame Type 01: Poll For Master

The Poll For Master frame is transmitted by master nodes during configuration and periodically during normal network operation. It is used to discover the presence of other master nodes on the network and to determine a successor node in the token ring. The use of the Poll For Master frame in the token network is described in detail in 9.5.

There are no data octets in Poll For Master frames.

Both master and slave nodes shall expect to receive Poll For Master frames. Master nodes shall respond to Poll For Master Frames as described in 9.5.6.2. Slave nodes shall ignore Poll For Master frames, as described in 9.5.7.2.

9.3.3 Frame Type 02: Reply To Poll For Master

This frame is transmitted as a reply to the Poll For Master frame. It is used to indicate that the node sending the frame wishes to enter the token ring. The use of this frame in the token network is described in detail in 9.5.

There are no data octets in Reply To Poll For Master frames.

9.3.4 Frame Type 03: Test_Request

This frame is used to initiate a loopback test of the MS/TP to MS/TP transmission path. The use of this frame in the token network is described in detail in 9.1.3. The length of the data portion of a Test_Request frame may range from 0 to 501 octets.

9.3.5 Frame Type 04: Test_Response

This frame is used to reply to Test_Request frames. The use of this frame in the token network is described in detail in 9.1.3. The length of the data portion of a Test_Response frame may range from 0 to 501 octets. The data, if present, shall be that which was present in the initiating Test_Request.

9.3.6 Frame Type 05: BACnet Data Expecting Reply

This frame is used by master nodes to convey the data parameter of a DL_UNITDATA.request whose DER parameter is TRUE. The length of the data portion of a BACnet Data Expecting Reply frame may range from 0 to 501 octets.

9.3.7 Frame Type 06: BACnet Data Not Expecting Reply

This frame is used to convey the data parameter of a DL_UNITDATA.request whose DER parameter is FALSE. The length of the data portion of a BACnet Data Not Expecting Reply frame may range from 0 to 501 octets.

9.3.8 Frame Type 07: Reply Postponed

This frame is used by master nodes to defer sending a reply to a previously received BACnet Data Expecting Reply frame. The use of this frame in the token network is described in detail in 9.5.6.

There are no data octets in Reply Postponed frames.

9.3.9 Frame Types 128 through 255: Proprietary Frames

These frames are available to vendors as proprietary (non-BACnet) frames. The first two octets of the Data field shall specify the unique vendor identification code, most significant octet first, for the type of vendor-proprietary frame to be conveyed. The length of the data portion of a Proprietary frame shall be in the range of 2 to 501 octets.

9.4 Overview of the MS/TP Network

MS/TP uses a token to control access to a bus network. A master node may initiate the transmission of a data frame when it holds the token. Both master and slave nodes may transmit data frames in response to requests from master nodes. After sending at most $\mathbf{N}_{\mathbf{max\_info\_frames}}$ data frames (and awaiting any expected replies), a master node shall pass the token to the next master node.

It is generally easier to deal with a lost token than with the presence of two tokens in a ring: a simple timeout will detect token loss, and regeneration of the token and recovery of the ring may proceed in an orderly fashion. If more than one token exists, however, collisions are likely. These will disrupt communications and slow throughput but may not be severe enough to cause

loss of the tokens. In such a case, a persistent reduction in throughput might result. For this reason, the ring maintenance rules in this clause favor the loss of the token over the creation of a second token.

Token frames are not acknowledged. If the acknowledgment of a token were lost, the token's sender might retry, resulting in the creation of two tokens. Instead, after a node passes the token, it listens to see if the intended receiver node begins using the token. Usage in this case is defined as the reception of $\mathbf{N}_{min\_octets}$ octets from the network within $\mathbf{T}_{usage\_timeout}$ after the final octet of the token frame is transmitted.

Most token bus networks, such as ARCNET, do not distinguish between requests and replies: both are passed in the same type of frames, which are sent only when the sending node has the token. Since MS/TP defines slave nodes that never hold the token, a means must be provided to allow replies to be returned from slave devices. For simplicity, the same mechanism is used for replies returned from master nodes.

When a request that expects a reply is sent to an MS/TP node, the sender shall wait for the reply to be returned before passing the token. If the responding node is a master, it may return the reply or it may return a Reply Postponed frame, indicating that the actual reply will be returned later, when the replying node holds the token.

9.5 MS/TP Medium Access Control

The description that follows defines variables and procedures that may in some ways resemble the variables and procedures used in various computer languages. This description is in no way intended to prescribe the method of implementation. An implementation may be constructed in any fashion desired as long as it matches the behavior described by this standard. The description that follows is intended only to specify that behavior clearly and precisely.

9.5.1 UART Receiver Model

In this subclause, we present a model of the receiver interface to a UART as a data register and two Boolean flags. These are intended to closely resemble the functions of commercial UART chips but in a generic and non-prescriptive fashion. The model is used by the procedural and state machine descriptions.

9.5.1.1 DataRegister

The DataRegister holds the octet most recently received. The contents of this register after the occurrence of a framing or overrun error are not specified.

9.5.1.2 DataAvailable

The flag DataAvailable is TRUE if an octet is available in DataRegister. A means of setting this flag to FALSE when the associated data have been read from DataRegister shall be provided. Many common UARTs set DataAvailable FALSE automatically when DataRegister is read.

9.5.1.3 ReceiveError

The flag ReceiveError is TRUE if an error is detected during the reception of an octet. Many common UARTs detect several types of receive errors, in particular framing errors and overrun errors. ReceiveError shall be TRUE if any of these errors is detected.

A framing error occurs if a logical zero is received when a stop bit (logical one) is expected.

An overrun error occurs if an octet is received before an earlier octet is read from DataRegister. In general, the occurrence of overrun errors is evidence of improper design. However, it is recognized that critical system events may cause overrun errors to occur from time to time. The inclusion of this error in the state machine processing ensures that such errors are handled in a well-defined fashion.

A means of setting ReceiveError to FALSE when the associated error has been recognized shall be provided.

9.5.2 Variables

A number of variables and timers are used in the descriptions that follow:

DataCRC Used to accumulate the CRC on the data field of a frame.

DataLength Used to store the data length of a received frame.

DestinationAddress Used to store the destination address of a received frame.

EventCount Used to count the number of received octets or errors. This is used in the detection of link activity.

FrameType Used to store the frame type of a received frame.

FrameCount The number of frames sent by this node during a single token hold. When this counter reaches the value $\mathbf{N}_{\mathbf{max\_info\_frames}}$, the node must pass the token.

HeaderCRC Used to accumulate the CRC on the header of a frame.

Index Used as an index by the Receive State Machine, up to a maximum value of InputBufferSize.

InputBuffer[] An array of octets, used to store octets as they are received. InputBuffer is indexed from 0 to InputBufferSize-1. The maximum size of a frame is 501 octets. A smaller value for InputBufferSize may be used by some implementations.

InputBufferSize The number of elements in the array InputBuffer[].

NS "Next Station," the MAC address of the node to which This Station passes the token. If the Next Station is unknown, NS shall be equal to TS.

PS "Poll Station," the MAC address of the node to which This Station last sent a Poll For Master. This is used during token maintenance.

ReceivedInvalidFrame A Boolean flag set to TRUE by the Receive State Machine if an error is detected during the reception of a frame. Set to FALSE by the main state machine.

ReceivedValidFrame A Boolean flag set to TRUE by the Receive State Machine if a valid frame is received. Set to FALSE by the main state machine.

RetryCount A counter of transmission retries used for Token and Poll For Master transmission.

SilenceTimer A timer with nominal 5 millisecond resolution used to measure and generate silence on the medium between octets. It is incremented by a timer process and is cleared by the Receive State Machine when activity is detected and by the SendFrame procedure as each octet is transmitted. Since the timer resolution is limited and the timer is not necessarily synchronized to other machine events, a timer value of N will actually denote intervals between N-1 and N.

SoleMaster A Boolean flag set to TRUE by the master machine if this node is the only known master node.

SourceAddress Used to store the Source Address of a received frame.

TokenCount The number of tokens received by this node. When this counter reaches the value $\mathbf{N}_{\mathbf{poll}}$, the node polls the address range between **TS** and **NS** for additional master nodes. **TokenCount** is set to one at the end of the polling process.

TS "This Station," the MAC address of this node. TS is generally read from a hardware DIP switch, or from nonvolatile memory. Valid values for TS are 0 to 254. The value 255 is used to denote broadcast when used as a destination address but is not allowed as a value for TS.

9.5.3 Parameters

Parameter values used in the description:

$N_{max\_info\_frames}$	This parameter represents the value of the Max_Info_Frames property of the node's Device object. The value of Max_Info_Frames specifies the maximum number of information frames the node may send before it must pass the token. Max_Info_Frames may have different values on different nodes. This may be used to allocate more or less of the available link bandwidth to particular nodes. If Max_Info_Frames is not writable in a node, its value shall be 1.
N_{max_master}	This parameter represents the value of the Max_Master property of the node's Device object. The value of Max_Master specifies the highest allowable address for master nodes. The value of Max_Master shall be less than or equal to 127. If Max_Master is not writable in a node, its value shall be 127.
N_{poll}	The number of tokens received or used before a Poll For Master cycle is executed: 50.
N_{retry_token}	The number of retries on sending Token: 1.
N_{min_octets}	The minimum number of DataAvailable or ReceiveError events that must be seen by a receiving node in order to declare the line "active": 4.
T_{frame_abort}	The minimum time without a DataAvailable or ReceiveError event within a frame before a receiving node may discard the frame: 60 bit times. (Implementations may use larger values for this timeout, not to exceed 100 milliseconds.)
T_{frame_gap}	The maximum idle time a sending node may allow to elapse between octets of a frame the node is transmitting: 20 bit times.
T_{no_token}	The time without a DataAvailable or ReceiveError event before declaration of loss of token: 500 milliseconds.
$T_{postdrive}$	The maximum time after the end of the stop bit of the final octet of a transmitted frame before a node must disable its EIA-485 driver: 15 bit times.
T_{reply_delay}	The maximum time a node may wait after reception of a frame that expects a reply before sending the first octet of a reply or Reply Postponed frame: 250 milliseconds.
$T_{reply\_timeout}$	The minimum time without a DataAvailable or ReceiveError event that a node must wait for a station to begin replying to a confirmed request: 255 milliseconds. (Implementations may use larger values for this timeout, not to exceed 300 milliseconds.)
T_{roff}	Repeater turnoff delay. The duration of a continuous logical one state at the active input port of an MS/TP repeater after which the repeater will enter the IDLE state: 29 bit times < T_{roff} < 40 bit times.
T_{slot}	The width of the time slot within which a node may generate a token: 10 milliseconds.
$T_{turnaround}$	The minimum time after the end of the stop bit of the final octet of a received frame before a node may enable its EIA-485 driver: 40 bit times.
T_{usage_delay}	The maximum time a node may wait after reception of the token or a Poll For Master frame before sending the first octet of a frame: 15 milliseconds.
$T_{usage\_timeout}$	The minimum time without a DataAvailable or ReceiveError event that a node must wait for a remote node to begin using a token or replying to a Poll For Master frame: 20 milliseconds. (Implementations may use larger values for this timeout, not to exceed 100 milliseconds.)

9.5.4 Receive Frame Finite State Machine

This section describes the reception of an MS/TP frame by a BACnet device. The description of operation is as a finite state machine. Figure 9-3 shows the Receive Frame state machine, which is described fully in this clause. Each state is given a name, specified in all capital letters. Transitions are also named, in mixed upper- and lowercase letters. Transitions are

described as a series of conditions followed by a series of actions to be taken if the conditions are met. The final action in each transition is entry into a new state, which may be the same as the current state.

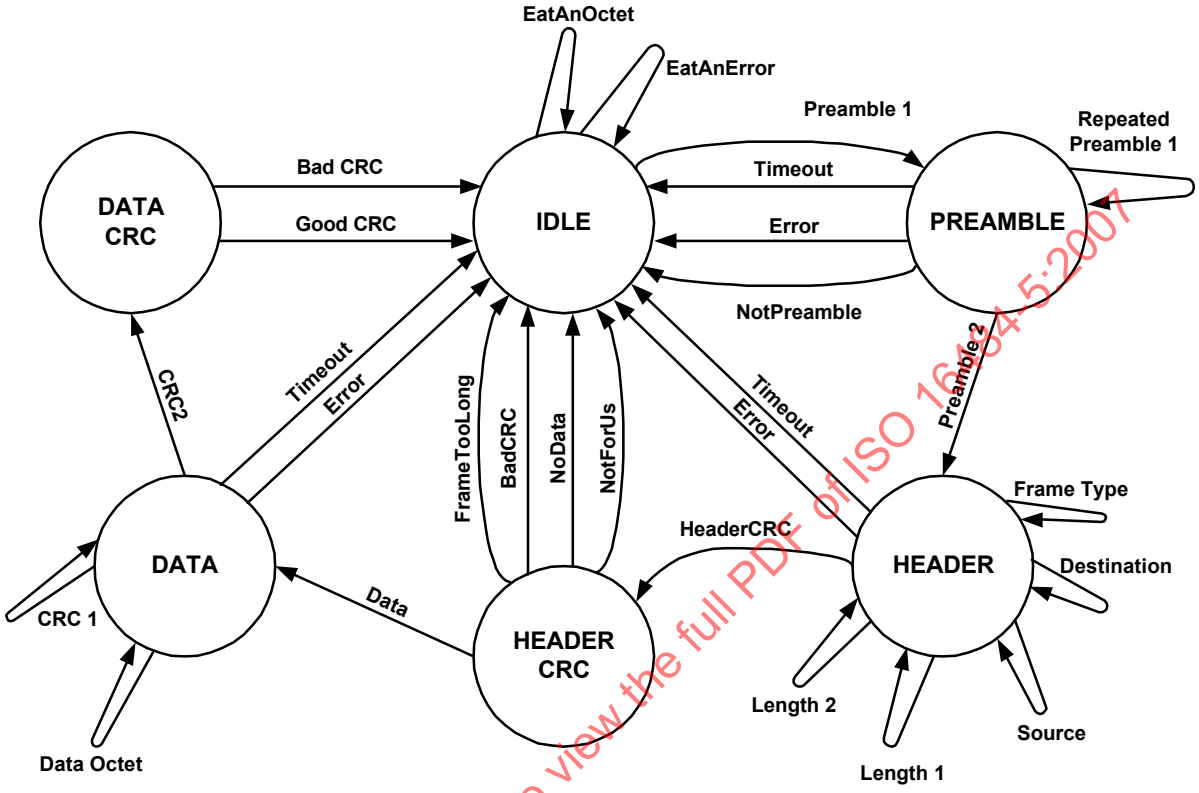


Figure 9-3. Receive Frame State Machine.

The Receive Frame state machine operates independently from the MS/TP Master Node or Slave Node machine, communicating with it by means of flags and other variables. The description assumes that the Master Node or Slave Node state machine can process received frames and other indications from the Receive Frame state machine before the next frame begins. The means by which this behavior is implemented are a local matter.

This description assumes that the node will not receive its own transmissions. If a given implementation does receive its own transmissions, then the implementation shall be constructed so that the Receive Frame machine will ignore the transmissions.

9.5.4.1 IDLE

In the IDLE state, the node waits for the beginning of a frame.

EatAnError

If ReceiveError is TRUE,

then set ReceiveError to FALSE; set SilenceTimer to zero; increment EventCount; and enter the IDLE state to wait for the start of a frame.

EatAnOctet

If ReceiveError is FALSE and DataAvailable is TRUE and the content of DataRegister is not X'55',

then set DataAvailable to FALSE; set SilenceTimer to zero; increment EventCount; and enter the IDLE state to wait for the start of a frame.

Preamble1

If ReceiveError is FALSE and DataAvailable is TRUE and the content of DataRegister is X '55',

then set DataAvailable to FALSE; set SilenceTimer to zero; increment EventCount; and enter the PREAMBLE state to receive the remainder of the frame.

9.5.4.2 PREAMBLE

In the PREAMBLE state, the node waits for the second octet of the preamble.

Timeout

If SilenceTimer is greater than $\mathbf{T}_{\text{frame\_abort}}$,

then a correct preamble has not been received. Enter the IDLE state to wait for the start of a frame.

Error

If ReceiveError is TRUE,

then set ReceiveError to FALSE; set SilenceTimer to zero; increment EventCount; and enter the IDLE state to wait for the start of a frame.

RepeatedPreamble1

If ReceiveError is FALSE and DataAvailable is TRUE and the content of DataRegister is X '55',

then set DataAvailable to FALSE; set SilenceTimer to zero; increment EventCount; and enter the PREAMBLE state to wait for the second preamble octet.

NotPreamble

If ReceiveError is FALSE and DataAvailable is TRUE and the content of DataRegister is not X 'FF' or X '55',

then set DataAvailable to FALSE; set SilenceTimer to zero; increment EventCount; and enter the IDLE state to wait for the start of a frame.

Preamble2

If ReceiveError is FALSE and DataAvailable is TRUE and the content of DataRegister is X 'FF',

then set DataAvailable to FALSE; set SilenceTimer to zero; increment EventCount; set Index to zero; set HeaderCRC to X 'FF'; and enter the HEADER state to receive the remainder of the frame.

9.5.4.3 HEADER

In the HEADER state, the node waits for the fixed message header.

Timeout

If SilenceTimer is greater than $\mathbf{T}_{\text{frame\_abort}}$,

then set ReceivedInvalidFrame to TRUE to indicate that an error has occurred during the reception of a frame, and enter the IDLE state to wait for the start of a frame.

Error

If ReceiveError is TRUE,

then set ReceiveError to FALSE; set SilenceTimer to zero; increment EventCount; set ReceivedInvalidFrame to TRUE to indicate that an error has occurred during the reception of a frame; and enter the IDLE state to wait for the start of a frame.

FrameType

If ReceiveError is FALSE and DataAvailable is TRUE and Index is 0,

then set DataAvailable to FALSE; set SilenceTimer to zero; increment EventCount; accumulate the contents of DataRegister into HeaderCRC; save the contents of DataRegister as FrameType; set Index to 1; and enter the HEADER state.

Destination

If ReceiveError is FALSE and DataAvailable is TRUE and Index is 1

then set DataAvailable to FALSE; set SilenceTimer to zero; increment EventCount; accumulate the contents of DataRegister into HeaderCRC; save the contents of DataRegister as DestinationAddress; set Index to 2; and enter the HEADER state.

Source

If ReceiveError is FALSE and DataAvailable is TRUE and Index is 2,

then set DataAvailable to FALSE; set SilenceTimer to zero; increment EventCount; accumulate the contents of DataRegister into HeaderCRC; save the contents of DataRegister as SourceAddress; set Index to 3; and enter the HEADER state.

Length1

If ReceiveError is FALSE and DataAvailable is TRUE and Index is 3,

then set DataAvailable to FALSE; set SilenceTimer to zero; increment EventCount; accumulate the contents of DataRegister into HeaderCRC; multiply the contents of DataRegister by 256 and save the result as DataLength; set Index to 4; and enter the HEADER state.

Length2

If ReceiveError is FALSE and DataAvailable is TRUE and Index is 4,

then set DataAvailable to FALSE; set SilenceTimer to zero; increment EventCount; accumulate the contents of DataRegister into HeaderCRC; add the contents of DataRegister to DataLength and save the result as DataLength; set Index to 5; and enter the HEADER state.

HeaderCRC

If ReceiveError is FALSE and DataAvailable is TRUE and Index is 5,

then set DataAvailable to FALSE; set SilenceTimer to zero; increment EventCount; accumulate the contents of DataRegister into HeaderCRC; and enter the HEADER_CRC state.

9.5.4.4 HEADER_CRC

In the HEADER_CRC state, the node validates the CRC on the fixed message header.

BadCRC

If the value of HeaderCRC is not X '55',

then set ReceivedInvalidFrame to TRUE to indicate that an error has occurred during the reception of a frame, and enter the IDLE state to wait for the start of the next frame.

NotForUs

If the value of the HeaderCRC is X '55' and the value of DestinationAddress is not equal to either TS (this station) or 255 (broadcast),

then enter the IDLE state to wait for the start of the next frame.

FrameTooLong

If the value of the HeaderCRC is X '55' and the value of DestinationAddress is equal to either TS (this station) or 255 (broadcast) and DataLength is greater than InputBufferSize,

then set ReceivedInvalidFrame to TRUE to indicate that a frame with an illegal or unacceptable data length has been received, and enter the IDLE state to wait for the start of the next frame.

NoData

If the value of HeaderCRC is X '55' and the value of DestinationAddress is equal to either TS (this station) or 255 (broadcast) and DataLength is zero,

then set ReceivedValidFrame to TRUE to indicate that a frame with no data has been received, and enter the IDLE state to wait for the start of the next frame.

Data

If the value of HeaderCRC is X'55' and the value of DestinationAddress is equal to either TS (this station) or 255 (broadcast) and DataLength is not zero and DataLength is less than or equal to InputBufferSize,

then set Index to zero; set DataCRC to X'FFFF'; and enter the DATA state to receive the data portion of the frame.

9.5.4.5 DATA

In the DATA state, the node waits for the data portion of a frame.

Timeout

If SilenceTimer is greater than $\mathbf{T}_{\text{frame\_abort}}$,

then set ReceivedInvalidFrame to TRUE to indicate that an error has occurred during the reception of a frame, and enter the IDLE state to wait for the start of the next frame.

Error

If ReceiveError is TRUE,

then set ReceiveError to FALSE; set SilenceTimer to zero; set ReceivedInvalidFrame to TRUE to indicate that an error has occurred during the reception of a frame; and enter the IDLE state to wait for the start of the next frame.

DataOctet

If ReceiveError is FALSE and DataAvailable is TRUE and Index is less than DataLength,

then set DataAvailable to FALSE; set SilenceTimer to zero; accumulate the contents of DataRegister into DataCRC; save the contents of DataRegister at InputBuffer[Index]; increment Index by 1; and enter the DATA state.

CRC1

If ReceiveError is FALSE and DataAvailable is TRUE and Index is equal to DataLength,

then set DataAvailable to FALSE; set SilenceTimer to zero; accumulate the contents of DataRegister into DataCRC; increment Index by 1; and enter the DATA state.

CRC2

If ReceiveError is FALSE and DataAvailable is TRUE and Index is equal to DataLength plus 1,

then set DataAvailable to FALSE; set SilenceTimer to zero; accumulate the contents of DataRegister into DataCRC; and enter the DATA_CRC state.

9.5.4.6 DATA_CRC

In the DATA_CRC state, the node validates the CRC of the message data.

BadCRC

If the value of DataCRC is not X'F0B8',

then set ReceivedInvalidFrame to TRUE to indicate that an error has occurred during the reception of a frame, and enter the IDLE state to wait for the start of the next frame.

GoodCRC

If the value of DataCRC is X'F0B8',

then set ReceivedValidFrame to TRUE to indicate the complete reception of a valid frame, and enter the IDLE state to wait for the start of the next frame.

9.5.5 The SendFrame Procedure

The transmission of an MS/TP frame proceeds as follows:

Procedure SendFrame

(a) If SilenceTimer is less than $\mathbf{T}_{\mathbf{turnaround}}$, wait ($\mathbf{T}_{\mathbf{turnaround}}$ - SilenceTimer) in order to avoid line contention.

(b) Disable the receiver, and enable the transmit line driver.

(c) Transmit the preamble octets X'55', X'FF'. As each octet is transmitted, set SilenceTimer to zero.

(d) Initialize HeaderCRC to X'FF'.

(e) Transmit the Frame Type, Destination Address, Source Address, and Data Length octets. Accumulate each octet into HeaderCRC. As each octet is transmitted, set SilenceTimer to zero.

(f) Transmit the ones-complement of HeaderCRC. Set SilenceTimer to zero.

(g) If there are data octets, initialize DataCRC to X'FFFF'.

(h) Transmit any data octets. Accumulate each octet into DataCRC. As each octet is transmitted, set SilenceTimer to zero.

(i) Transmit the ones-complement of DataCRC, least significant octet first. As each octet is transmitted, set SilenceTimer to zero.

(j) Wait until the final stop bit of the most significant CRC octet has been transmitted but not more than $\mathbf{T}_{\mathbf{postdrive}}$.

(k) Disable the transmit line driver.

(l) Return.

9.5.6 Master Node Finite State Machine

The description of operation is as a finite state machine. Figure 9-4 shows the Master Node state machine, which is described fully in this clause. Each state is given a name, specified in all capital letters. Transitions are also named, in mixed upper- and lowercase letters. Transitions are described as a series of conditions followed by a series of actions to be taken if the conditions are met. The final action in each transition is entry into a new state, which may be the same as the current state.

A master node that supports segmentation shall not use a segmentation window size greater than one.

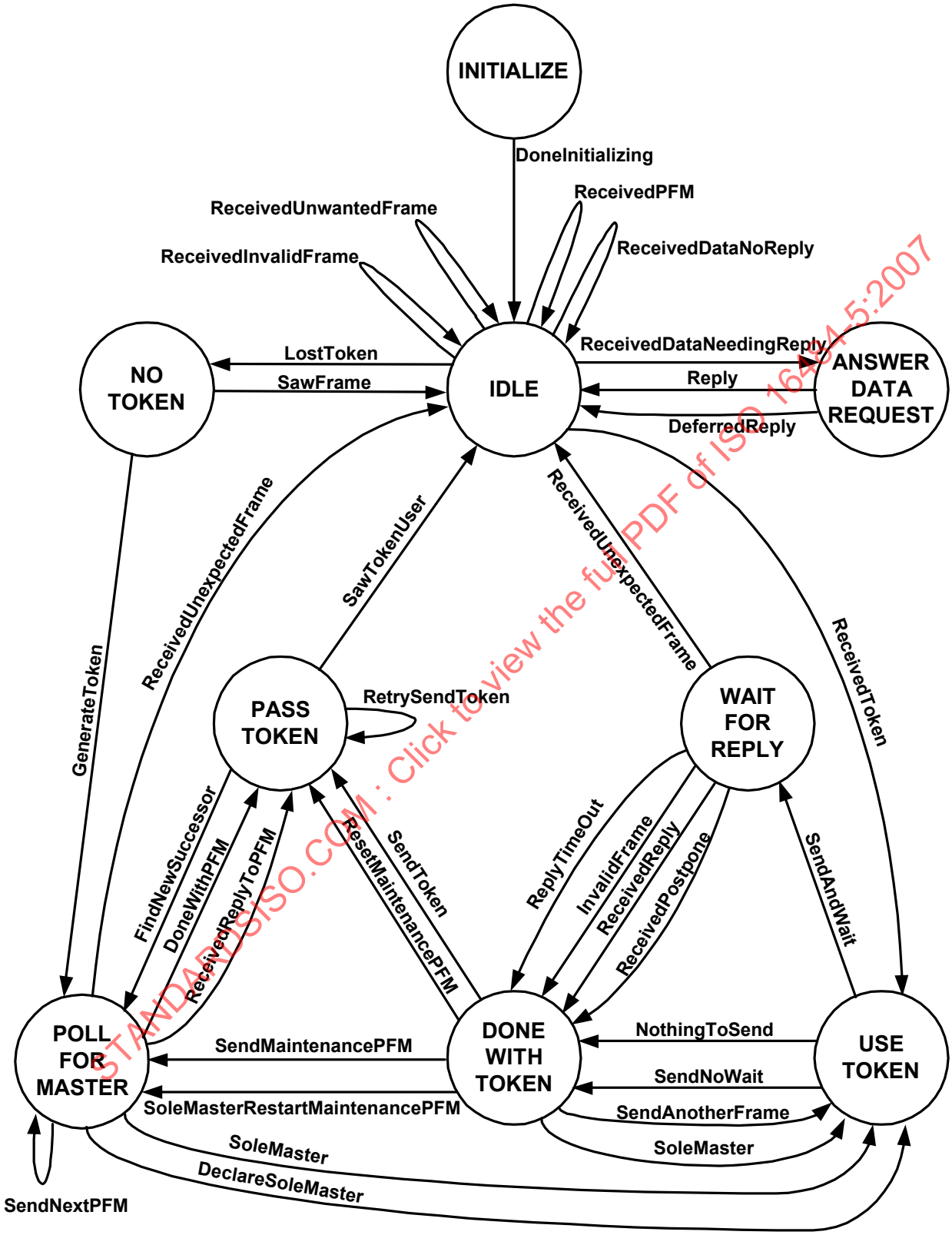


Figure 9-4. Master Node State Machine.

9.5.6.1 INITIALIZE

When a master node is powered up or reset, it shall unconditionally enter the INITIALIZE state.

DoneInitializing
Unconditionally,

set TS to the node's station address, set NS equal to TS (indicating that the next station is unknown), set PS equal to TS, set TokenCount to $\mathbf{N_{poll}}$ (thus causing a Poll For Master to be sent when this node first receives the token), set SoleMaster to FALSE, set ReceivedValidFrame and ReceivedInvalidFrame to FALSE, and enter the IDLE state.

9.5.6.2 IDLE

In the IDLE state, the node waits for a frame.

LostToken
If SilenceTimer is greater than or equal to $\mathbf{T_{no\_token}}$,

then assume that the token has been lost. Enter the NO_TOKEN state.

ReceivedInvalidFrame
If ReceivedInvalidFrame is TRUE,

then an invalid frame was received. Set ReceivedInvalidFrame to FALSE, and enter the IDLE state to wait for the next frame.

ReceivedUnwantedFrame
If ReceivedValidFrame is TRUE and either

(a) DestinationAddress is not equal to either TS (this station) or 255 (broadcast) or

(b) DestinationAddress is equal to 255 (broadcast) and FrameType has a value of Token, BACnet Data Expecting Reply, Test_Request, or a proprietary type known to this node that expects a reply (such frames may not be broadcast) or

(c) FrameType has a value that indicates a standard or proprietary type that is not known to this node,

then an unexpected or unwanted frame was received. Set ReceivedValidFrame to FALSE, and enter the IDLE state to wait for the next frame.

ReceivedToken
If ReceivedValidFrame is TRUE and DestinationAddress is equal to TS (this station) and FrameType is equal to Token,

then set ReceivedValidFrame to FALSE; set FrameCount to zero; set SoleMaster to FALSE; and enter the USE_TOKEN state.

ReceivedPFM
If ReceivedValidFrame is TRUE and DestinationAddress is equal to TS (this station) and FrameType is equal to Poll For Master,

then call SendFrame to transmit a Reply To Poll For Master frame to the node whose address is specified by SourceAddress (Source Address of the Poll); set ReceivedValidFrame to FALSE; and enter the IDLE state.

ReceivedDataNoReply
If ReceivedValidFrame is TRUE and DestinationAddress is equal to either TS (this station) or 255 (broadcast) and FrameType is equal to BACnet Data Not Expecting Reply, Test_Response, or a proprietary type known to this node that does not expect a reply,

then indicate successful reception to the higher layers; set ReceivedValidFrame to FALSE; and enter the IDLE state.

ReceivedDataNeedingReply

If ReceivedValidFrame is TRUE and DestinationAddress is equal to TS (this station) and FrameType is equal to BACnet Data Expecting Reply, Test Request, or a proprietary type known to this node that expects a reply,

then indicate successful reception to the higher layers (management entity in the case of Test_Request); set ReceivedValidFrame to FALSE; and enter the ANSWER_DATA_REQUEST state.

9.5.6.3 USE_TOKEN

In the USE_TOKEN state, the node is allowed to send one or more data frames. These may be BACnet Data frames or proprietary frames.

NothingToSend

If there is no data frame awaiting transmission,

then set FrameCount to $\mathbf{N}_{max\_info\_frames}$ and enter the DONE_WITH_TOKEN state.

SendNoWait

If the next frame awaiting transmission that is of type Test_Response, BACnet Data Not Expecting Reply, or a proprietary type that does not expect a reply,

then call SendFrame to transmit the frame; increment FrameCount; and enter the DONE_WITH_TOKEN state.

SendAndWait

If the next frame awaiting transmission that is of type Test_Request, BACnet Data Expecting Reply, or a proprietary type that expects a reply,

then call SendFrame to transmit the data frame; increment FrameCount; and enter the WAIT_FOR_REPLY state.

9.5.6.4 WAIT_FOR_REPLY

In the WAIT_FOR_REPLY state, the node waits for a reply from another node.

ReplyTimeout

If SilenceTimer is greater than or equal to $\mathbf{T}_{reply\_timeout}$,

then assume that the request has failed. Set FrameCount to $\mathbf{N}_{max\_info\_frames}$ and enter the DONE_WITH_TOKEN state. Any retry of the data frame shall await the next entry to the USE_TOKEN state. (Because of the length of the timeout, this transition will cause the token to be passed regardless of the initial value of FrameCount.)

InvalidFrame

If SilenceTimer is less than $\mathbf{T}_{reply\_timeout}$ and ReceivedInvalidFrame is TRUE,

then there was an error in frame reception. Set ReceivedInvalidFrame to FALSE and enter the DONE_WITH_TOKEN state.

ReceivedReply

If SilenceTimer is less than $\mathbf{T}_{reply\_timeout}$ and ReceivedValidFrame is TRUE and DestinationAddress is equal to TS (this station) and FrameType is equal to Test_Response, BACnet Data Not Expecting Reply, or a proprietary type that indicates a reply,

then indicate successful reception to the higher layers; set ReceivedValidFrame to FALSE; and enter the DONE_WITH_TOKEN state.

ReceivedPostpone

If SilenceTimer is less than $\mathbf{T}_{reply\_timeout}$ and ReceivedValidFrame is TRUE and DestinationAddress is equal to TS (this station) and FrameType is equal to Reply Postponed,

then the reply to the message has been postponed until a later time. Set ReceivedValidFrame to FALSE and enter the DONE_WITH_TOKEN state.

ReceivedUnexpectedFrame

If SilenceTimer is less than $\mathbf{T}_{reply\_timeout}$ and ReceivedValidFrame is TRUE and either

(a) DestinationAddress is not equal to TS (the expected reply should not be broadcast) or

(b) FrameType has a value other than Test_Response, BACnet Data Not Expecting Reply, or proprietary reply frame,

then an unexpected frame was received. This may indicate the presence of multiple tokens. Set ReceivedValidFrame to FALSE, and enter the IDLE state to synchronize with the network. This action drops the token.

9.5.6.5 DONE_WITH_TOKEN

The DONE_WITH_TOKEN state either sends another data frame, passes the token, or initiates a Poll For Master cycle.

SendAnotherFrame

If FrameCount is less than $\mathbf{N}_{max\_info\_frames}$,

then this node may send another information frame before passing the token. Enter the USE_TOKEN state.

SoleMaster

If FrameCount is greater than or equal to $\mathbf{N}_{max\_info\_frames}$ and TokenCount is less than $\mathbf{N}_{poll}$–1 and SoleMaster is TRUE,

then there are no other known master nodes to which the token may be sent (true master-slave operation). Set FrameCount to zero, increment TokenCount, and enter the USE_TOKEN state.

SendToken

If FrameCount is greater than or equal to $\mathbf{N}_{max\_info\_frames}$ and TokenCount is less than $\mathbf{N}_{poll}$–1 and SoleMaster is FALSE, or if NS is equal to (TS+1) modulo ($\mathbf{N}_{max\_master}$+1),

then increment TokenCount; call SendFrame to transmit a Token frame to NS; set RetryCount and EventCount to zero; and enter the PASS_TOKEN state. (The comparison of NS and TS+1 eliminates the Poll For Master if there are no addresses between TS and NS, since there is no address at which a new master node may be found in that case).

SendMaintenancePFM

If FrameCount is greater than or equal to $\mathbf{N}_{max\_info\_frames}$ and TokenCount is greater than or equal to $\mathbf{N}_{poll}$–1 and (PS+1) modulo ($\mathbf{N}_{max\_master}$+1) is not equal to NS,

then set PS to (PS+1) modulo ($\mathbf{N}_{max\_master}$+1); call SendFrame to transmit a Poll For Master frame to PS; set RetryCount to zero; and enter the POLL_FOR_MASTER state.

ResetMaintenancePFM

If FrameCount is greater than or equal to $\mathbf{N}_{max\_info\_frames}$ and TokenCount is greater than or equal to $\mathbf{N}_{poll}$–1 and (PS+1) modulo ($\mathbf{N}_{max\_master}$+1) is equal to NS, and SoleMaster is FALSE,

then set PS to TS; call SendFrame to transmit a Token frame to NS; set RetryCount and EventCount to zero; set TokenCount to one; and enter the PASS_TOKEN state.

SoleMasterRestartMaintenancePFM

If FrameCount is greater than or equal to $\mathbf{N}_{max\_info\_frames}$, TokenCount is greater than or equal to $\mathbf{N}_{poll}$, (PS+1) modulo ($\mathbf{N}_{max\_master}$+1) is equal to NS, and SoleMaster is TRUE,

then set PS to (NS +1) modulo ($\mathbf{N}_{max\_master}$+1) ; call SendFrame to transmit a Poll For Master to PS; set NS to TS (no known successor node); set RetryCount and EventCount to zero; set TokenCount to one; and enter the POLL_FOR_MASTER state to find a new successor to TS.

9.5.6.6 PASS_TOKEN

The PASS_TOKEN state listens for a successor to begin using the token that this node has just attempted to pass.

SawTokenUser

If SilenceTimer is less than $\mathbf{T}_{usage\_timeout}$ and EventCount is greater than $\mathbf{N}_{min\_octets}$,

then assume that a frame has been sent by the new token user. Enter the IDLE state to process the frame.

RetrySendToken

If SilenceTimer is greater than or equal to $\mathbf{T}_{usage\_timeout}$ and RetryCount is less than $\mathbf{N}_{retry\_token}$,

then increment RetryCount; call SendFrame to transmit a Token frame to NS; set EventCount to zero; and re-enter the current state to listen for NS to begin using the token.

FindNewSuccessor

If SilenceTimer is greater than or equal to $\mathbf{T}_{usage\_timeout}$ and RetryCount is greater than or equal to $\mathbf{N}_{retry\_token}$,

then assume that NS has failed. Set PS to (NS+1) modulo ($\mathbf{N}_{max\_master}$+1); call SendFrame to transmit a Poll For Master frame to PS; set NS to TS (no known successor node); set RetryCount, TokenCount, and EventCount to zero; and enter the POLL_FOR_MASTER state to find a new successor to TS.

9.5.6.7 NO_TOKEN

The NO_TOKEN state is entered if SilenceTimer becomes greater than $\mathbf{T}_{no\_token}$, indicating that there has been no network activity for that period of time. The timeout is continued to determine whether or not this node may create a token.

SawFrame

If SilenceTimer is less than $\mathbf{T}_{no\_token}$+($\mathbf{T}_{slot}$*TS) and EventCount is greater than $\mathbf{N}_{min\_octets}$,

then some other node exists at a lower address. Enter the IDLE state to receive and process the incoming frame.

GenerateToken

If SilenceTimer is greater than or equal to $\mathbf{T}_{no\_token}$+($\mathbf{T}_{slot}$*TS) and SilenceTimer is less than $\mathbf{T}_{no\_token}$+($\mathbf{T}_{slot}$*(TS+1)),

then assume that this node is the lowest numerical address on the network and is empowered to create a token. Set PS to (TS+1) modulo ($\mathbf{N}_{max\_master}$+1); call SendFrame to transmit a Poll For Master frame to PS; set NS to TS (indicating that the next station is unknown); set TokenCount, RetryCount, and EventCount to zero; and enter the POLL_FOR_MASTER state to find a new successor to TS.

9.5.6.8 POLL_FOR_MASTER

In the POLL_FOR_MASTER state, the node listens for a reply to a previously sent Poll For Master frame in order to find a successor node.

ReceivedReplyToPFM

If ReceivedValidFrame is TRUE and DestinationAddress is equal to TS (this station) and FrameType is equal to Reply To Poll For Master,

then set SoleMaster to FALSE; set NS equal to SourceAddress; set EventCount to zero; call SendFrame to transmit a Token frame to NS; set PS to the value of TS; set TokenCount and RetryCount to zero; set ReceivedValidFrame to FALSE; and enter the PASS_TOKEN state.

ReceivedUnexpectedFrame

If ReceivedValidFrame is TRUE and either

(a) DestinationAddress is not equal to TS or

(b) FrameType is not equal to Reply To Poll For Master,

then an unexpected frame was received. This may indicate the presence of multiple tokens. Set ReceivedValidFrame to FALSE and enter the IDLE state to synchronize with the network. This action drops the token.

SoleMaster

If SoleMaster is TRUE and either

(a) SilenceTimer is greater than or equal to $\mathbf{T}_{usage\_timeout}$ or

(b) ReceivedInvalidFrame is TRUE,

then there was no valid reply to the periodic poll by the sole known master for other masters. Set FrameCount to zero, set ReceivedInvalidFrame to FALSE, and enter the USE_TOKEN state.

DoneWithPFM

If SoleMaster is FALSE and NS is not equal to TS and either:

(a) SilenceTimer is greater than or equal to $\mathbf{T}_{usage\_timeout}$ or

(b) ReceivedInvalidFrame is TRUE,

then there was no valid reply to the maintenance poll for a master at address PS. Set EventCount to zero; call SendFrame to transmit a Token frame to NS; set RetryCount to zero; set ReceivedInvalidFrame to FALSE; and enter the PASS_TOKEN state.

SendNextPFM

If SoleMaster is FALSE and NS is equal to TS (no known successor node) and (PS+1) modulo ($\mathbf{N}_{max\_master}$+1) is not equal to TS and either:

(a) SilenceTimer greater than or equal to $\mathbf{T}_{usage\_timeout}$ or

(b) ReceivedInvalidFrame is TRUE,

then set PS to (PS+1) modulo ($\mathbf{N}_{max\_master}$+1); call SendFrame to transmit a Poll For Master frame to PS; set RetryCount to zero; set ReceivedInvalidFrame to FALSE; and re-enter the current state.

DeclareSoleMaster

If SoleMaster is FALSE and NS is equal to TS (no known successor node) and (PS+1) modulo ($\mathbf{N}_{max\_master}$+1) is equal to TS and either

(a) SilenceTimer is greater than or equal to $\mathbf{T}_{usage\_timeout}$ or

(b) ReceivedInvalidFrame is TRUE,

then set SoleMaster TRUE to indicate that this station is the only master; set FrameCount to zero; set ReceivedInvalidFrame to FALSE; and enter the USE_TOKEN state.

9.5.6.9 ANSWER_DATA_REQUEST

The ANSWER_DATA_REQUEST state is entered when a BACnet Data Expecting Reply, a Test_Request, or a proprietary frame that expects a reply is received.

Reply

If a reply is available from the higher layers within $\mathbf{T}_{\text{reply\_delay}}$ after the reception of the final octet of the requesting frame (the mechanism used to determine this is a local matter),

then call SendFrame to transmit the reply frame and enter the IDLE state to wait for the next frame.

DeferredReply

If no reply will be available from the higher layers within $\mathbf{T}_{\text{reply\_delay}}$ after the reception of the final octet of the requesting frame (the mechanism used to determine this is a local matter),

then an immediate reply is not possible. Any reply shall wait until this node receives the token. Call SendFrame to transmit a Reply Postponed frame, and enter the IDLE state.

9.5.7 Slave Node Finite State Machine

The state machine for a slave node is similar to, but considerably simpler than, that for a master node. A slave node shall neither transmit nor receive segmented messages. If a slave node receives a segmented BACnet-Confirmed-Request-PDU, the node shall respond with a BACnet-Abort-PDU specifying abort-reason "segmentation not supported." Figure 9-5 shows the Slave Node state machine, which is described fully in the following text.

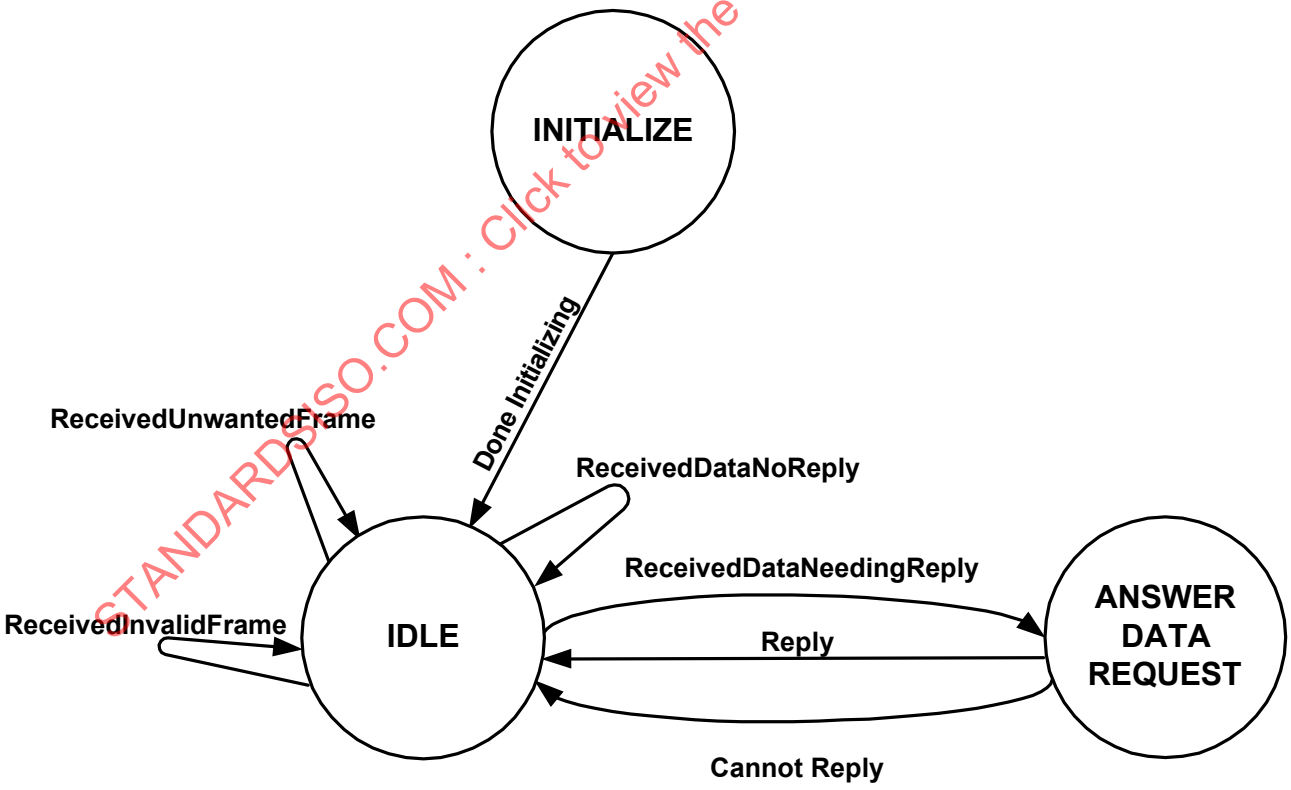


Figure 9-5. Slave Node State Machine

9.5.7.1 INITIALIZE

When a slave node is powered up or reset, it shall unconditionally enter the INITIALIZE state.

DoneInitializing
Unconditionally,

set TS to the node's station address; set ReceivedValidFrame and ReceivedInvalidFrame to FALSE; and enter the IDLE state.

9.5.7.2 IDLE

In the IDLE state, the node waits for a frame.

ReceivedInvalidFrame
If ReceivedInvalidFrame is TRUE,

then an invalid frame was received. Set ReceivedInvalidFrame to FALSE, and enter the IDLE state to wait for the next frame.

ReceivedUnwantedFrame
If ReceivedValidFrame is TRUE and either

(a) DestinationAddress is not equal to either TS (this station) or 255 (broadcast) or

(b) DestinationAddress is equal to 255 (broadcast) and FrameType has a value of BACnet Data Expecting Reply, Test Request, or a proprietary type known to this node that expects a reply (such frames may not be broadcast) or

(c) FrameType has a value of Token, Poll For Master, Reply To Poll For Master, Reply Postponed, or a standard or proprietary frame type not known to this node,

then an unexpected or unwanted frame was received. Set ReceivedValidFrame to FALSE, and enter the IDLE state to wait for the next frame.

ReceivedDataNoReply
If ReceivedValidFrame is TRUE and DestinationAddress is equal to either TS (this station) or 255 (broadcast) and FrameType is equal to BACnet Data Not Expecting Reply, Test_Response, or a proprietary type known to this node that does not expect a reply,

then indicate successful reception to the higher layers, set ReceivedValidFrame to FALSE, and enter the IDLE state.

ReceivedDataNeedingReply
If ReceivedValidFrame is TRUE and DestinationAddress is equal to TS (this station) and FrameType is equal to BACnet Data Expecting Reply, Test Request, or a proprietary type known to this node that expects a reply,

then indicate successful reception to the higher layers (management entity in the case of Test_Request), set ReceivedValidFrame to FALSE, and enter the ANSWER_DATA_REQUEST state.

9.5.7.3 ANSWER_DATA_REQUEST

The ANSWER_DATA_REQUEST state is entered when a BACnet Data Expecting Reply, a Test_Request, or a proprietary frame that expects a reply is received.

Reply
If a reply is available from the higher layers within $\mathbf{T}_{\mathbf{reply\_delay}}$ after the reception of the final octet of the requesting frame (the mechanism used to determine this is a local matter),

then call SendFrame to transmit the reply frame, and enter the IDLE state to wait for the next frame.

CannotReply

If no reply will be available from the higher layers within $\mathbf{T}_{\mathbf{reply\_delay}}$ after the reception of the final octet of the requesting frame (the mechanism used to determine this is a local matter),

then no reply is possible. Enter the IDLE state.

9.6 Cyclic Redundancy Check (CRC)

MS/TP uses Cyclic Redundancy Checks (CRC) to provide error-detection. CRCs have a number of advantages over simpler error detection methods, such as parity or checksums, which are commonly used with UART-based networks. The major drawbacks of parity are that it adds one bit of overhead to each transmitted octet and that it will not detect an even number of errors within one octet. Exclusive OR checksums, sometimes called longitudinal parity, offer reduced overhead over octet parity but suffer from the same inability to detect an even number of errors in a given bit position. Additive checksums are similar but the exact error detection characteristics are dependent on bit position.

ISO 8802-3, ARCNET, and many other standard and proprietary communications systems use more robust CRCs. Mathematically, a CRC is the remainder that results when a data stream (such as a frame) taken as a binary number is divided modulo two by a generator polynomial. The proof of the error-detecting properties of the CRC and the selection of appropriate polynomials are beyond the scope of this document.

The MS/TP frame header CRC uses the polynomial

$$G(X) = X^8 + X^7 + 1$$

In operation, at the transmitter, the initial content of the CRC register of the device computing the remainder of the division is preset to all ones. The register is then modified by division by the generator polynomial G(x) of the Frame Type, Destination Address, Source Address, and Length fields. The ones-complement of the resulting remainder is transmitted as the 8-bit Header CRC.

At the receiver, the initial content of the CRC register of the device computing the remainder of the division is preset to all ones. The register is then modified by division by the generator polynomial G(x) of the Frame Type, Destination Address, Source Address, Length, and Header CRC fields of the incoming message. In the absence of transmission errors, the resultant remainder will be:

0101 0101 (x^0 through x^7, respectively).

The MS/TP data CRC uses the CRC-CCITT polynomial

$$G(X) = X^{16} + X^{12} + X^5 + 1$$

In operation, at the transmitter, the initial content of the CRC register of the device computing the remainder of the division is preset to all ones. The register is then modified by division by the generator polynomial G(x) of the Data field. The ones-complement of the resulting remainder is transmitted, least significant octet first, as the 16 bit Data CRC.

At the receiver, the initial content of the CRC register of the device computing the remainder of the division is preset to all ones. The register is then modified by division by the generator polynomial G(x) of the Data and Data CRC fields of the incoming message. In the absence of transmission errors, the resultant remainder will be

1111 0000 1011 1000 (x^0 through x^{15}, respectively).

NOTE: The initialization of the CRC register to all ones and the complementing of the register before transmission prevent the CRC from having a value of zero if the covered field is all zeros.

Annex G describes the implementation of the CRC algorithms in software.

9.7 Interfacing MS/TP LANs with Other BACnet LANs

9.7.1 Routing of Messages from MS/TP

When a network entity with routing capability receives from a directly connected MS/TP data link an NPDU whose 'data_expecting_reply' parameter is TRUE and the NPDU is to be routed to another network according to the procedures of Clause 6, the network entity shall direct the MS/TP data link to transmit a Reply Postponed frame before attempting to route the NPDU. This allows the routing node to leave the ANSWER_DATA_REQUEST state and the sending node to leave the WAIT_FOR_REPLY state before the potentially lengthy process of routing the NPDU is begun.

9.7.2 Routing of Messages to MS/TP

When a network entity issues a DL_UNITDATA.request to a directly connected MS/TP data link, it shall set the 'data_expecting_reply' parameter of the DL-UNITDATA.request equal to the value of the 'data_expecting_reply' parameter of the network protocol control information of the NPDU, which is transferred in the 'data' parameter of the request.

9.8 Responding BACnet User Processing of Messages from MS/TP

In 5.4.5.3, AWAIT_RESPONSE, the following transition shall be added:

PostponeReply

If a CONF_SERV.response will not be received from the local application layer early enough that a reply MS/TP frame would be received by the remote node within $\mathbf{T}_{\text{reply\_timeout}}$ (defined in 9.5.3) after the transmission of the original BACnet-Confirmed-Request-PDU (the means of this determination are a local matter),

then direct the MS/TP data link to transmit a Reply Postponed frame and enter the AWAIT_RESPONSE state.

In 5.4.5.3, AWAIT_RESPONSE, in the transition SendSegmentedComplexACK, the text "transmit a BACnet-ComplexACK-PDU..." shall be replaced by "direct the MS/TP data link to transmit a Reply Postponed frame; transmit a BACnet-ComplexACK-PDU...." (It is necessary to postpone the reply because transmission of the segmented ComplexACK cannot begin until the node holds the token.)

9.9 Repeaters

If any of the limits in 9.2.1 and 9.2.2 are exceeded, one or more repeaters is required. An MS/TP EIA-485 Repeater is defined as an active device that provides selective interconnection between two or more segments of MS/TP cable. The repeater contains logic that detects and passes signals received from one segment onto all other segments. The segment from which signals are received is determined according to a priority algorithm.

The method used by a repeater to detect signals and to distinguish them from noise is a local matter, subject to the following constraints:

(a) The repeater may not lengthen or shorten the duration of any bit of the output data stream by more than 2% relative to the input data stream.

(b) The repeater may not delay the output data stream by more than two bit times relative to the input data stream.

No more than 10 bit times of delay shall exist in the path between any two nodes of an MS/TP network. This corresponds to five repeaters with worst-case delays (if delay by the medium is negligible, as it will be at all except the highest baud rates) or a greater number of repeaters with smaller delays.

The minimum value of repeater turnoff delay $\mathbf{T}_{\text{roff}}$ is dictated by the maximum amount of idle line allowed during a single frame, $\mathbf{T}_{\text{frame\_gap}}$. If the value of the octet immediately preceding the idle is X'FF', then there may be up to $9+\mathbf{T}_{\text{frame\_gap}} = 29$ bit times of one state between zero (start) bits. Thus, $\mathbf{T}_{\text{roff}}$ must be larger than 29 bit times if it is not to turn off during the transmission of a frame.

In order to avoid contention between repeater turnoff and the beginning of the next frame, the repeater turnoff delay $\mathbf{T_{roff}}$ may be no larger than $\mathbf{T_{turnaround}}$ bit times, and a node may not enable its driver until a minimum of $\mathbf{T_{turnaround}}$ after the end of the previous frame. Thus

29 bit times < $\mathbf{T_{roff}}$ < 40 bit times.

An N-way repeater may be represented by an N+1 state finite state machine. One state is the IDLE state, and the others each receive from one segment and re-transmit on all other segments. The repeater state machine uses the timer PortIdle to control the return to the IDLE state. PortIdle shall have a resolution of one bit time or finer.

9.9.1 IDLE

In the IDLE state, all receivers are enabled and all transmitters are disabled. The repeater remains in the IDLE state until a logical zero (i.e., the beginning of a start bit) is detected on some port.

When a zero is detected, the repeater disables all receivers, except the one on which the zero was detected, and enables all transmitters, except the one associated with the port on which the zero was detected. The repeater then enters the state associated with the active receiver port. If a zero is detected simultaneously on more than one port, the method used to arbitrate between them is a local matter.

Port1Active
If a zero is detected on Port 1,

then disable all receivers except Port 1; enable all transmitters except Port 1; pass the zero to all enabled transmitters; set PortIdle to zero; and enter the PORT_1_ACTIVE state.

Port2Active
If a zero is detected on Port 2,

then disable all receivers except Port 2; enable all transmitters except Port 2; pass the zero to all enabled transmitters; set PortIdle to zero; and enter the PORT_2_ACTIVE state.

This may be extended to as many ports as desired simply by adding transitions and PORT_N_ACTIVE states.

9.9.2 PORT_i_ACTIVE

In the PORT_i_ACTIVE state, the Repeater passes signals from Port i to all other ports. The Repeater will remain in this state until Port i becomes idle. Idleness is defined as the absence of the logical zero state at Port i for more than $\mathbf{T_{roff}}$ bit times.

When idleness is detected, all transmitters are disabled, all receivers are enabled, and the Repeater enters the IDLE state to await renewed activity.

PortActive0
If a zero is detected on Port i,

then pass the zero to all other ports, set PortIdle to zero, and re-enter the current state.

PortActive1
If PortIdle is less than $\mathbf{T_{roff}}$ and a one is detected at Port i,

then pass the one to all other ports and re-enter the current state.

PortInactive
If PortIdle is greater than or equal to $\mathbf{T_{roff}}$,

then disable all transmitters, enable all receivers, and enter the IDLE state.

10 DATA LINK/PHYSICAL LAYERS: POINT-TO-POINT (PTP)

This clause defines a data link layer protocol by which two BACnet devices may communicate using a variety of point-to-point (PTP) communication mechanisms. These mechanisms may be accessed through an EIA-232 or bus-level interface to modems, line drivers, or other data communication equipment. The specific physical connection composing the PTP connection is a local matter.

This clause does not attempt to specify the means by which the virtual or physical connection is established. Rather, it specifies a protocol that allows two BACnet network layer entities to establish a BACnet PTP data link connection, reliably exchange BACnet PDUs, and perform an orderly termination of a BACnet PTP connection using an already established physical connection.

This data link layer protocol addresses the particular characteristics associated with a PTP connection. A PTP connection differs from other BACnet data link/physical layer options in several ways: it is capable of full duplex operation, it may be temporary in nature, and it may be significantly slower. The PTP protocol is only used between devices that are half- routers. See 6.7.

This protocol does not assume that the answering device is in a state where it can accept BACnet PDUs containing binary information. For instance, the same serial device port that is dialed into by a BACnet device may also be dialed into and logged onto by a human operator using a simple ANSI X3.4 terminal. Therefore, the connection establishment procedure is initiated using an ANSI X3.4 printable character sequence.

The connection process also provides for optional password protection. The configuration and checking of the password parameter is considered to be a local matter.

10.1 Overview

Once a physical connection has been established between the calling device and the answering device, a sequence of frames are exchanged to establish a BACnet connection. If the connection is established, the two devices may freely exchange BACnet PDUs. Either the calling device or the answering device may initiate a termination of the connection. The connection remains until a request for termination has been issued by either device, either device determines that the physical layer connection has been lost, or until a local timer expires, indicating that the peer device is no longer active. Unlike other BACnet data link protocols, the PTP protocol is acknowledged using an alternating bit approach. It should be noted that the protocol allows PDUs to be exchanged between the two devices simultaneously to take advantage of full duplex operation.

Note that it is also possible that two devices are permanently connected at the physical layer in which case the BACnet connect sequence is performed only once, at initialization time. In this case both devices would be running the PTP data link layer and would always be capable of sending and receiving BACnet PTP data link frames.

10.2 Service Specification

PTP includes a data link layer sufficient to provide to the BACnet network layer the same services as are offered by ISO 8802-2 Type 1. Because PTP is a connection-oriented data link layer, additional primitives are needed to manage the connection establishment and termination phases. PTP does not provide all of the functionality of ISO 8802-2 Type 2.

This subclause describes the primitives and parameters associated with the provided services. The parameters are described in an abstract sense, which does not constrain the implementation method. These primitives provide an interface to the BACnet network layer consistent with the other BACnet data link options except for the addition of connection management primitives.

10.2.1 DL-UNITDATA.request

10.2.1.1 Function

This primitive is the service request primitive for the unacknowledged connectionless-mode data transfer service.

10.2.1.2 Semantics of the Service Primitive

The primitive shall provide parameters as follows:

DL-UNITDATA.request (
source_address,
destination_address,
data,
priority
)

Each source and destination address consists of the logical concatenation of a medium access control (MAC) address and a link service access point (LSAP). However, since PTP does not define or use MAC addresses and since it supports only the BACnet network layer, the 'source_address' and 'destination_address' parameters are ignored.

The 'data' parameter specifies the link service data unit (LSDU) to be transferred by the PTP entity.

The 'priority' parameter specifies the priority desired for the data unit transfer. The priority parameter is ignored by PTP.

10.2.1.3 When Generated

This primitive is passed from the network layer to the PTP entity to request that a network protocol data unit (NPDU) be sent to one or more remote LSAPs using unacknowledged connectionless-mode procedures.

10.2.1.4 Effect on Receipt

Receipt of this primitive causes the PTP entity to attempt to send the NPDU using unacknowledged connectionless-mode procedures.

10.2.2 DL-UNITDATA.indication

10.2.2.1 Function

This primitive is the service indication primitive for the unacknowledged connectionless-mode data transfer service.

10.2.2.2 Semantics of the Service Primitive

The primitive shall provide the following parameters:

DL-UNITDATA.indication (
source_address,
destination_address,
data,
priority
)

Each source and destination address consists of the logical concatenation of a medium access control (MAC) address and a link service access point (LSAP). However, since PTP does not define or use MAC addresses, and since it supports only the BACnet network layer, the 'source_address' and 'destination_address' parameters are ignored.

The 'data' parameter specifies the link service data unit that has been received by the PTP entity.

The 'priority' parameter specifies the priority desired for the data unit transfer. The priority parameter is ignored by PTP.

10.2.2.3 When Generated

This primitive is passed from the PTP entity to the network layer to indicate the arrival of an NPDU from the specified remote entity.

10.2.2.4 Effect on Receipt

The effect of receipt of this primitive by the network layer is unspecified.

10.2.3 Test_Request and Test_Response

ISO 8802-2 Type 1 defines XID and TEST PDUs and procedures but does not define an interface to invoke them from the network layer. Test_Request and Test_Response PDUs and procedures have been defined for PTP to accomplish the same

functions. Because PTP supports only the equivalent of a single LSAP, these PDUs are sufficient to implement the relevant aspects of XID as well.

The response with Test_Response to a Test_Request PDU is mandatory for all PTP nodes. The origination of a Test_Request PDU is optional.

10.2.3.1 Use of Test_Request and Test_Response for ISO 8802-2 TEST Functions

The TEST function provides a facility to conduct loopback tests of the PTP to PTP transmission path. Successful completion of the test consists of sending a Test_Request PDU with a particular information field to the designated destination, and receiving, in return, the identical information field in a Test_Response PDU.

If a receiving node can successfully receive and return the information field, it shall do so. If it cannot receive and return the entire information field but can detect the reception of a valid Test_Request frame (for example, by computing the CRC on octets as they are received), then the receiving node shall discard the information field and return a Test_Response containing no information field. If the receiving node cannot detect the valid reception of frames with overlength information fields, then no response shall be returned.

10.2.3.2 Use of Test_Request and Test_Response for ISO 8802-2 XID functions

ISO 8802-2 describes seven possible uses of XID:

(a) XID can be used with a null DSAP and null SSAP as an "Are You There" test. Since PTP supports only the equivalent of a single LSAP, the Test_Request PDU with no data can perform this function.

(b) XID can be used with a group or global DSAP to identify group members or all active stations. Since PTP supports only the equivalent of a single LSAP, the Test_Request PDU with no data can perform this function.

(c) XID can be used for a duplicate address check.

(d) Class II LLCs may use XID to determine window size. PTP does not support Class II operation.

(e) XID may be used to identify the class of each LLC. Since PTP supports only Class I operation, this is a trivial operation.

(f) XID may be used to identify the service types supported by each LSAP. Since PTP supports only Class I operation, this is a trivial operation.

(g) An LLC can announce its presence by transmitting an XID with global DSAP. Since PTP supports only one LSAP, the equivalent can be accomplished by transmitting a Test_Response PDU.

10.2.4 DL-CONNECT.request

10.2.4.1 Function

This primitive is the service request primitive for the connection establishment service.

10.2.4.2 Semantics of the Service Primitive

The primitive shall provide the following parameters:

```
DL-CONNECT.request (
        destination_address,
        password
        )
```

The 'destination_address' parameter specifies the information required by the PTP entity to initiate the establishment of a physical connection between the local and remote BACnet devices. Although, as stated at the beginning of this clause, the establishment of the physical connection is a local matter, it is likely that this parameter would convey information such as

contained in the Port Info field of the Initialize-Routing-Table network layer message and subsequently stored in a node's routing table. See 6.4.7.

The 'password' parameter specifies the password to be used in the PTP connection process described in 10.4.8.

10.2.4.3 When Generated

This primitive is passed from the network layer to the PTP entity to request that a logical link connection be established.

10.2.4.4 Effect on Receipt

The receipt of this primitive causes the PTP entity to initiate establishment of a connection with the remote PTP entity.

10.2.5 DL-CONNECT.indication

10.2.5.1 Function

This primitive is the service indication primitive for the connection establishment service.

10.2.5.2 Semantics of the Service Primitive

The primitive provides no parameters.

10.2.5.3 When Generated

This primitive is passed from the PTP entity to the network layer to indicate that a logical link connection has been established.

10.2.5.4 Effect on Receipt

The network layer entity may use this connection for data unit transfer.

10.2.6 DL-CONNECT.confirm

10.2.6.1 Function

This primitive is the service confirmation primitive for the connection establishment service.

10.2.6.2 Semantics of the Service Primitive

The primitive shall provide the following parameter:

```
DL-CONNECT.confirm (
        status
        )
```

The 'status' parameter specifies whether or the not the connection has been successfully established.

10.2.6.3 When Generated

This primitive is passed from the PTP entity to the network layer to indicate that a logical link connection has been established.

10.2.6.4 Effect on Receipt

The network layer entity may use this connection for data unit transfer if the 'status' parameter indicates the successful establishment of a PTP connection.

10.2.7 DL-DISCONNECT.request

10.2.7.1 Function

This primitive is the service request primitive for the connection termination service.

10.2.7.2 Semantics of the Service Primitive

The primitive shall provide the following parameters:

DL-DISCONNECT.request (
destination_address
)

The 'destination_address' parameter specifies the information required by the PTP entity to initiate the establishment of a physical connection between the local and remote BACnet devices. The PTP entity uses this same information to identify the particular PTP connection instance that is to be terminated.

10.2.7.3 When Generated

This primitive is passed from the network layer to the PTP entity to request that a logical link connection be terminated.

10.2.7.4 Effect on Receipt

The receipt of this primitive causes the PTP entity to initiate termination of a connection with the remote PTP entity.

10.2.8 DL-DISCONNECT.indication

10.2.8.1 Function

This primitive is the service indication primitive for the connection termination service.

10.2.8.2 Semantics of the Service Primitive

The primitive shall provide the following parameters:

DL-DISCONNECT.indication (
reason
)

The 'reason' parameter specifies the reason for the disconnection. The reasons for disconnection may include a request by the remote entity, loss of physical connection, or an error internal to the PTP sublayer.

10.2.8.3 When Generated

This primitive is passed from the PTP entity to the network layer to indicate that a logical link connection has been terminated.

10.2.8.4 Effect on Receipt

The network layer entity may no longer use this connection for data unit transfer.

10.2.9 DL-DISCONNECT.confirm

10.2.9.1 Function

This primitive is the service confirmation primitive for the connection termination service.

10.2.9.2 Semantics of the Service Primitive

The primitive shall provide the following parameters:

DL-DISCONNECT.confirm (
destination_address
)

The 'destination_address' parameter specifies the information required by the PTP entity to initiate the establishment of a physical connection between the local and remote BACnet devices. The network layer entity uses this same information to identify the particular PTP connection instance that has been terminated.

10.2.9.3 When Generated

This primitive is passed from the PTP entity to the network layer to indicate that a logical link connection has been terminated.

10.2.9.4 Effect on Receipt

The network layer entity may no longer use this connection for data unit transfer.

10.3 Point-to-Point Frame Format

All PTP data link frames, with the exception of the ANSI X3.4 sequence used to initiate a PTP connection, have the following format:

Preamble	two octet preamble X'55FF'
Frame Type	one octet
Length	length of data field not including CRC, two octets, most significant octet first
Header CRC	one octet
Data	varies with frame type; variable length
Data CRC	(if data is present) two octets, least significant octet first

The Preamble, Frame Type, Length, and Header CRC are collectively known as the header segment of the frame. The Data and Data CRC are collectively known as the data segment of the frame. The Frame Type is used to distinguish between different types of MAC frames. Defined types are:

X'00'	Heartbeat XOFF
X'01'	Heartbeat XON
X'02'	Data 0
X'03'	Data 1
X'04'	Data Ack 0 XOFF
X'05'	Data Ack 1 XOFF
X'06'	Data Ack 0 XON
X'07'	Data Ack 1 XON
X'08'	Data Nak 0 XOFF
X'09'	Data Nak 1 XOFF
X'0A'	Data Nak 0 XON
X'0B'	Data Nak 1 XON
X'0C'	Connect Request
X'0D'	Connect Response
X'0E'	Disconnect Request
X'0F'	Disconnect Response
X'14'	Test_Request
X'15'	Test_Response

Frame Types X'00' through X'7F' are reserved by ASHRAE. Frame types X'10', X'11', and X '13' shall never be used for a valid frame type because of the character transparency method described in 10.3.1. Frame Types X'80' through X'FF' are available to vendors for proprietary (non-BACnet) frames. Proprietary PTP frames shall follow the same state machine transitions defined for Data frames.

The Data field is conditional on the Frame Type, as specified in the description of each Frame Type. If there is no Data field, then the length field shall be zero and the Data and Data CRC (data segment) shall be omitted.

The header CRC octet is the ones complement of the remainder that results when the Frame Type and Length fields are divided by the CRC polynomial

$$G(X) = X^8 + X^7 + 1.$$

The data CRC octets are the ones complement of the remainder that results when the Data field is divided by the CRC-CCITT polynomial

$$G(X) = X^{16} + X^{12} + X^5 + 1.$$

Annex G describes in detail the generation and checking of the CRCs.

10.3.1 Character Transparency and Flow Control

In order to support modems that respond to flow control or other control characters, character stuffing is used to prevent transmission of these codes as part of the data. Where a value corresponding to a control character would appear in a frame, it shall be prefixed with a data link escape code (X'10') and the high order bit shall be set in the value as transmitted. The control characters listed below shall be encoded in this manner. Implementations shall be able to receive and decode all encoded control characters.

X'10'	(DLE)	=>	X'10' X'90'
X'11'	(XON)	=>	X'10' X'91'
X'13'	(XOFF)	=>	X'10' X'93'

The characters X'11' (XON) and X'13' (XOFF) are never transmitted by the SendFrame procedure described in 10.4.4 and are ignored by the Receive Frame state machine described in 10.4.7. The use of these characters or of Request To Send (RTS), Clear To Send (CTS), or other EIA-232 control lines for flow control purposes is a local matter. The use of such methods of flow control is allowed only between a PTP device and local equipment such as a modem. Flow control between PTP devices shall be implemented using the flow control frames defined in 10.3.

10.3.2 Frame Types X'00' and X'01': Heartbeat Frames

A frame of one of these types is transmitted by each device periodically when no other data are ready to transmit, to indicate to the peer device that the data link is still active. Heartbeat frames contain no data segment. A type X'00' frame is transmitted to indicate to the peer device that the local device is not ready to accept Data frames. A type X'01' frame is transmitted to indicate readiness to receive Data frames.

10.3.3 Frame Types X'02' and X'03': Data Frames

A frame of one of these types is transmitted to convey data (NPDUs) to the peer device. The length of the data field of a Data frame may range from 0 to 501 octets. Successive transmissions alternate frame types; type X'02' corresponds to transmit sequence number 0, and type X'03' corresponds to transmit sequence number 1.

10.3.4 Frame Types X'04' through X'07': Data Ack Frames

A frame of one of these types is transmitted to acknowledge a correctly received Data frame. Data Ack frames contain no data segment. Frame types X'04' and X'06' acknowledge receipt of Data frames with sequence number 0 (type X'02'). Frame types X'05' and X'07' acknowledge receipt of Data frames with sequence number 1 (type X'03'). Frame types X'04' and X'05' indicate that the device is not ready to receive additional Data frames (XOFF). Frame types X'06' and X'07' indicate that the device is ready to receive additional Data frames (XON).

10.3.5 Frame Types X'08' through X'0B': Data Nak Frames

A frame of one of these types is used to reject an incorrectly received Data frame. Data Nak frames are transmitted when the header segment of a Data frame has been correctly received but the data segment of the frame contains an error or when a Data frame cannot be accepted due to receiver buffer limitations. Data Nak frames contain no data segment. Frame types X'08' and X'0A' reject Data frames with sequence number 0 (type X'02'). Frame types X'09' and X'0B' reject Data frames with sequence number 1 (type X'03'). Frame types X'08' and X'09' indicate that the device is not ready to receive additional Data frames (XOFF). Frame types X'0A' and X'0B' indicate that the device is ready to receive additional Data frames (XON).

10.3.6 Frame Type X'0C': Connect Request Frame

The Connect Request frame is issued by the answering device in an attempt to establish a BACnet connection. Connect Request frames contain no data segment.

10.3.7 Frame Type X'0D': Connect Response Frame

The Connect Response frame is issued by a device in response to a received Connect Request frame. The data field of the Connect Response frame, if present, contains a password. The length and content of the optional password field are a local matter.

10.3.8 Frame Type X'0E': Disconnect Request Frame

The Disconnect Request frame may be issued by either device when it wishes to discontinue the BACnet PTP dialogue. The data field of the frame conveys the reason for requesting a disconnect and shall be one octet in length. The permissible values for the data are:

X'00' no more data needs to be transmitted,
X'01' the peer process is being preempted,
X'02' the received password is invalid,
X'03' other.

10.3.9 Frame Type X'0F': Disconnect Response Frame

The Disconnect Response frame is used to acknowledge a previously received Disconnect Request frame. The Disconnect Response frame indicates that the responding device accepts the request to disconnect. Disconnect Response frames contain no data segment.

10.3.10 Frame Type X'14': Test_Request

This frame is used to initiate a loopback test of the PTP transmission path. The use of this frame is described in detail in 10.2.3. The length of the data field of a Test_Request frame may range from 0 to 501 octets.

10.3.11 Frame Type X'15': Test_Response

This frame is used to reply to a Test_Request frame. The use of this frame is described in detail in 10.2.3. The length of the data field of a Test_Response frame may range from 0 to 501 octets. The data, if present, shall be those that were present in the initiating Test_Request.

10.4 PTP Medium Access Control Protocol

This subclause defines the PTP protocol. The protocol definition has been broken into several discrete parts that collectively describe and define the entire PTP protocol. The first part presents a universal asynchronous receiver transmitter (UART) model of the hardware platform. This is followed by definitions of the variables and constants used to define the protocol. A finite state machine is used to define in detail the process of receiving PTP frames (see 10.4.7). An informal procedure describes in detail the process of transmitting a PTP frame (see 10.4.4). These details are referenced in subsequent subclauses, which define the protocol at a higher level of abstraction.

The process of establishing a PTP connection and terminating a PTP connection is defined by a finite state machine called the Connection State Machine (see 10.4.9). The PTP protocol is full duplex. Thus, when a PTP connection is active, each device may simultaneously play the role of transmitting and receiving PTP messages. Two separate finite-state machines define the aspects of the protocol that pertain to these two roles. These finite state machines are call the Transmission State Machine (see 10.4.10) and the Reception State Machine (see 10.4.11). The Transmission State Machine and the Reception State Machine are assumed to operate concurrently whenever the Connection State Machine is in the CONNECTED state. In this model, the Reception State Machine and the Transmission State Machine exchange information through the use of shared variables. The details of transmitting and receiving a PTP frame are described by the SendFrame, SendHeaderOctet, and SendOctet procedures and the ReceiveFrame state machine respectively.

10.4.1 UART Receiver Model

The receiver interface to a UART is modeled as a data register and two Boolean flags. These are intended to closely resemble the functions of commercial UART chips but in a generic and nonprescriptive fashion. The model is used by both the procedural and state machine descriptions.

10.4.1.1 DataRegister

The DataRegister holds the octet most recently received. The contents of this register after the occurrence of a framing or overrun error are not specified.

10.4.1.2 DataAvailable

The flag DataAvailable is TRUE if an octet is available in DataRegister. A means of setting this flag to FALSE when the associated data have been read from DataRegister shall be provided. Many common UARTs set DataAvailable FALSE automatically when DataRegister is read.

10.4.1.3 ReceiveError

The flag ReceiveError is TRUE if an error is detected during the reception of an octet. Many common UARTs detect several types of receive errors, in particular framing errors and overrun errors. ReceiveError shall be TRUE if any of these errors is detected.

A framing error occurs if a logical zero is received when a stop bit (logical one) is expected.

An overrun error occurs if an octet is received before an earlier octet is read from DataRegister. In general, the occurrence of overrun errors is evidence of improper design. However, it is recognized that critical system events may cause overrun errors to occur from time to time. The inclusion of this error in the state machine processing ensures that such errors are handled in a deterministic fashion.

A means of setting ReceiveError to FALSE when the associated error has been recognized shall be provided.

10.4.2 Variables

The variables and timers used by the PTP protocol are described in this subclause.

Ack0Received A Boolean flag indicating whether (TRUE) or not (FALSE) the Reception State Machine has received an acknowledgment that a previous Data frame with a sequence number of 0 was correctly received.

Ack1Received A Boolean flag indicating whether (TRUE) or not (FALSE) the Reception State Machine has received an acknowledgment that a previous Data frame with a sequence number of 1 was correctly received.

DataCRC Used to accumulate the CRC on the data field of a frame.

DataLength An unsigned integer in the range from 0 to 501 that indicates the expected number of octets in the Data field of a PTP frame. This value is derived from the Length field of the received PTP frame. The value of DataLength excludes any data link escape octets (X'10') and the octets in the Data CRC.

DLE_Mask A bit mask used to used to process received octets in order to account for the fact that they may be encoded.

FrameType Used by the Receive State Machine to store the frame type of a received frame.

HeaderCRC Used to accumulate the CRC of the header of a frame.

HeartbeatTimer A timer used to initiate Heartbeat frames to keep the link active.

InactivityTimer A timer used to monitor the time since this station received a frame.

Index Indicates the location of the end of the data in the InputBuffer array.

InputBuffer[ ] An array of octets used to store data octets as they are received. InputBuffer is indexed from 0 to InputBufferSize-1. The maximum size of the data field of a frame is 501 octets.

InputBufferSize The number of elements in the array InputBuffer[].

Nak0Received A Boolean flag indicating whether (TRUE) or not (FALSE) the Reception State Machine has received a reject in response to a previous Data frame with a sequence number of 0.

Nak1Received A Boolean flag indicating whether (TRUE) or not (FALSE) the Reception State Machine has received a reject in response to a previous Data frame with a sequence number of 1.

ReceivedInvalidFrame A Boolean flag set to TRUE by the Receive Frame Machine if an error of any type is detected. Errors include octet framing, overrun, CRC, and receive buffer overflow.

ReceivedValidFrame A Boolean flag set to TRUE by the Receive Frame Machine if a valid frame has been received.

ReceptionBlocked An enumerated variable indicating whether or not reception of Data frames is blocked. The values of the enumeration are BLOCKED, ALMOST_BLOCKED, and NOT_BLOCKED. The value of this variable is determined by a buffer manager. The buffer manager process is a local matter.

ResponseTimer A timer used to monitor the time spent waiting for a response from the remote device.

RetryCount A counter of transmission retries.

RxSequenceNumber An integer containing the sequence number (0 or 1) expected for the next Data frame to be received.

SendingFrameNow A Boolean flag indicating whether (TRUE) or not (FALSE) an invocation of the SendFrame procedure is currently in the process of transmitting octets.

SilenceTimer A timer used to monitor the time since this station received an octet.

TransmissionBlocked A Boolean flag indicating whether (TRUE) or not (FALSE) frame transmission has been blocked. The value of this flag is determined by the receipt of XON and XOFF frames from the peer device.

TransmitDataCRC Used to accumulate the CRC on the data field of a frame being transmitted.

TransmitHeaderCRC Used to accumulate the CRC on the header of a frame being transmitted.

TxSequenceNumber An integer containing the sequence number (0 or 1) for the next Data frame to be transmitted.

10.4.3 Parameters

The following parameters are used in the PTP data link protocol.

$N_{retries}$ The maximum number of times a frame shall be sent before an error is reported. The value of $N_{retries}$ shall be 3.

T_{conn_rqst} The maximum time allowed by a calling device for an answering device to issue a PTP Connect Request frame once the physical connection has been established and the ANSI X3.4 trigger sequence has been transmitted. The value of T_{conn_rqst} shall be 15 seconds. This represents the processing time required for the answering device to recognize the ANSI X3.4 trigger sequence, prepare the communication port for PTP protocol, and transmit the Connect Request frame.

T_{conn_rsp} The maximum time allowed to respond to a PTP Connect Request frame with a PTP Connect Response frame. The value of T_{conn_rsp} shall be 15 seconds. This represents the time required for the calling device to process the Connect Request frame and the time required to transmit the Connect Response frame.

T_{frame_abort} The maximum time allowed between receipt of octets in a frame after which time the receiving device shall assume a transmission error. The value of T_{frame_abort} shall be 2 seconds.

$T_{heartbeat}$ The maximum delay between frame transmissions before a heartbeat frame must be sent. The value of $T_{heartbeat}$ shall be 15 seconds.

$T_{inactivity}$ The maximum amount of time the InactivityTimer may attain, after which a device may assume that the PTP connection has been disrupted. The value of $T_{inactivity}$ shall be 60 seconds.

$T_{response}$ The maximum time allowed waiting for a Data Ack frame in response to a Data frame. The value of $T_{response}$ shall be 5 seconds.

10.4.4 SendFrame Procedure

This subclause describes the transmission of PTP data link frames. A mutual exclusion semaphore, SendingFrameNow, synchronizes access to the functionality by multiple, asynchronous invocations. Frames are transmitted in two parts, the header segment and the data segment.

NOTE: At the implementer's option, character-level flow control may be implemented by conditioning the transmission of octets by this procedure based on the reception of the characters XOFF (X'13') and XON (X'11') or by the presence or absence of active levels on certain EIA-232 control lines. Such character-level flow control is a local matter.

(a) If SendingFrameNow is TRUE, then wait for the other invocation of the SendFrame procedure to set SendingFrameNow to FALSE.

(b) Set SendingFrameNow to TRUE.

(c) Transmit the preamble octets X'55' and X'FF'.

(d) Initialize TransmitHeaderCRC to X'FF'.

(e) Call SendHeaderOctet to transmit the frame type and the high and low data length octets.

(f) Call SendOctet to transmit the one's-complement of TransmitHeaderCRC.

(g) If the data length is zero, proceed to step (m). Otherwise, proceed to step (h).

(h) Initialize TransmitDataCRC to X'FFFF'.

(i) Accumulate each data octet into TransmitDataCRC.

(j) Call SendOctet to transmit each data octet.

(k) Call SendOctet to transmit the one's-complement of the least significant octet of TransmitDataCRC.

(l) Call SendOctet to transmit the one's-complement of the most significant octet of TransmitDataCRC.

(m) Set HeartbeatTimer to zero, set SendingFrameNow to FALSE. Transmission is complete.

10.4.5 SendHeaderOctet Procedure

This subclause describes the transmission of the header octets of PTP frames.

(a) Accumulate the header octet into TransmitHeaderCRC.

(b) Call SendOctet to transmit the header octet.

10.4.6 SendOctet Procedure

This subclause describes the transmission of octets of PTP frames.

(a) If the value of the octet is not X'10', X'11', or X'13', then transmit the octet.

(b) If the value of the octet is X'10', X'11', or X'13', then transmit the value X'10', set the high order bit of the octet, and transmit the modified octet.

10.4.7 Receive Frame State Machine

This subclause describes the reception of a PTP frame by a BACnet device. The description of operation is as a finite state machine. Figure 10-1 shows the Receive Frame state machine, which is described fully in this subclause. Each state is given a name, specified in all capital letters. Transitions are also named, in mixed upper- and lowercase letters. Transitions are described as a series of conditions followed by a series of actions to be taken if the conditions are met. The final action in each transition is entry into a new state, which may be the same as the current state.

The Receive Frame state machine operates independently from the other PTP state machines, communicating with them by means of flags and other variables. The description assumes that the other state machines can process received frames and other indications from the Receive Frame state machine before the next frame begins. The means by which this behavior is implemented are a local matter.

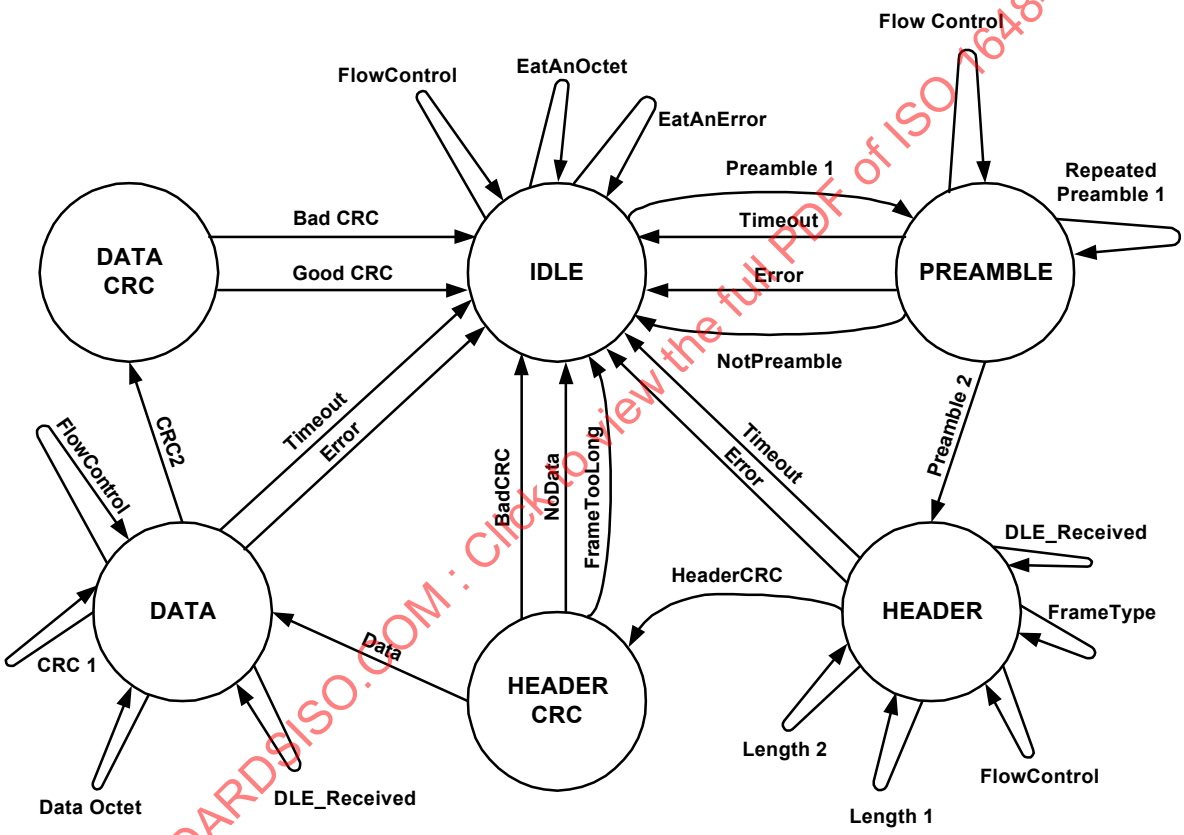


Figure 10-1. Receive Frame State Machine.

10.4.7.1 IDLE

In the IDLE state, the node waits for the beginning of a frame.

EatAnError

If ReceiveError is TRUE,

then set SilenceTimer to zero; set ReceiveError to FALSE; and enter the IDLE state to wait for the start of a frame.

FlowControl

If ReceiveError is FALSE and DataAvailable is TRUE and the content of DataRegister is equal to either X'11' or X'13',

then set SilenceTimer to zero; set DataAvailable to FALSE; and enter the IDLE state.

EatAnOctet

If ReceiveError is FALSE and DataAvailable is TRUE and the content of DataRegister is not equal to X'55', X'11', or X'13',

then set SilenceTimer to zero; set DataAvailable to FALSE; and enter the IDLE state to wait for the start of a frame.

Preamble1

If ReceiveError is FALSE and DataAvailable is TRUE and the content of DataRegister is equal to X'55',

then set SilenceTimer to zero; set DataAvailable to FALSE; set ReceivedValidFrame to FALSE; set ReceivedInvalidFrame to FALSE; and enter the PREAMBLE state to receive the remainder of the frame.

10.4.7.2 PREAMBLE

In the PREAMBLE state, the node waits for the second octet of the preamble.

Timeout

If SilenceTimer is greater than $\mathbf{T}_{\text{frame\_abort}}$,

then a correct preamble has not been received. Enter the IDLE state to wait for the start of a frame.

Error

If ReceiveError is TRUE,

then set SilenceTimer to zero; set ReceiveError to FALSE; and enter the IDLE state to wait for the start of a frame.

FlowControl

If ReceiveError is FALSE and DataAvailable is TRUE and the content of DataRegister is equal to either X'11' or X'13',

then set SilenceTimer to zero; set DataAvailable to FALSE; and enter the PREAMBLE state.

RepeatedPreamble1

If ReceiveError is FALSE and DataAvailable is TRUE and the contents of DataRegister is equal to X'55',

then set SilenceTimer to zero; set DataAvailable to FALSE; and enter the PREAMBLE state to wait for the second preamble octet.

NotPreamble

If ReceiveError is FALSE and DataAvailable is TRUE and the content of DataRegister is not equal to X'FF', X'55', X'11', or X'13',

then set SilenceTimer to zero; set DataAvailable to FALSE; and enter the IDLE state to wait for the start of a frame.

Preamble2

If ReceiveError is FALSE and DataAvailable is TRUE and the content of DataRegister is equal to X'FF',

then set SilenceTimer to zero; set DLE_Mask to X'00'; set HeaderCRC to X'FF'; set DataAvailable to FALSE; set Index to zero; and enter the HEADER state to receive the remainder of the frame.

10.4.7.3 HEADER

In the HEADER state, the node waits for the fixed frame header.

Timeout
If SilenceTimer is greater than $\mathbf{T}_{\mathbf{frame\_abort}}$,

then enter the IDLE state to wait for the start of a frame.

Error
If ReceiveError is TRUE,

then set SilenceTimer to zero; set ReceiveError to FALSE; and enter the IDLE state to wait for the start of a frame.

FlowControl
If ReceiveError is FALSE and DataAvailable is TRUE and the contents of DataRegister is equal to either X'11' or X'13',

then set SilenceTimer to zero; set DataAvailable to FALSE; and enter the HEADER state.

DLE_Received
If ReceiveError is FALSE and DataAvailable is TRUE and the content of the DataRegister is equal to X'10',

then set DLE_Mask to X'80' and enter the HEADER state.

FrameType
If ReceiveError is FALSE and DataAvailable is TRUE and Index is 0 and the content of DataRegister is not equal to X'10', X'11', or X'13',

then perform a bitwise AND of the ones-complement of the DLE_Mask and the contents of DataRegister; save the result as FrameType; accumulate the result into HeaderCRC; set DataAvailable to FALSE; set DLE_Mask to X'00'; set Index to 1; and enter the HEADER state.

Length1
If ReceiveError is FALSE and DataAvailable is TRUE and Index is 1 and the content of DataRegister is not equal to X'10', X'11', or X'13',

then perform a bitwise AND of the ones-complement of the DLE_Mask and the contents of DataRegister; accumulate the result into HeaderCRC; multiply the result by 256 and save this result as DataLength; set DataAvailable to FALSE; set DLE_Mask to X'00'; set Index to 2; and enter the HEADER state.

Length2
If ReceiveError is FALSE and DataAvailable is TRUE and Index is 2 and the content of DataRegister is not equal to X'10', X'11', or X'13',

then perform a bitwise AND of the ones-complement of the DLE_Mask and the contents of DataRegister; accumulate the result into HeaderCRC; add the result to DataLength and save this result as DataLength; set DataAvailable to FALSE; set DLE_Mask to X'00'; set Index to 3; and enter the HEADER state.

HeaderCRC
If ReceiveError is FALSE and DataAvailable is TRUE and Index is 3 and the content of DataRegister is not equal to X'10', X'11', or X'13',

then perform a bitwise AND of the ones-complement of the DLE_Mask and the contents of DataRegister; accumulate the result into HeaderCRC; set DataAvailable to FALSE; set DLE_Mask to X'00'; and enter the HEADER_CRC state.

10.4.7.4 HEADER_CRC

In the HEADER_CRC state, the node validates the CRC on the fixed frame header.

BadCRC
If the value of HeaderCRC is not X'55',

then enter the IDLE state to wait for the start of the next frame.

FrameTooLong
If the value of HeaderCRC is X'55' and DataLength is greater than InputBufferSize,

then set ReceivedInvalidFrame to TRUE to indicate that a frame cannot be received, and enter the IDLE state to wait for the start of the next frame.

NoData
If the value of HeaderCRC is X'55' and DataLength is zero,

then set ReceivedValidFrame to TRUE to indicate that a frame with no data has been received, and enter the IDLE state to wait for the start of the next frame.

Data
If the value of HeaderCRC is X'55' and DataLength is greater than zero but less than or equal to InputBufferSize,

then set Index to zero; set DataCRC to X'FFFF'; and enter the DATA state to receive the data field of the frame.

10.4.7.5 DATA

In the DATA state, the node waits for the data field of a frame.

Timeout
If SilenceTimer is greater than $\mathbf{T}_{\text{frame\_abort}}$,

then set ReceivedInvalidFrame to TRUE to indicate that an error has occurred during the reception of a frame, and enter the IDLE state to wait for the start of the next frame.

Error
If ReceiveError is TRUE,

then set SilenceTimer to zero; set ReceiveError to FALSE; set ReceivedInvalidFrame to TRUE to indicate that an error has occurred during the reception of a frame; and enter the IDLE state to wait for the start of the next frame.

FlowControl
If ReceiveError is FALSE and DataAvailable is TRUE and the content of DataRegister is equal to either X'11' or X'13',

then set SilenceTimer to zero; set DataAvailable to FALSE; and enter the DATA state.

DLE_Received
If ReceiveError is FALSE and DataAvailable is TRUE and the content of the DataRegister is equal to X'10',

then set DLE_Mask to X'80' and enter the DATA state.

DataOctet
If ReceiveError is FALSE and DataAvailable is TRUE and Index is less than DataLength and the content of DataRegister is not equal to X'10', X'11', or X'13',

then perform a bitwise AND of the ones-complement of the DLE_Mask and the contents of DataRegister; accumulate the result into DataCRC; save the results at InputBuffer[Index]; increment Index by 1; set DataAvailable to FALSE; set DLE_Mask to X'00'; and enter the DATA state.

CRC1

If ReceiveError is FALSE and DataAvailable is TRUE and Index is equal to DataLength and the content of DataRegister is not equal to X'10', X'11', or X'13',

then perform a bitwise AND of the ones-complement of the DLE_Mask and the contents of DataRegister; accumulate the result into DataCRC; increment Index by 1; set DataAvailable to FALSE; set DLE_Mask to X'00'; and enter the DATA state.

CRC2

If ReceiveError is FALSE and DataAvailable is TRUE and Index is equal to DataLength plus 1 and the content of DataRegister is not equal to X'10', X'11', or X'13',

then perform a bitwise AND of the ones-complement of the DLE_Mask and the contents of DataRegister; accumulate the result into DataCRC; set DataAvailable to FALSE; set DLE_Mask to X'00'; and enter the DATA_CRC state.

10.4.7.6 DATA_CRC

In the DATA_CRC state, the node validates the CRC on the frame data.

BadCRC

If the value of DataCRC is not X'F0B8',

then set ReceivedInvalidFrame to TRUE to indicate that an error has occurred during the reception of a frame, and enter the IDLE state to wait for the start of the next frame.

GoodCRC

If the value of DataCRC is X'F0B8',

then set ReceivedValidFrame to TRUE to indicate the complete reception of a valid frame, and enter the IDLE state to wait for the start of the next frame.

10.4.8 Data Link Connection Establishment and Termination Procedures

This subclause provides an overview of the protocol for establishing and terminating PTP connections. The details for this protocol are defined by the Connection State Machine in 10.4.9.

Upon establishment of a physical connection between BACnet devices, the calling device shall transmit the seven character ANSI X3.4 trigger sequence "BACnet<CR>", where "<CR>" denotes the ANSI X3.4 character X'0D', to inform the answering device that it wishes to establish a BACnet PTP connection. The answering device shall then transmit a Connect Request frame. The calling device shall respond by transmitting a Connect Response frame including a password, if password protection is implemented. After successful completion of this process, including verification of the password, both devices enter the data exchange phase.

Upon completion of the data link establishment procedure, each device shall assume that the other is not yet ready to receive Data frames. When a Heartbeat XON frame is received from a device, data transmission to that device may begin. Upon completion of the data link establishment procedure and when each device is ready to receive Data frames, it shall immediately transmit a Heartbeat XON frame.

When either device wishes to terminate an active PTP connection, it shall transmit a Disconnect Request frame indicating the reason for the disconnection. The peer device shall respond by transmitting a Disconnect Response frame to acknowledge the request. Both devices shall then notify their respective network layers of data link termination.

If no Disconnect Response is received in reply to the Disconnect Request, an assumption is made that the request was not received by the peer device. The Disconnect Request is retransmitted up to three times. If a Disconnect Response is not received after the third retry, the connection is unilaterally terminated.

If, following transmission of a Disconnect Request, a device receives a Disconnect Request frame from the peer device, the device shall respond by transmitting a Disconnect Response frame and terminating the connection.

If, following transmission of a Disconnect Request, a device receives a Data frame from the peer device, the device shall not acknowledge the Data frame. Thus, a Disconnect Request is an attempt to terminate the connection in an orderly manner, but it is not negotiable. Once a Disconnect Request has been made, the connection shall be terminated. If the peer device needs to continue the communication, a new connection must be established.

10.4.9 Connection State Machine

The operation of the connection establishment state machine is described in this subclause and is depicted in Figure 10-2. The state machine models the actions taken to establish a BACnet PTP data link between two devices and includes the actions required for both the calling and answering devices.

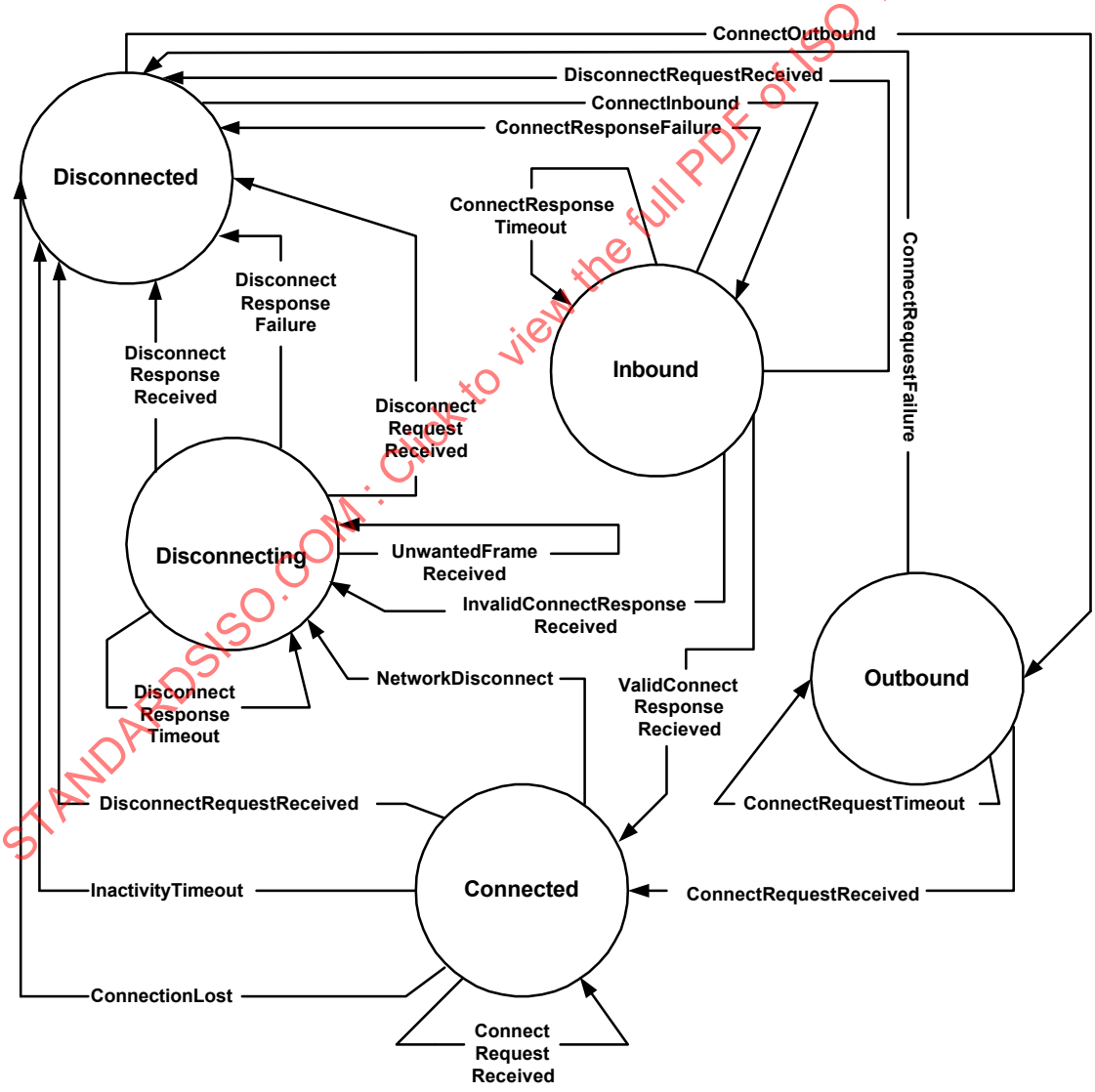


Figure 10-2. Point-To-Point Connection State Machine

10.4.9.1 DISCONNECTED

In this state, the device waits for the network layer to initiate a PTP data link connection or for the physical layer to indicate the occurrence of a physical layer connection.

ConnectOutbound

If a DL-CONNECT.request is received,

then establish a physical connection; transmit the "BACnet<CR>" trigger sequence; set RetryCount to zero; set ResponseTimer to zero; and enter the OUTBOUND state.

ConnectInbound

If a physical layer connection has been made and the "BACnet<CR>" trigger sequence is received,

then call SendFrame to transmit a Connect Request frame; set RetryCount to zero; set ResponseTimer to zero; and enter the INBOUND state.

10.4.9.2 OUTBOUND

In this state, the network layer has issued a request to start the data link as a caller, and the device is waiting for a Connect Request frame from the answering device.

ConnectRequestReceived

If ReceivedValidFrame is TRUE and FrameType is equal to Connect Request,

then set ReceivedValidFrame to FALSE; call SendFrame to transmit a Connect Response frame containing the password contained in the "data" parameter of the DL-CONNECT.request that initiated the connection; issue a DL-CONNECT.confirm to notify the network layer that a connection has been established; and enter the CONNECTED state.

ConnectRequestTimeout

If ResponseTimer is greater than or equal to $\mathbf{T}_{\mathbf{conn\_rqst}}$ and RetryCount is less than $\mathbf{N}_{\mathbf{retries}}$,

then set RetryCount to RetryCount + 1; retransmit the "BACnet<CR>" trigger sequence; set the ResponseTimer to zero; and enter the OUTBOUND state.

ConnectRequestFailure

If ResponseTimer is greater than or equal to $\mathbf{T}_{\mathbf{conn\_rqst}}$ and RetryCount is greater than or equal to $\mathbf{N}_{\mathbf{retries}}$,

then issue a DL-CONNECT.confirm to notify the network layer of the failure and enter the DISCONNECTED state.

10.4.9.3 INBOUND

In this state, the Connection State Machine has recognized that the calling device wishes to establish a BACnet connection, and the local device is waiting for a Connect Response frame from the calling device.

ValidConnectResponseReceived

If ReceivedValidFrame is TRUE and FrameType is equal to Connect Response and a password is not needed or a valid password is present in the data field of the frame,

then set ReceivedValidFrame to FALSE; issue a DL-CONNECT.indication to notify the network layer of the connection; and enter the CONNECTED state.

InvalidConnectResponseReceived

If ReceivedValidFrame is TRUE and FrameType is equal to Connect Response and a password is needed but not present or an invalid password is present in the data field of the frame,

then set ReceivedValidFrame to FALSE; call SendFrame to transmit a Disconnect Request frame indicating the receipt of an invalid password; set ResponseTimer to zero; set RetryCount to zero; and enter the DISCONNECTING state.

ConnectResponseTimeout
If ResponseTimer is greater than or equal to $\mathbf{T}_{conn\_rsp}$ and RetryCount is less than $\mathbf{N}_{retries}$,

then set RetryCount to RetryCount + 1; call SendFrame to transmit a Connect Request frame; set ResponseTimer to zero; and enter the INBOUND state.

ConnectResponseFailure
If ResponseTimer is greater than or equal to $\mathbf{T}_{conn\_rsp}$ and RetryCount is greater than or equal to $\mathbf{N}_{retries}$,

then enter the DISCONNECTED state.

DisconnectRequestReceived
If ReceivedValidFrame is TRUE and FrameType is equal to Disconnect Request,

then set ReceivedValidFrame to FALSE; call SendFrame to transmit a Disconnect Response frame; and enter the DISCONNECTED state.

10.4.9.4 CONNECTED

In this state, the connection procedure has been completed, and the two devices may exchange BACnet PDUs. The data link remains in this state until termination.

NetworkDisconnect
If a DL-DISCONNECT.request is received,

then call SendFrame to transmit a Disconnect Request frame; set ResponseTimer to zero; issue a DL-DISCONNECT.confirm to notify the network layer of the disconnection; set RetryCount to zero; and enter the DISCONNECTING state.

DisconnectRequestReceived
If ReceivedValidFrame is TRUE and FrameType is equal to Disconnect Request,

then set ReceivedValidFrame to FALSE; call SendFrame to transmit a Disconnect Response frame; issue a DL-DISCONNECT.indication to notify the network layer of the disconnection; and enter the DISCONNECTED state.

ConnectRequestReceived
If ReceivedValidFrame is TRUE and FrameType is equal to Connect Request,

then set ReceivedValidFrame to FALSE; call SendFrame to transmit a Connect Response frame; issue a DL-CONNECT.indication to notify the network layer of the connection; and enter the CONNECTED state.

InactivityTimeout
If InactivityTimer is greater than $T_{inactivity}$ and ReceivedValidFrame is FALSE,

then issue a DL-DISCONNECT.indication to notify the network layer of the disconnection and enter the DISCONNECTED state.

ConnectionLost
If the physical connection has been terminated, e.g., due to loss of carrier,

then issue a DL-DISCONNECT.indication to notify the network layer of the disconnection and enter the DISCONNECTED state.

10.4.9.5 DISCONNECTING

In this state, the network layer has requested termination of the data link. The device is waiting for a Disconnect Response frame from the peer device.

DisconnectResponseReceived
If ReceivedValidFrame is TRUE and FrameType is equal to Disconnect Response,

then set ReceivedValidFrame to FALSE and enter the DISCONNECTED state.

DisconnectRequestReceived
If ReceivedValidFrame is TRUE and FrameType is equal to Disconnect Request,

then set ReceivedValidFrame to FALSE; call SendFrame to transmit a Disconnect Response frame; and enter the DISCONNECTED state.

UnwantedFrameReceived
If ReceivedValidFrame is TRUE and FrameType is not equal to either Disconnect Response or Disconnect Request,

then set ReceivedValidFrame to FALSE and enter the DISCONNECTING state. (Note that ResponseTimer is not reset in this case.)

DisconnectResponseTimeout
If ResponseTimer is greater than $\mathbf{T}_{\mathbf{response}}$ and ReceivedValidFrame is FALSE and RetryCount is less than $\mathbf{N}_{\mathbf{retries}}$,

then increment RetryCount; call SendFrame to transmit a Disconnect Request frame; set ResponseTimer to zero; and enter the DISCONNECTING state.

DisconnectResponseFailure
If ResponseTimer is greater than or equal to $\mathbf{T}_{\mathbf{response}}$ and ReceivedValidFrame is FALSE and RetryCount is greater than or equal to $\mathbf{N}_{\mathbf{retries}}$,

then enter the DISCONNECTED state.

10.4.10 Transmission State Machine

The operation of the Transmission State Machine is described in this subclause and is depicted in Figure 10-3. The state machine models the actions taken to transmit data frames and receive corresponding acknowledgments.

10.4.10.1 TRANSMIT IDLE

In this state, the transmitter is waiting for the data link to be established between the local device and the peer device. The transmitter waits to be notified that a peer device is ready to communicate.

ConnectionEstablishedXON
If the Connection State Machine is in the CONNECTED state and ReceptionBlocked is equal to NOT_BLOCKED,

then call SendFrame to transmit a HeartbeatXON frame; set TxSequenceNumber to zero; set HeartbeatTimer to zero; and enter the TRANSMIT BLOCKED state.

ConnectionEstablishedXOFF
If the Connection State Machine is in the CONNECTED state and ReceptionBlocked is equal to ALMOST_BLOCKED or BLOCKED,

then call SendFrame to transmit a HeartbeatXOFF frame; set TxSequenceNumber to zero; and enter the TRANSMIT BLOCKED state.

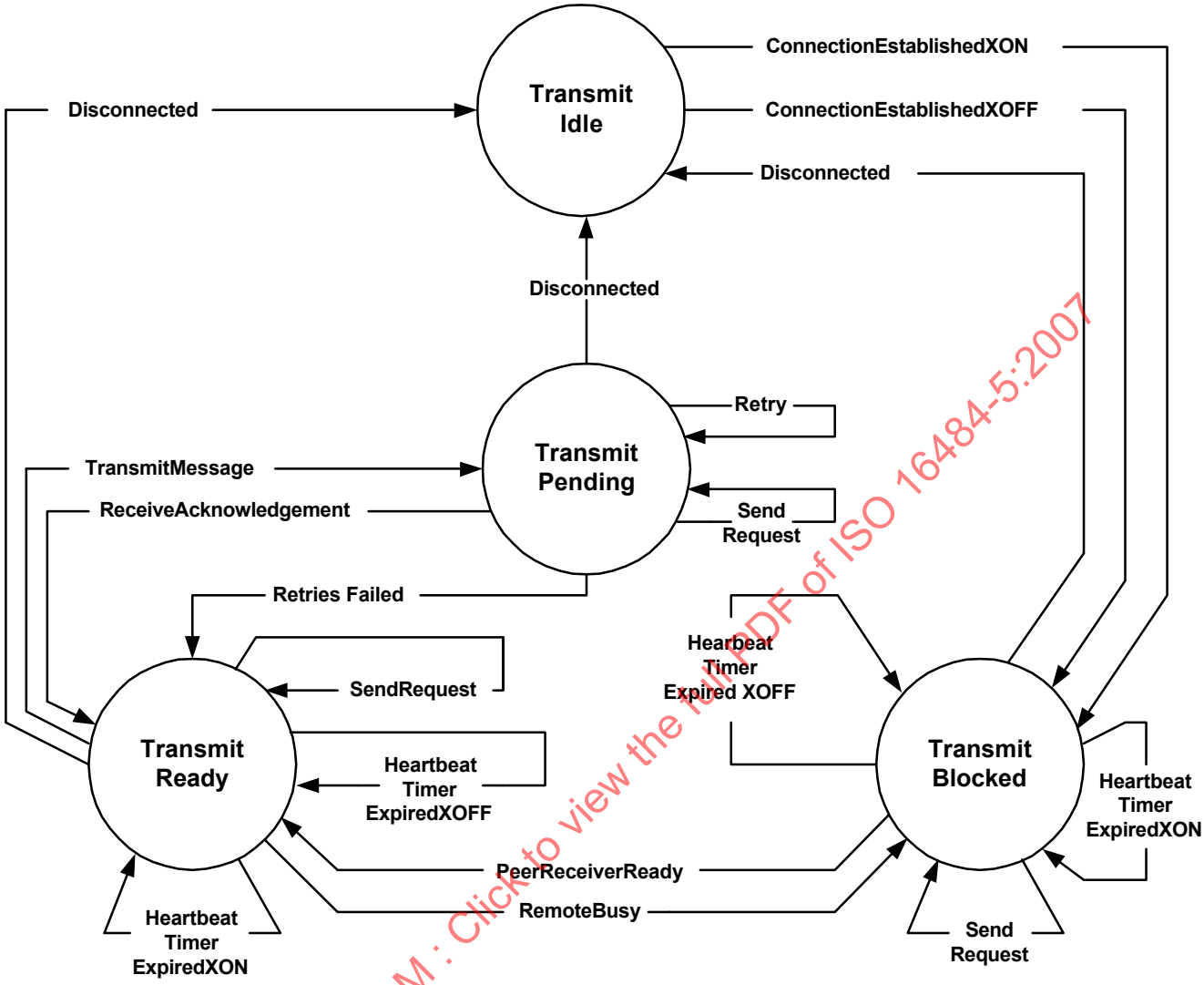


Figure 10-3. Point-To-Point Transmission State Machine.

10.4.10.2 TRANSMIT BLOCKED

In this state, the peer device has indicated that it is not ready to receive data frames. The local device may have data ready to transmit. The local device periodically transmits a Heartbeat frame to maintain the data-link and waits for the peer device to become ready to receive data or for the termination of the data link.

SendRequest

If a DL-UNITDATA.request primitive is received,

then queue the request for later transmission and enter the TRANSMIT BLOCKED state.

PeerReceiverReady

If TransmissionBlocked is equal to FALSE,

then enter the TRANSMIT READY state.

Disconnected

If the Connection State Machine is in the DISCONNECTED state,

then enter the TRANSMIT IDLE state.

HeartbeatTimerExpiredXON
If HeartbeatTimer is greater than $\mathbf{T}_{\mathbf{heartbeat}}$ and ReceptionBlocked is equal to NOT_BLOCKED,

then call SendFrame to transmit a HeartbeatXON frame; set HeartbeatTimer to zero; and enter the TRANSMIT BLOCKED state.

HeartbeatTimerExpiredXOFF
If HeartbeatTimer is greater than $\mathbf{T}_{\mathbf{heartbeat}}$ and ReceptionBlocked is equal to BLOCKED or ALMOST_BLOCKED,

then call SendFrame to transmit a HeartbeatXOFF frame; set HeartbeatTimer to zero; and enter the TRANSMIT BLOCKED state.

10.4.10.3 TRANSMIT READY

In this state, the peer device has indicated its readiness to receive Data frames, but the local device has no data ready to transmit. The local device periodically transmits a Heartbeat frame to maintain the data link and waits for a local request to transmit data or for the termination of the data link.

Disconnected
If the Connection State Machine is in the DISCONNECTED state,

then enter the TRANSMIT IDLE state.

SendRequest
If a DL-UNITDATA.request primitive is received,

then queue the request for later transmission and enter the TRANSMIT READY state.

TransmitMessage
If the transmit queue is not empty and TransmissionBlocked is equal to FALSE,

then call SendFrame to transmit the frame at the head of the queue using a Data frame type (Data 0 or Data 1) that indicates TxSequenceNumber; set RetryCount to zero; set ResponseTimer to zero; and enter the TRANSMIT PENDING state.

RemoteBusy
If TransmissionBlocked is equal to TRUE,

then enter the TRANSMIT BLOCKED state.

HeartbeatTimerExpiredXON
If HeartbeatTimer is greater than $\mathbf{T}_{\mathbf{heartbeat}}$ and ReceptionBlocked is equal to NOT_BLOCKED,

then call SendFrame to transmit a HeartbeatXON frame; set HeartbeatTimer to zero; and enter the TRANSMIT READY state.

HeartbeatTimerExpiredXOFF
If HeartbeatTimer is greater than $\mathbf{T}_{\mathbf{heartbeat}}$ and ReceptionBlocked is equal to BLOCKED or ALMOST_BLOCKED,

then call SendFrame to transmit a HeartbeatXOFF frame; set HeartbeatTimer to zero; and enter the TRANSMIT READY state.

10.4.10.4 TRANSMIT PENDING

In this state, the local device has transmitted a data frame to the peer device and is waiting for an acknowledgment from the peer device.

Disconnected

If the Connection State Machine is in the DISCONNECTED state,

then enter the TRANSMIT IDLE state.

SendRequest

If a DL-UNITDATA.request primitive is received,

then queue the request for later transmission and enter the TRANSMIT PENDING state.

ReceiveAcknowledgment

If TxSequenceNumber is equal to 0 and Ack0Received is equal to TRUE or if TxSequenceNumber is equal to 1 and Ack1Received is equal to TRUE,

then set TxSequenceNumber = 1 - TxSequenceNumber; set Ack0Received to FALSE; set Ack1Received to FALSE; and enter the TRANSMIT READY state.

Retry

If RetryCount is less than $\mathbf{N}_{\mathbf{retries}}$ and either

(a) TxSequenceNumber is equal to 0 and Nak0Received is equal to TRUE or

(b) TxSequenceNumber is equal to 1 and Nak1Received is equal to TRUE or

(c) ResponseTimer is greater than $\mathbf{T}_{\mathbf{response}}$,

then set RetryCount to RetryCount + 1; set Nak0Received to FALSE; set Nak1Received to FALSE; set ResponseTimer to zero; call SendFrame to retransmit the Data frame; and enter the TRANSMIT PENDING state.

RetriesFailed

If RetryCount is equal to $\mathbf{N}_{\mathbf{retries}}$ and either

(a) TxSequenceNumber is equal to 0 and Nak0Received is equal to TRUE or

(b) TxSequenceNumber is equal to 1 and Nak1Received is equal to TRUE or

(c) ResponseTimer is greater than $\mathbf{T}_{\mathbf{response}}$,

then set RetryCount to 0; set Nak0Received to FALSE; set Nak1Received to FALSE; set ResponseTimer to zero; and enter the TRANSMIT READY state.

10.4.11 Reception State Machine

The operation of the Reception State Machine is described in this subclause and is depicted in Figure 10-4.

10.4.11.1 RECEIVE IDLE

In this state, the receiver is waiting for the data link to be established between the local device and the peer device. The receiver waits to be notified that a peer device is ready to communicate.

ConnectionEstablished

If the Connection State Machine is in the CONNECTED state,

then set RxSequenceNumber to zero and enter the RECEIVE READY state.

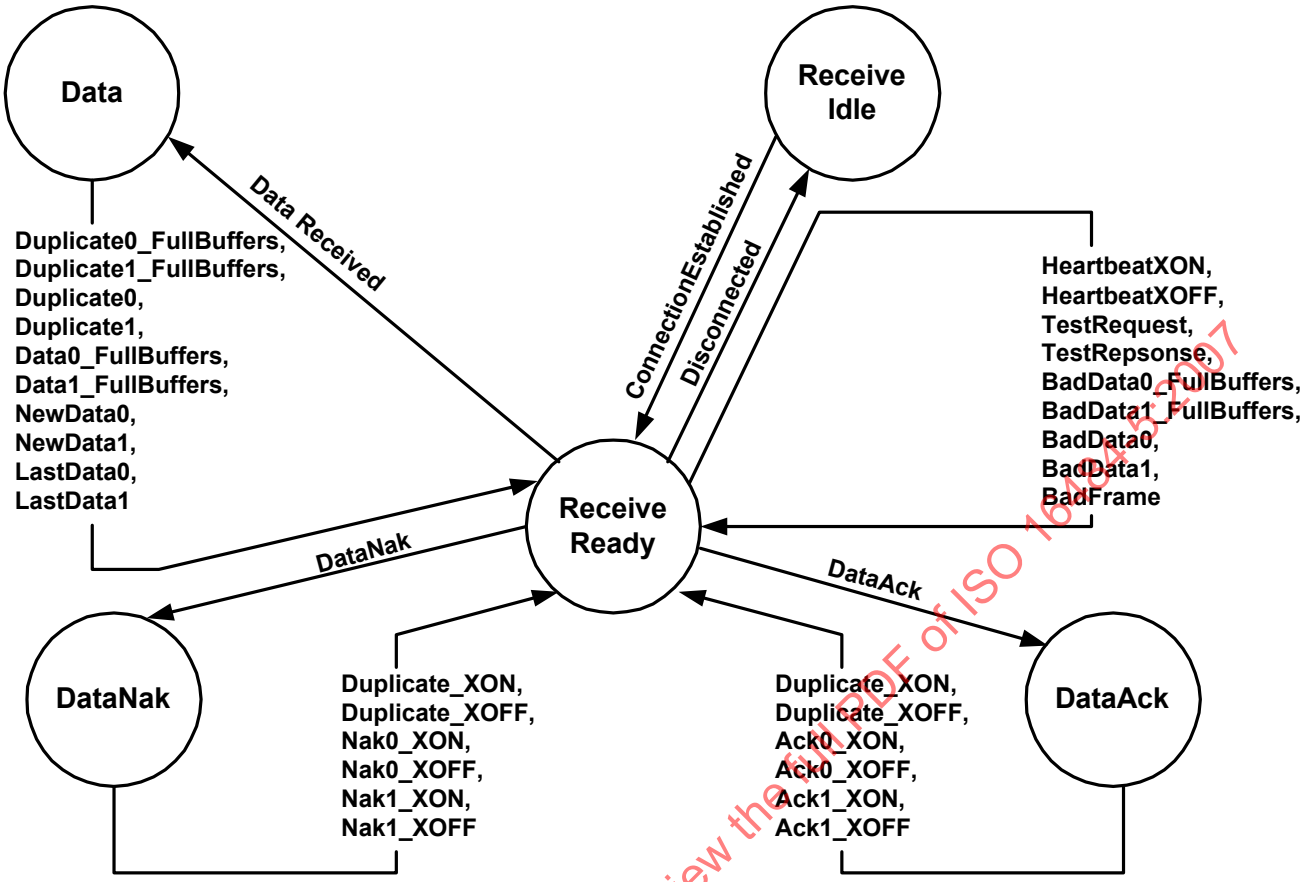


Figure 10-4. Point-To-Point Reception State Machine.

10.4.11.2 RECEIVE READY

In this state, the device is ready to receive frames from the peer device.

DataReceived

If ReceivedValidFrame is TRUE and FrameType is equal to Data 0 or Data 1,

then set ReceivedValidFrame to FALSE; set InactivityTimer to zero; and enter the DATA state.

DataAck

If ReceivedValidFrame is TRUE and FrameType is equal to Data Ack 0 XOFF, Data Ack 0 XON, Data ACK 1 XOFF, or Data Ack 1 XON,

then set ReceivedValidFrame to FALSE; set InactivityTimer to zero; and enter the DATA ACK state.

DataNak

If ReceivedValidFrame is TRUE and FrameType is equal to Data Nak 0 XOFF, Data Nak 0 XON, Data Nak 1 XOFF, or Data Nak 1 XON,

then set ReceivedValidFrame to FALSE; set InactivityTimer to zero; and enter the DATA NAK state.

HeartbeatXON

If ReceivedValidFrame is TRUE and FrameType is equal to Heartbeat XON,

then set TransmissionBlocked to FALSE; set InactivityTimer to zero; set ReceivedValidFrame to FALSE; and enter the RECEIVE READY state.

HeartbeatXOFF

If ReceivedValidFrame is TRUE and FrameType is equal to Heartbeat XOFF,

then set TransmissionBlocked to TRUE; set InactivityTimer to zero; set ReceivedValidFrame to FALSE; and enter the RECEIVE READY state.

TestRequest

If ReceivedValidFrame is TRUE and FrameType is equal to Test_Request,

then call SendFrame to transmit the Test_Response; set InactivityTimer to zero; set ReceivedValidFrame to FALSE; and enter the RECEIVE READY state.

TestResponse

If ReceivedValidFrame is TRUE and FrameType is equal to Test_Response,

then issue a DL-UNITDATA.indication conveying the Test_Response data; set InactivityTimer to zero; set ReceivedValidFrame to FALSE; and enter the RECEIVE READY state.

BadData0_FullBuffers

If ReceivedInvalidFrame is TRUE and FrameType is equal to Data 0 and ReceptionBlocked is equal to BLOCKED,

then discard the frame; set InactivityTimer to zero; call SendFrame to transmit a Data Nak 0 XOFF frame; set ReceivedInvalidFrame to FALSE; and enter the RECEIVE READY state.

BadData1_FullBuffers

If ReceivedInvalidFrame is TRUE and FrameType is equal to Data 1 and ReceptionBlocked is equal to BLOCKED,

then discard the frame; set InactivityTimer to zero; call SendFrame to transmit a Data Nak 1 XOFF frame; set ReceivedInvalidFrame to FALSE; and enter the RECEIVE READY state.

BadData0

If ReceivedInvalidFrame is TRUE and FrameType is equal to Data 0 and ReceptionBlocked is equal to NOT_BLOCKED or ALMOST_BLOCKED,

then discard the frame; set InactivityTimer to zero; call SendFrame to transmit a Data Nak 0 XON frame; set ReceivedInvalidFrame to FALSE; and enter the RECEIVE READY state.

BadData1

If ReceivedInvalidFrame is TRUE and FrameType is equal to Data 1 and ReceptionBlocked is equal to NOT_BLOCKED or ALMOST_BLOCKED,

then discard the frame; set InactivityTimer to zero; call SendFrame to transmit a Data Nak 1 XON frame; set ReceivedInvalidFrame to FALSE; and enter the RECEIVE READY state.

BadFrame

If ReceivedInvalidFrame is TRUE and FrameType is not equal to either Data 0 or Data 1,

then discard the frame; set InactivityTimer to zero; set ReceivedInvalidFrame to FALSE; and enter the RECEIVE READY state.

Disconnected
If the Connection State Machine is in the DISCONNECTED state,

then enter the RECEIVE IDLE state.

10.4.11.3 DATA

In this state the device has received a Data frame for processing.

Duplicate0_FullBuffers
If FrameType is equal to Data 0 and RxSequenceNumber is equal to 1 and ReceptionBlocked is equal to BLOCKED,

then discard the frame as a duplicate; call SendFrame to transmit a Data Ack 0 XOFF frame; and enter the RECEIVE READY state.

Duplicate1_FullBuffers
If FrameType is equal to Data 1 and RxSequenceNumber is equal to 0 and ReceptionBlocked is equal to BLOCKED,

then discard the frame as a duplicate; call SendFrame to transmit a Data Ack 1 XOFF frame; and enter the RECEIVE READY state.

Duplicate0
If FrameType is equal to Data 0 and RxSequenceNumber is equal to 1 and ReceptionBlocked is equal to NOT_BLOCKED or ALMOST_BLOCKED,

then discard the frame as a duplicate; call SendFrame to transmit a Data Ack 0 XON frame; and enter the RECEIVE READY state.

Duplicate1
If FrameType is equal to Data 1 and RxSequenceNumber is equal to 0 and ReceptionBlocked is equal to NOT_BLOCKED or ALMOST_BLOCKED,

then discard the frame as a duplicate; call SendFrame to transmit a Data Ack 1 XON frame; and enter the RECEIVE READY state.

Data0_FullBuffers
If FrameType is equal to Data 0 and RxSequenceNumber is equal to 0 and ReceptionBlocked is equal to BLOCKED,

then discard the frame for lack of space; call SendFrame to transmit a Data Nak 0 XOFF frame; and enter the RECEIVE READY state.

Data1_FullBuffers
If FrameType is equal to Data 1 and RxSequenceNumber is equal to 1 and ReceptionBlocked is equal to BLOCKED,

then discard the frame for lack of space; call SendFrame to transmit a Data Nak 1 XOFF frame; and enter the RECEIVE READY state.

NewData0
If FrameType is equal to Data 0 and RxSequenceNumber is equal to 0 and ReceptionBlocked is equal to NOT_BLOCKED,

then issue a DL-UNITDATA.indication conveying the data; call SendFrame to transmit a Data Ack 0 XON frame; set RxSequenceNumber to 1; and enter the RECEIVE READY state.

NewData1
If FrameType is equal to Data 1 and RxSequenceNumber is equal to 1 and ReceptionBlocked is equal to NOT_BLOCKED,

then issue a DL-UNITDATA.indication conveying the data; call SendFrame to transmit a Data Ack 1 XON frame; set RxSequenceNumber to 0; and enter the RECEIVE READY state.

LastData0
If FrameType is equal to Data 0 and RxSequenceNumber is equal to 0 and ReceptionBlocked is equal to ALMOST_BLOCKED,

then issue a DL-UNITDATA.indication conveying the data; call SendFrame to transmit a Data Ack 0 XOFF frame; set RxSequenceNumber to 1; and enter the RECEIVE READY state.

LastData1
If FrameType is equal to Data 1 and RxSequenceNumber is equal to 1 and ReceptionBlocked is equal to ALMOST_BLOCKED,

then issue a DL-UNITDATA.indication conveying the data; call SendFrame to transmit a Data Ack 1 XOFF frame; set RxSequenceNumber to 0; and enter the RECEIVE READY state.

10.4.11.4 DATA ACK

In this state the device has received a Data Ack frame for processing.

Duplicate_XON
If FrameType is equal to Data Ack 0 XON and TxSequenceNumber is equal to 1, or if FrameType is equal to Data Ack 1 XON and TxSequenceNumber is equal to 0,

then set TransmissionBlocked to FALSE and enter the RECEIVE READY state.

Duplicate_XOFF
If FrameType is equal to Data Ack 0 XOFF and TxSequenceNumber is equal to 1, or if FrameType is equal to Data Ack 1 XOFF and TxSequenceNumber is equal to 0,

then set TransmissionBlocked to TRUE and enter the RECEIVE READY state.

Ack0_XON
If FrameType is equal to Data Ack 0 XON and TxSequenceNumber is equal to 0,

then set Ack0Received to TRUE; set TransmissionBlocked to FALSE; and enter the RECEIVE READY state.

Ack0_XOFF
If FrameType is equal to Data Ack 0 XOFF and TxSequenceNumber is equal to 0,

then set Ack0Received to TRUE; set TransmissionBlocked to TRUE; and enter the RECEIVE READY state.

Ack1_XON
If FrameType is equal to Data Ack 1 XON and TxSequenceNumber is equal to 1,

then set Ack1Received to TRUE; set TransmissionBlocked to FALSE; and enter the RECEIVE READY state.

Ack1_XOFF
If FrameType is equal to Data Ack 1 XOFF and TxSequenceNumber is equal to 1,

then set Ack1Received to TRUE; set TransmissionBlocked to TRUE; and enter the RECEIVE READY state.

10.4.11.5 DATA NAK

In this state the device has received a Data Nak frame for processing.

Duplicate_XON
: If FrameType is equal to Data Nak 0 XON and TxSequenceNumber is equal to 1, or if FrameType is equal to Data Nak 1 XON and TxSequenceNumber is equal to 0,

  then set TransmissionBlocked to FALSE and enter the RECEIVE READY state.

Duplicate_XOFF
: If FrameType is equal to Data Nak 0 XOFF and TxSequenceNumber is equal to 1, or if FrameType is equal to Data Nak 1 XOFF and TxSequenceNumber is equal to 0,

  then set TransmissionBlocked to TRUE and enter the RECEIVE READY state.

Nak0_XON
: If FrameType is equal to Data Nak 0 XON and TxSequenceNumber is equal to 0,

  then set Nak0Received to TRUE; set TransmissionBlocked to FALSE; and enter the RECEIVE READY state.

Nak0_XOFF
: If FrameType is equal to Data Nak 0 XOFF and TxSequenceNumber is equal to 0,

  then set Nak0Received to TRUE; set TransmissionBlocked to TRUE; and enter the RECEIVE READY state.

Nak1_XON
: If FrameType is equal to Data Nak 1 XON and TxSequenceNumber is equal to 1,

  then set Nak1Received to TRUE; set TransmissionBlocked to FALSE; and enter the RECEIVE READY state.

Nak1_XOFF
: If FrameType is equal to Data Nak 1 XOFF and TxSequenceNumber is equal to 1,

  then set Nak1Received to TRUE; set TransmissionBlocked to TRUE; and enter the RECEIVE READY state.

11 DATA LINK/PHYSICAL LAYERS: EIA/CEA-709.1 ("LonTalk") LAN

This clause describes the transport of BACnet LSDUs using the services of the LonTalk protocol described in *EIA/CEA-709.1-B-2002 Control Network Protocol Specification*. EIA/CEA-709.1-B-2002, as amended and extended by the Electronic Industries Alliance, is deemed to be included in this standard by reference. Persons desiring to implement BACnet in products containing the LonTalk Protocol may obtain an OEM license to do so, without cost, by contacting Echelon Corporation, San Jose, California.

11.1 The Use of ISO 8802-2 Logical Link Control (LLC)

Standard BACnet networks may pass BACnet link service data units (LSDUs) using the data link services of ISO 8802-2 LLC. A BACnet LSDU consists of an NPDU constructed as described in Clause 6. BACnet devices using LonTalk LAN technology shall conform to the requirements of LLC Class I, subject to the constraints specified in this clause. Class I LLC service consists of Type 1 LLC - Unacknowledged Connectionless-Mode service. LLC parameters shall be conveyed using the DL-UNITDATA primitives as described in the referenced standards.

In a LonTalk implementation, BACnet DL-UNITDATA primitives are mapped into the LonTalk Application Layer Interface. The mapping of these primitives onto the LonTalk Application layer primitives is described in 11.3.

11.2 Parameters Required by the LLC Primitives

The DL-UNITDATA primitive requires source address, destination address, data, and priority parameters. Each source and destination address consists of a LonTalk address, link service access point (LSAP), and a message code (MC). The LonTalk address is a variable-length value determined by the configuration of the BACnet device, and the MC used to indicate a BACnet frame is the single-octet value X'4E'. Since the LonTalk message code identifies the BACnet network layer, the LSAP is not used. The data parameter is the NPDU from the network layer.

11.3 Mapping the LLC Services to the LonTalk Application Layer

The Type 1 Unacknowledged Connectionless LLC service, DL_UNITDATA.request shall map onto the LonTalk msg_send request primitive, while the DL_UNITDATA.indication shall map to the LonTalk msg_receive request primitive.

An LPDU longer than 228 octets cannot be conveyed via LonTalk.

11.4 Parameters Required by the Application Layer Primitives

The LonTalk Application layer primitives are msg_send and msg_receive. These convey the encoded LLC data using the destination LonTalk address described above in conjunction with the BACnet message code. The DL_UNITDATA.request primitive contains the following parameters:

```
DL_UNITDATA.request (
                destination_address,
                data,
                priority,
                message_code
                )
```

The 'destination_address' consists of any form of a LonTalk address except address format 2B (which is used exclusively for multicast acknowledgments and multicast responses). See Figure 6-4. The 'data' parameter specifies the LSDU to be transferred. The 'priority' parameter conveys the priority specified for the data unit. Any BACnet priority other than "Normal message" shall be sent using the LonTalk priority mechanism. The 'message_code' parameter shall be X'4E' for BACnet LPDUs.

LonTalk Authentication is not supported in BACnet.

LonTalk "UNACKD" and "UNACKD_RPT" are the only LonTalk services that shall be allowed within BACnet. The choice between these two LonTalk services and the repeat count for the "UNACKD_RPT" service shall be considered a local matter.

The DL_UNITDATA.indication primitive contains the following parameters:

DL_UNITDATA.indication (
source_address,
destination_address,
length,
data,
message_code,
priority
)

Except as noted below, the parameters in DL_UNITDATA.indication convey the same information as the parameters in DL_UNITDATA.request.

The 'source_address' always consists of address format 2A.

The 'length' indicates the number of octets contained in the 'data' parameter.

Figure 11-1 illustrates the format of a MPDU on a LonTalk BACnet network destined for a device on the same LonTalk BACnet network.

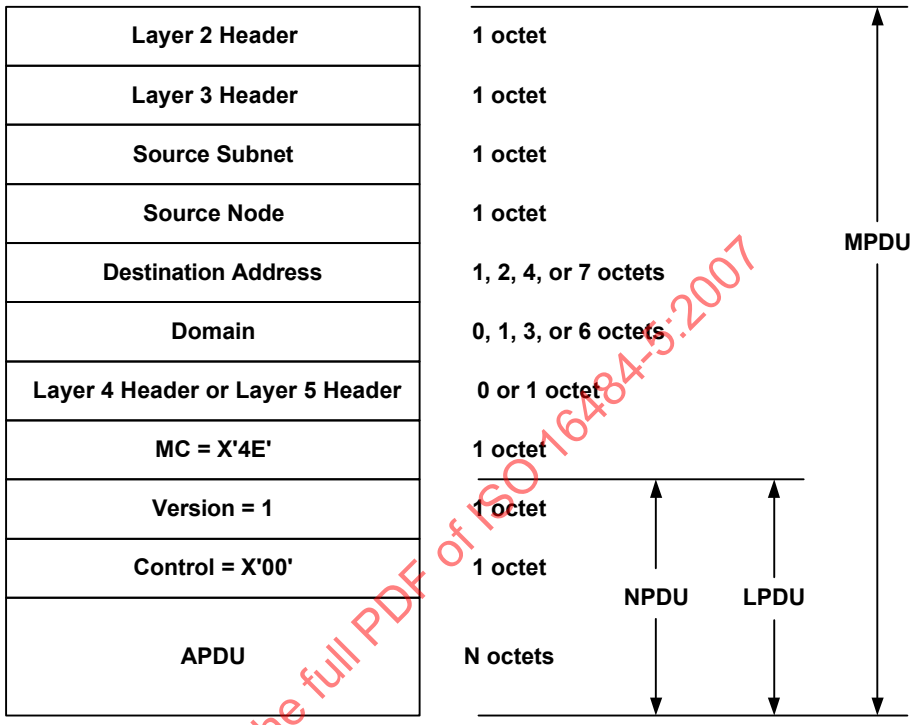


Figure 11-1. Format of an MPDU on a LonTalk network destined for a device on the same LonTalk BACnet network.

11.5 Physical Media

Any of the "Standard Channel Types" defined in the *LonMark® Layer 1-6 Interoperability Guidelines* is acceptable. Transceivers built for this network technology shall follow the guidelines specified in the *LonMark® Layer 1-6 Interoperability Guidelines*. The most recent version of the *LonMark® Layer 1-6 Interoperability Guidelines* as released by the LonMark® Interoperability Association (currently version 3.3) shall apply.

12 MODELING CONTROL DEVICES AS A COLLECTION OF OBJECTS

The data structures used in a device to store information are a local matter. In order to exchange that information with another device using this protocol, there must be a "network-visible" representation of the information that is standardized. An object-oriented approach has been adopted to provide this network-visible representation. This clause defines a set of standard object types. These object types define an abstract data structure that provides a framework for building the application layer services. The application layer services are designed, in part, to access and manipulate the properties of these standard object types. Mapping the effect of these services to the real data structures used in the device is a local matter. The number of instances of a particular object type that a device will support is also a local matter.

All objects are referenced by their Object_Identifier property. Each object within a single BACnet Device shall have a unique value for the Object_Identifier property. When combined with the system-wide unique Object_Identifier of the BACnet Device, this provides a mechanism for referencing every object in the control system network. No object shall have an Object_Identifier with an instance number of 4194303. Object properties that contain BACnetObjectIdentifiers may use 4194303 to indicate that the property is not initialized.

Not all object types defined in this standard need to be supported in order to conform to the standard. In addition, some properties of particular object types are optional. At the beginning of each standard object type specification that follows is a summary of the properties of the object type. The summary includes the property identifier, the datatype of the property, and one of the following : **O**, **R**, **W**

where **O** indicates that the property is optional,
R indicates that the property is required to be present and readable using BACnet services,
W indicates that the property is required to be present, readable, and writable using BACnet services.

When a property is designated as required or **R**, this shall mean that the property is required to be present in all BACnet standard objects of that type. When a property is designated as optional or **O**, this shall mean that the property is not required to be present in all standard BACnet objects of that type. The value of **R** or **O** properties may be examined through the use of one or more of the ReadProperty services defined in this standard. Such **R** or **O** properties may also be writable at the implementor's option unless specifically prohibited in the text describing that particular standard object's property. When a property is designated as writable or **W**, this shall mean that the property is required to be present in all BACnet standard objects of that type and that the value of the property can be changed through the use of one or more of the WriteProperty services defined in this standard. The value of **W** properties may be examined through the use of one or more of the ReadProperty services defined in this standard. An **O** property, if present in a particular object, is not required to be writable unless specifically identified as such in the text describing that particular standard object's property.

In some devices, property values may be stored internally in a different form than indicated by the property datatype. For example, real numbers may be stored internally as integers. This may result in the situation where a property value is changed by one of the WriteProperty services but a subsequent read returns a slightly different value. This behavior is acceptable as long as a "best effort" is made to store the written value specified.

It is intended that the collection of object types and their properties defined in this standard be comprehensive, but implementors are free to define additional nonstandard object types or additional nonstandard properties of standard object types. This is the principal means for extending the standard as control technology develops. Innovative changes can be accommodated without waiting for changes in the standard. This extensibility could also be used to adapt this standard to other types of building services. See 23.3 and 23.4.

Nonstandard object types are required to support the following properties:

- Object_Identifier BACnetObjectIdentifier
- Object_Name CharacterString
- Object_Type BACnetObjectType

These properties shall be implemented to behave as they would when present in standard BACnet objects. This means that the Object_Identifier and Object_Name properties shall be unique within the BACnet device that maintains them. The Object_Name string shall be at least one character in length and shall consist only of printable characters.

A BACnet standard object shall support all required properties specified in the standard. It may support, in addition to these properties, any optional properties specified in the standard or properties not defined in the standard. A required property shall function as specified in the standard for each object of that type. If properties that are defined as optional in the standard are supported, then they shall function as specified in the standard. A required property shall be present in all objects of that type. An optional property, if present in one object of a given type, need not be present in all objects of that type. A supported property, whether required or optional, shall return the datatype specified in the standard. A supported property, whether required or optional, is not required to be able to return the entire range of values for a datatype unless otherwise specified in the property description. Supported properties, whether required or optional, which do not return the entire range of values for a datatype when read or which restrict the range of values that may be written to the property, shall specify those restrictions for each such property in the protocol implementation conformance statement (PICS).

Some of the properties of certain BACnet objects need to represent a collection of data elements of the same type, rather than a single primitive data value or a complex datatype constructed from other datatypes. In some instances, the size of this collection of data elements is fixed, while in other instances the number of elements may be variable. In some cases the elements may need to be accessed individually or their order may be important. BACnet provides two forms of datatypes for properties that represent a collection of data elements of the same type: "BACnetARRAY" and "List of."

A "BACnetARRAY" datatype is a structured datatype consisting of an ordered sequence of data elements, each having the same datatype. The components of an array property may be individually accessed (read or written) using an "array index," which is an unsigned integer value. An index of 0 (zero) shall specify that the count of the number of data elements be returned. If the array index is omitted, it means that all of the elements of the array are to be accessed. An array index N, greater than zero, shall specify the Nth element in the sequence. When array properties are used in BACnet objects, the notation "BACnetARRAY[N] of datatype" shall mean an ordered sequence of N data elements, each of which has that datatype. If the size of an array may be changed by writing to the array, then array element 0 shall be writable. If the value of array element 0 is decreased, the array shall be truncated and the elements of the array with an index greater than the new value of array element 0 are deleted. If the value of array element 0 is increased, the new elements of the array, those with an index greater than the old value of array element 0, shall be created; the values that are assigned to those elements shall be a local matter except where otherwise specified. Where the size of an array is allowed to be changed, writing the entire array as a single property with a different number of elements shall cause the array size to be changed. An attempt to write to an array element with an index greater than the size of the array shall result in an error and shall not cause the array to grow to accommodate the element. Arrays whose sizes are fixed by the Standard shall not be resizable.

A "List of" datatype is a structured datatype consisting of a sequence of zero or more data elements, each having the same datatype. The length of each "List of" may be variable. Unless specified for a particular use, no maximum size should be assumed for any "List of" implementation. The notation "List of datatype" shall mean a sequence of zero or more data elements, each of which has the indicated type**.**

The difference between a "BACnetARRAY" property and a "List of" property is that the elements of the array can be uniquely accessed by an array index while the elements of the "List of" property cannot. Moreover, the number of elements in the BACnetARRAY may be ascertained by reading the array index 0, while the number of elements present in a "List of" property can only be determined by reading the entire property value and performing a count.

Several object types defined in this clause— the Command, Event Enrollment, Group, Loop, and Schedule— have one or more properties of type BACnetObjectPropertyReference. The property identifier component of these references may not be the special property identifiers ALL, REQUIRED, or OPTIONAL. These are reserved for use in the ReadPropertyConditional and ReadPropertyMultiple services or in services not defined in this standard.

Several object types defined in this clause have a property called "Reliability." This property is an enumerated datatype that may have different possible enumerations for different object types. The values defined below are a superset of all possible values of the Reliability property for all object types. The range of possible values returned for each specific object is defined in the appropriate object type definition.

NO_FAULT_DETECTED	The present value is reliable; that is, no other fault (enumerated below) has been detected.
NO_SENSOR	No sensor is connected to the Input object.
OVER_RANGE	The sensor connected to the Input is reading a value higher than the normal operating range. If the object is a Binary Input, this is possible when the Binary state is derived from an analog sensor or a binary input equipped with electrical loop supervision circuits.
UNDER_RANGE	The sensor connected to the Input is reading a value lower than the normal operating range. If the object is a Binary Input, this is possible when the Binary Input is actually a binary state calculated from an analog sensor.
OPEN_LOOP	The connection between the defined object and the physical device is providing a value indicating an open circuit condition.
SHORTED_LOOP	The connection between the defined object and the physical device is providing a value indicating a short circuit condition.
NO_OUTPUT	No physical device is connected to the Output object.
PROCESS_ERROR	A processing error was encountered.
MULTI_STATE_FAULT	The Present_Value of the Multi-state object is equal to one of the states in the Fault_Values property and no other fault has been detected.
CONFIGURATION_ERROR	The object's properties are not in a consistent state.
UNRELIABLE_OTHER	The controller has detected that the present value is unreliable, but none of the other conditions describe the nature of the problem. A generic fault other than those listed above has been detected, e.g., a Binary Input is not cycling as expected.

12.1 Accumulator Object Type

The Accumulator object type defines a standardized object whose properties represent the externally visible characteristics of a device that indicates measurements made by counting pulses.

This object maintains precise measurement of input count values, accumulated over time. The accumulation of pulses represents the measured quantity in unsigned integer units. This object is also concerned with the accurate representation of values presented on meter read-outs. This includes the ability to initially set the Present_Value property to the value currently displayed by the meter (as when the meter is installed), and to duplicate the means by which it is advanced, including simulating a modulo-N divider prescaling the actual meter display value, as shown in Figure 12-1.

Typical applications of such devices are in peak load management and in accounting and billing management systems. This object is not intended to meet all such applications. Its purpose is to provide information about the quantity being measured, such as electric power, water, or natural gas usage, according to criteria specific to the application.

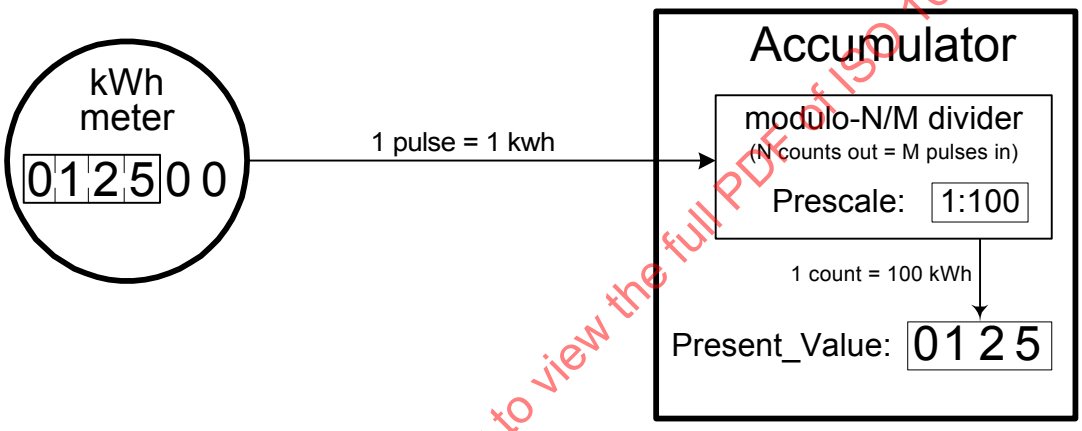


Figure 12-1. Example of an Accumulator object

The object and its properties are summarized in Table 12-1 and described in detail in this subclause.

Table 12-1. Properties of the Accumulator Object

Property Identifier	Property Datatype	Conformance Code
Object_Identifier	BACnetObjectIdentifier	R
Object_Name	CharacterString	R
Object_Type	BACnetObjectType	R
Present_Value	Unsigned	R[1]
Description	CharacterString	O
Device_Type	CharacterString	O
Status_Flags	BACnetStatusFlags	R
Event_State	BACnetEventState	R
Reliability	BACnetReliability	O
Out_Of_Service	BOOLEAN	R
Scale	BACnetScale	R
Units	BACnetEngineeringUnits	R
Prescale	BACnetPrescale	O
Max_Pres_Value	Unsigned	R
Value_Change_Time	BACnetDateTime	O[2]
Value_Before_Change	Unsigned	O[2,3]
Value_Set	Unsigned	O[2,3]
Logging_Record	BACnetAccumulatorRecord	O
Logging_Object	BACnetObjectIdentifier	O
Pulse_Rate	Unsigned	O[1,4]
High_Limit	Unsigned	O[4]
Low_Limit	Unsigned	O[4]
Limit_Monitoring_Interval	Unsigned	O[4]
Notification_Class	Unsigned	O[4]
Time_Delay	Unsigned	O[4]
Limit_Enable	BACnetLimitEnable	O[4]
Event_Enable	BACnetEventTransitionBits	O[4]
Acked_Transitions	BACnetEventTransitionBits	O[4]
Notify_Type	BACnetNotifyType	O[4]
Event_Time_Stamps	BACnetARRAY[3] of BACnetTimeStamp	O[4]
Profile_Name	CharacterString	O

[1] This property is required to be writable when Out_Of_Service is TRUE.
[2] These properties are required if either Value_Before_Change or Value_Set is writable.
[3] Either Value_Before_Change or Value_Set may be writable, but not both.
[4] These properties are required if the object supports intrinsic reporting.

12.1.1 Object_Identifier

This property, of type BACnetObjectIdentifier, is a numeric code that is used to identify the object. It shall be unique within the BACnet Device that maintains it.

12.1.2 Object_Name

This property, of type CharacterString, shall represent a name for the object that is unique within the BACnet Device that maintains it. The minimum length of the string shall be one character. The set of characters used in the Object_Name shall be restricted to printable characters.

12.1.3 Object_Type

This property, of type BACnetObjectType, indicates membership in a particular object type class. The value of this property shall be ACCUMULATOR.

12.1.4 Present_Value

This property, of type Unsigned, indicates the count of the input pulses, prescaled if the Prescale property is present, acquired since the value was most recently set by writing to the Value_Set property.

The value of this property shall remain in the range from zero through Max_Pres_Value. All operations on the Present_Value property are performed modulo (Max_Pres_Value+1).

This property shall be writable when Out_Of_Service is TRUE.

12.1.5 Description

This property, of type CharacterString, is a string of printable characters whose content is not restricted.

12.1.6 Device_Type

This property, of type CharacterString, is a text description of the physical device represented by the Accumulator object. It will typically be used to describe the type of sensor represented by the Accumulator.

12.1.7 Status_Flags

This property, of type BACnetStatusFlags, represents four Boolean flags that indicate the general "health" of an Accumulator object. Three of the flags are associated with the values of other properties of this object. A more detailed status could be determined by reading the properties that are linked to these flags. The relationship between individual flags is not defined by the protocol. The four flags are

{IN_ALARM, FAULT, OVERRIDDEN, OUT_OF_SERVICE}

where:

IN_ALARM — Logical FALSE (0) if the Event_State property has a value of NORMAL, otherwise logical TRUE (1).

FAULT — Logical TRUE (1) if the Reliability property is present and does not have a value of NO_FAULT_DETECTED, otherwise logical FALSE (0).

OVERRIDDEN — Logical TRUE (1) if the point has been overridden by some mechanism local to the BACnet Device. In this context "overridden" is taken to mean that the Present_Value and Reliability properties are no longer tracking changes to the physical input. Otherwise, the value is logical FALSE (0).

OUT_OF_SERVICE — Logical TRUE (1) if the Out_Of_Service property has a value of TRUE, otherwise logical FALSE (0).

12.1.8 Event_State

The Event_State property, of type BACnetEventState, is included in order to provide a way to determine if this object has an active event state associated with it. If the object supports intrinsic reporting, then the Event_State property shall indicate the event state of the object. If the object does not support intrinsic reporting and if the Reliability property is not present, then the value of this property shall be NORMAL. If the Reliability property is present and does not have a value of NO_FAULT_DETECTED, then the value of the Event_State property shall be FAULT. Changes in the Event_State property to the value FAULT are considered to be "fault" events.

12.1.9 Reliability

The Reliability property, of type BACnetReliability, provides an indication of whether the Present_Value property or the operation of the physical input in question is "reliable" as far as the BACnet Device or operator can determine and, if not, why. The Reliability property for this object type may have any of the following values:

{NO_FAULT_DETECTED, NO_SENSOR, OVER_RANGE, UNDER_RANGE, OPEN_LOOP, SHORTED_LOOP, UNRELIABLE_OTHER}

12.1.10 Out_Of_Service

The Out_Of_Service property, of type BOOLEAN, is an indication whether (TRUE) or not (FALSE) the physical input that the object represents is not in service. This means that the Present_Value and Pulse_Rate properties are decoupled from the physical input and will not track changes to the physical input when the value of Out_Of_Service is TRUE. In addition, the Reliability property and the corresponding state of the FAULT flag of the Status_Flags property shall be decoupled from the physical input when Out_Of_Service is TRUE. While the Out_Of_Service property is TRUE, the Present_Value, Pulse_Rate and Reliability properties may be changed to any value as a means of simulating specific fixed conditions or for testing purposes. Other functions that depend on the state of the Present_Value, Pulse_Rate or Reliability properties shall respond to changes made to these properties while Out_Of_Service is TRUE, as if those changes had occurred in the physical input.

12.1.11 Scale

This property, of type BACnetScale, indicates the conversion factor to be multiplied with the value of the Present_Value property to provide a value in the units indicated by Units. The choice of options for this property determine how the scaling operation (which is performed by the client reading this object) is performed:

Option	Datatype	Indicated Value in Units
floatScale	REAL	Present_Value x Scale
integerScale	INTEGER	Present_Value x 10^{Scale}

12.1.12 Units

This property, of type BACnetEngineeringUnits, indicates the measurement units of the Present_Value when multiplied with the scaling factor indicated by Scale. See the BACnetEngineeringUnits ASN.1 production in Clause 21 for a list of engineering units defined by this standard.

12.1.13 Prescale

This property, of type BACnetPrescale, presents the coefficients that are used for converting the pulse signals generated by the measuring instrument into the value displayed by Present_Value. The conversions are performed using integer arithmetic in such a fashion that no measurement-generated pulse signals are lost in the conversion.

These coefficients might simply document a conversion performed prior to the reception of the input pulses by the Accumulator object, or they might actually be used by the Accumulator to convert input pulses into the value displayed by Present_Value. Whichever is done is a local matter.

The coefficients are as follows:

multiplier	The numerator of the conversion factor expressed as a ratio of integers.
moduloDivide	The denominator of the conversion factor expressed as a ratio of integers.

The conversion algorithm is performed as follows, utilizing a non-displayed variable called an accumulator:

For each input pulse:
Add the value of 'multiplier' to an accumulator and then,
while the accumulator is greater than or equal to the value of 'moduloDivide':
Increment the value of Present_Value by one, and
decrease the value of the accumulator by the value of 'moduloDivide'.

This procedure supports non-integral ratios of measurement pulses to Present_Value. For example, in an electrical metering application, the output of the voltage- and current-measuring systems might be 9000/1200 (scale / voltage*current) pulses per kWh, requiring the Accumulator object to accumulate 2/15 kWh/pulse. With this algorithm such pulses can be accurately accumulated and displayed when the units of Present_Value are KILOWATT_HOURS.

12.1.14 Max_Pres_Value

This property, of type Unsigned, indicates the maximum value of the Present_Value property.

12.1.15 Value_Change_Time

This read-only property, of type BACnetDateTime, shall be present if the Present_Value property is adjustable by writing to the Value_Before_Change or Value_Set properties. It represents the date and time of the most recent occurrence of such a write operation. If no such write has yet occurred, this property shall have wildcard values for all date and time fields.

12.1.16 Value_Before_Change

This property, of type Unsigned, indicates the value of the Present_Value property just prior to the most recent write to the Value_Set or Value_Before_Change properties. If no such write has yet occurred, this property shall have the value zero. If this property is writable, the Value_Set property shall be read-only.

If this property is writable, the following series of operations, for which the associated properties are present, shall be performed atomically by the object when this property is written:

(1) The value of Present_Value shall be copied to the Value_Set property.
(2) The value written to Value_Before_Change shall be stored in the Value_Before_Change property.
(3) The current date and time shall be stored in the Value_Change_Time property.

While this series of operations is being performed, it is critical that any other process not change the Present_Value, Value_Set and Value_Before_Change properties.

12.1.17 Value_Set

This property, of type Unsigned, indicates the value of the Present_Value property after the most recent write to the Value_Set or Value_Before_Change properties. If no such write has yet occurred, this property shall have the value zero. If this property is writable, the Value_Before_Change property shall be read-only.

If this property is writable, the following series of operations, for which the associated properties are present, shall be performed atomically by the object when this property is written:

(1) The value of Present_Value shall be copied to the Value_Before_Change property.
(2) The value written to Value_Set shall be stored in both the Value_Set and Present_Value properties.
(3) The current date and time shall be stored in the Value_Change_Time property.

While this series of operations are being performed, it is critical that any other process not change the Present_Value, Value_Set and Value_Before_Change properties.

12.1.18 Logging_Record

This read-only property, of type BACnetAccumulatorRecord, is a list of values that must be acquired and returned "atomically" in order to allow proper interpretation of the data.

If the Logging_Object property is present, then, when Logging_Record is acquired by the object identified by Logging_Object, this list of values shall be saved and returned when read by other objects or devices. If the Logging_Object property is present and Logging_Record has not yet been acquired by the object identified by Logging_Object, 'timestamp' shall contain all wildcards, 'present-value' and 'accumulated-value' shall contain the value zero, and 'accumulator-status' shall indicate STARTING.

The list of values ('timestamp', 'present-value', 'accumulated-value', and 'accumulator-status') shall be acquired from the underlying system when they reflect a stable state of the device (for example, they shall not be acquired when Present_Value has just been incremented but the corresponding increment of 'accumulated-value' has not yet occurred).

The items returned in the list of values are:

timestamp — The local date and time when the data was acquired.

present-value — The value of the Present_Value property.

accumulated-value — The short term accumulated value of the counter. The algorithm used to calculate accumulated-value is a function of the value of accumulator-status. If this is the initial read, the value returned shall be zero.

accumulator-status — An indication of the reliability of the data in this list of values.

The accumulator-status parameter may take on any of the following values:

{NORMAL, STARTING, RECOVERED, ABNORMAL, FAILED}

where the values are defined as follows:

NORMAL — No event affecting the reliability of the data has occurred during the period from the preceding to the current qualified reads of the Logging_Record property. In this case 'accumulated-value' shall be represented by the expression:

$$\text{accumulated-value} = \text{Present\_Value}_{curren} - \text{Present\_Value}_{previous}$$

STARTING — This value indicates that the data in Logging_Records is either the first data to be acquired since startup by the object identified by Logging_Object (if 'timestamp' has non-wildcard values) or that no data has been acquired since startup by the object identified by Logging_Object (in which case 'timestamp' has all wildcard values).

RECOVERED — One or more writes to Value_Before_Change or Value_Set have occurred since Logging_Record was acquired by the object identified by Logging_Object. For the case of a single write, 'accumulated-value' shall be represented by the expression:

$$\text{accumulated-value} = (\text{Present\_Value}_{current} - \text{Value\_Set}) + (\text{Value\_Before\_Change} - \text{Present\_Value}_{previous})$$

ABNORMAL — The accumulation has been carried out, but some unrecoverable event such as the clock's time being changed by a significant amount since Logging_Record was acquired by the object identified by Logging_Object. (How much time is considered significant shall be a local matter.)

FAILED — The 'accumulated-value' item is not reliable due to some problem. The criteria for returning this value are a local matter.

Changes in the value of 'accumulator-status' shall occur only when the Logging_Record is acquired by the object identified by Logging_Object.

12.1.19 Logging_Object

This property, of type BACnetObjectIdentifier, indicates the object in the same device as the Accumulator object which, when it acquires Logging_Record data from the Accumulator object, shall cause the Accumulator object to acquire, present and store the data from the underlying system.

12.1.20 Pulse_Rate

This property, of type Unsigned, shall indicate the number of input pulses received during the most recent period specified by Limit_Monitoring_Interval. The mechanism that associates the input signal with the value indicated by this property is a local matter.

This property shall be writable when Out_Of_Service is TRUE.

12.1.21 High_Limit

This property, of type Unsigned, shall specify a limit that Pulse_Rate must exceed before an event is generated. This property is required if this object supports intrinsic reporting.

12.1.21.1 Conditions for Generating a TO-OFFNORMAL Event

A TO-OFFNORMAL event is generated under these conditions:

(a) Pulse_Rate must exceed High_Limit for a minimum period of time, specified in the Time_Delay property, and
(b) the HighLimitEnable flag must be set in the Limit_Enable property, and
(c) the TO-OFFNORMAL flag must be set in the Event_Enable property.

12.1.21.2 Conditions for Generating a TO-NORMAL Event

Once exceeded, Pulse_Rate must fall below High_Limit before a TO-NORMAL event is generated under these conditions:

(a) Pulse_Rate must remain below High_Limit for a minimum period of time, specified in the Time_Delay property, and
(b) the HighLimitEnable flag must be set in the Limit_Enable property, and
(c) the TO-NORMAL flag must be set in the Event_Enable property.

12.1.22 Low_Limit

This property, of type Unsigned, shall specify a limit that Pulse_Rate must fall below before an event is generated. This property is required if this object supports intrinsic reporting.

12.1.22.1 Conditions for Generating a TO-OFFNORMAL Event

A TO-OFFNORMAL event is generated under these conditions:

(a) Pulse_Rate must fall below Low_Limit for a minimum period of time, specified in the Time_Delay property, and
(b) the LowLimitEnable flag must be set in the Limit_Enable property, and
(c) the TO-OFFNORMAL flag must be set in the Event_Enable property.

12.1.22.2 Conditions for Generating a TO-NORMAL Event

Once Pulse_Rate has fallen below the Low_Limit, the Pulse_Rate must become greater than Low_Limit before a TO-NORMAL event is generated under these conditions:

(a) Pulse_Rate must become greater than Low_Limit for a minimum period of time, specified in the Time_Delay property, and
(b) the LowLimitEnable flag must be set in the Limit_Enable property, and
(c) the TO-NORMAL flag must be set in the Event_Enable property.

12.1.23 Limit_Monitoring_Interval

This property, of type Unsigned, specifies the monitoring period in seconds for determining the value of Pulse_Rate. The use of a fixed or sliding time window for detecting pulse rate is a local matter. This property is required if this object supports intrinsic reporting.

12.1.24 Notification_Class

This property, of type Unsigned, shall specify the notification class to be used when handling and generating event notifications for this object. The Notification_Class property implicitly refers to a Notification Class object that has a Notification_Class property with the same value. This property is required if this object supports intrinsic reporting.

12.1.25 Time_Delay

This property, of type Unsigned, shall specify the minimum period of time in seconds that Pulse_Rate must remain outside the range from Low_Limit through High_Limit, before a TO-OFFNORMAL event is generated, or within the same band before a TO-NORMAL event is generated. This property is required if this object supports intrinsic reporting.

12.1.26 Limit_Enable

This property, of type BACnetLimitEnable, shall convey two flags that separately enable and disable reporting of High_Limit and Low_Limit offnormal events and their return to normal. This property is required if this object supports intrinsic reporting.

12.1.27 Event_Enable

This property, of type BACnetEventTransitionBits, shall convey three flags that separately enable and disable reporting of TO-OFFNORMAL, TO-FAULT, and TO-NORMAL events. In the context of Accumulator objects, transitions to the High_Limit or Low_Limit Event_States are considered to be "offnormal" events. This property is required if this object supports intrinsic reporting.

12.1.28 Acked_Transitions

This property, of type BACnetEventTransitionBits, shall convey three flags that separately indicate the receipt of acknowledgements for TO-OFFNORMAL, TO-FAULT, and TO-NORMAL events. In the context of Accumulator objects, transitions to High_Limit and Low_Limit Event_State are considered to be "offnormal" events. These flags shall be cleared upon the occurrence of the corresponding event and set under any of these conditions:

(a) upon receipt of the corresponding acknowledgement;
(b) upon the occurrence of the event if the corresponding flag is not set in the Event_Enable property (meaning event notifications will not be generated for this condition and thus no acknowledgement is expected);
(c) upon the occurrence of the event if the corresponding flag is set in the Event_Enable property and the corresponding flag in the Ack_Required property of the Notification Class object implicitly referenced by the Notification_Class property of this object is not set (meaning no acknowledgement is expected).

This property is required if this object supports intrinsic reporting.

12.1.29 Notify_Type

This property, of type BACnetNotifyType, shall convey whether the notifications generated by the object should be Events or Alarms. This property is required if this object supports intrinsic reporting.

12.1.30 Event_Time_Stamps

This optional property, of type BACnetARRAY[3] of BACnetTimeStamp, shall convey the times of the last event notifications for TO-OFFNORMAL, TO-FAULT, and TO-NORMAL events, respectively. Time stamps of type Time or Date shall have X'FF' in each octet and Sequence number time stamps shall have the value 0 if no event notification of that type has been generated since the object was created. This property is required if this object supports intrinsic reporting.

12.1.31 Profile_Name

This property, of type CharacterString, is the name of an object profile to which this object conforms. To ensure uniqueness, a profile name must begin with a vendor identifier code (see Clause 23) in base-10 integer format, followed by a dash. All subsequent characters are administered by the organization registered with that vendor identifier code. The vendor identifier code that prefixes the profile name shall indicate the organization that publishes and maintains the profile document named by the remainder of the profile name. This vendor identifier need not have any relationship to the vendor identifier of the device within which the object resides.

A profile defines a set of additional properties, behavior, and/or requirements for this object beyond those specified here. This standard defines only the format of the names of profiles. The definition of the profiles themselves is outside the scope of this standard.

12.2 Analog Input Object Type

The Analog Input object type defines a standardized object whose properties represent the externally visible characteristics of an analog input. The object and its properties are summarized in Table 12-2 and described in detail in this subclause.

Table 12-2. Properties of the Analog Input Object Type

Property Identifier	Property Datatype	Conformance Code
Object_Identifier	BACnetObjectIdentifier	R
Object_Name	CharacterString	R
Object_Type	BACnetObjectType	R
Present_Value	REAL	R[1]
Description	CharacterString	O
Device_Type	CharacterString	O
Status_Flags	BACnetStatusFlags	R
Event_State	BACnetEventState	R
Reliability	BACnetReliability	O
Out_Of_Service	BOOLEAN	R
Update_Interval	Unsigned	O
Units	BACnetEngineeringUnits	R
Min_Pres_Value	REAL	O
Max_Pres_Value	REAL	O
Resolution	REAL	O
COV_Increment	REAL	O[2]
Time_Delay	Unsigned	O[3]
Notification_Class	Unsigned	O[3]
High_Limit	REAL	O[3]
Low_Limit	REAL	O[3]
Deadband	REAL	O[3]
Limit_Enable	BACnetLimitEnable	O[3]
Event_Enable	BACnetEventTransitionBits	O[3]
Acked_Transitions	BACnetEventTransitionBits	O[3]
Notify_Type	BACnetNotifyType	O[3]
Event_Time_Stamps	BACnetARRAY[3] of BACnetTimeStamp	O[3]
Profile_Name	CharacterString	O

[1] This property is required to be writable when Out_Of_Service is TRUE.
[2] This property is required if the object supports COV reporting.
[3] These properties are required if the object supports intrinsic reporting.

12.2.1 Object_Identifier

This property, of type BACnetObjectIdentifier, is a numeric code that is used to identify the object. It shall be unique within the BACnet Device that maintains it**.**

12.2.2 Object_Name

This property, of type CharacterString, shall represent a name for the object that is unique within the BACnet Device that maintains it. The minimum length of the string shall be one character. The set of characters used in the Object_Name shall be restricted to printable characters**.**

12.2.3 Object_Type

This property, of type BACnetObjectType, indicates membership in a particular object type class. The value of this property shall be ANALOG_INPUT.

12.2.4 Present_Value

This property, of type REAL, indicates the current value, in engineering units, of the input being measured. The Present_Value property shall be writable when Out_Of_Service is TRUE.

12.2.5 Description

This property, of type CharacterString, is a string of printable characters whose content is not restricted**.**

12.2.6 Device_Type

This property, of type CharacterString, is a text description of the physical device connected to the analog input. It will typically be used to describe the type of sensor attached to the analog input.

12.2.7 Status_Flags

This property, of type BACnetStatusFlags, represents four Boolean flags that indicate the general "health" of an analog input. Three of the flags are associated with the values of other properties of this object. A more detailed status could be determined by reading the properties that are linked to these flags. The relationship between individual flags is not defined by the protocol. The four flags are

{IN_ALARM, FAULT, OVERRIDDEN, OUT_OF_SERVICE}

where:

IN_ALARM — Logical FALSE (0) if the Event_State property has a value of NORMAL, otherwise logical TRUE (1).

FAULT — Logical TRUE (1) if the Reliability property is present and does not have a value of NO_FAULT_DETECTED, otherwise logical FALSE (0).

OVERRIDDEN — Logical TRUE (1) if the point has been overridden by some mechanism local to the BACnet Device. In this context "overridden" is taken to mean that the Present_Value and Reliability properties are no longer tracking changes to the physical input. Otherwise, the value is logical FALSE (0).

OUT_OF_SERVICE — Logical TRUE (1) if the Out_Of_Service property has a value of TRUE, otherwise logical FALSE (0).

12.2.8 Event_State

The Event_State property, of type BACnetEventState, is included in order to provide a way to determine if this object has an active event state associated with it. If the object supports intrinsic reporting, then the Event_State property shall indicate the event state of the object. If the object does not support intrinsic reporting, then the value of this property shall be NORMAL. If the Reliability property is present and does not have a value of NO_FAULT_DETECTED, then the value of the Event_State property shall be FAULT. Changes in the Event_State property to the value FAULT are considered to be "fault" events.

12.2.9 Reliability

The Reliability property, of type BACnetReliability, provides an indication of whether the Present_Value or the operation of the physical input in question is "reliable" as far as the BACnet Device or operator can determine and, if not, why. The Reliability property for this object type may have any of the following values:

{NO_FAULT_DETECTED, NO_SENSOR, OVER_RANGE, UNDER_RANGE, OPEN_LOOP, SHORTED_LOOP, UNRELIABLE_OTHER}

12.2.10 Out_Of_Service

The Out_Of_Service property, of type BOOLEAN, is an indication whether (TRUE) or not (FALSE) the physical input that the object represents is not in service. This means that the Present_Value property is decoupled from the physical input and will not track changes to the physical input when the value of Out_Of_Service is TRUE. In addition, the Reliability property and the corresponding state of the FAULT flag of the Status_Flags property shall be decoupled from the physical input when Out_Of_Service is TRUE. While the Out_Of_Service property is TRUE, the Present_Value and Reliability properties may be changed to any value as a means of simulating specific fixed conditions or for testing purposes. Other functions that depend on the state of the Present_Value or Reliability properties shall respond to changes made to these properties while Out_Of_Service is TRUE, as if those changes had occurred in the physical input.

12.2.11 Update_Interval

This property, of type Unsigned, indicates the maximum period of time between updates to the Present_Value in hundredths of a second when the input is not overridden and not out-of-service**.**

12.2.12 Units

This property, of type BACnetEngineeringUnits, indicates the measurement units of this object. See the BACnetEngineeringUnits ASN.1 production in Clause 21 for a list of engineering units defined by this standard.

12.2.13 Min_Pres_Value

This property, of type REAL, indicates the lowest number in engineering units that can be reliably obtained for the Present_Value property of this object.

12.2.14 Max_Pres_Value

This property, of type REAL, indicates the highest number in engineering units that can be reliably obtained for the Present_Value property of this object.

12.2.15 Resolution

This property, of type REAL, indicates the smallest recognizable change in Present_Value in engineering units (read-only).

12.2.16 COV_Increment

This property, of type REAL, shall specify the minimum change in Present_Value that will cause a COVNotification to be issued to subscriber COV-clients. This property is required if COV reporting is supported by this object.

12.2.17 Time_Delay

This property, of type Unsigned, shall specify the minimum period of time in seconds that the Present_Value must remain outside the band defined by the High_Limit and Low_Limit properties before a TO-OFFNORMAL event is generated or within the same band, including the Deadband property, before a TO-NORMAL event is generated. This property is required if intrinsic reporting is supported by this object.

12.2.18 Notification_Class

This property, of type Unsigned, shall specify the notification class to be used when handling and generating event notifications for this object. The Notification_Class property implicitly refers to a Notification Class object that has a Notification_Class property with the same value. This property is required if intrinsic reporting is supported by this object.

12.2.19 High_Limit

This property, of type REAL, shall specify a limit that the Present_Value must exceed before an event is generated. This property is required if intrinsic reporting is supported by this object.

12.2.19.1 Conditions for Generating a TO-OFFNORMAL Event

A TO-OFFNORMAL event is generated under these conditions:

(a) the Present_Value must exceed the High_Limit for a minimum period of time, specified in the Time_Delay property, and
(b) the HighLimitEnable flag must be set in the Limit_Enable property, and
(c) the TO-OFFNORMAL flag must be set in the Event_Enable property.

12.2.19.2 Conditions for Generating a TO-NORMAL Event

Once exceeded, the Present_Value must fall below the High_Limit minus the Deadband before a TO-NORMAL event is generated under these conditions:

(a) the Present_Value must fall below the High_Limit minus the Deadband for a minimum period of time, specified in the Time_Delay property, and
(b) the HighLimitEnable flag must be set in the Limit_Enable property, and
(c) the TO-NORMAL flag must be set in the Event_Enable property.

12.2.20 Low_Limit

This property, of type REAL, shall specify a limit that the Present_Value must fall below before an event is generated. This property is required if intrinsic reporting is supported by this object.

12.2.20.1 Conditions for Generating a TO-OFFNORMAL Event

A TO-OFFNORMAL event is generated under these conditions:

(a) the Present_Value must fall below the Low_Limit for a minimum period of time, specified in the Time_Delay property, and
(b) the LowLimitEnable flag must be set in the Limit_Enable property, and
(c) the TO-OFFNORMAL flag must be set in the Event_Enable property.

12.2.20.2 Conditions for Generating a TO-NORMAL Event

Once the Present_Value has fallen below the Low_Limit, the Present_Value must exceed the Low_Limit plus the Deadband before a TO-NORMAL event is generated under these conditions:

(a) the Present_Value must exceed the Low_Limit plus the Deadband for a minimum period of time, specified in the Time_Delay property, and
(b) the LowLimitEnable flag must be set in the Limit_Enable property, and
(c) the TO-NORMAL flag must be set in the Event_Enable property.

12.2.21 Deadband

This property, of type REAL, shall specify a range between the High_Limit and Low_Limit properties, which the Present_Value must remain within for a TO-NORMAL event to be generated under these conditions:

(a) the Present_Value must fall below the High_Limit minus Deadband, and
(b) the Present_Value must exceed the Low_Limit plus the Deadband, and
(c) the Present_Value must remain within this range for a minimum period of time, specified in the Time_Delay property, and
(d) either the HighLimitEnable or LowLimitEnable flag must be set in the Limit_Enable property, and
(e) the TO-NORMAL flag must be set in the Event_Enable property.

This property is required if intrinsic reporting is supported by this object.

12.2.22 Limit_Enable

This property, of type BACnetLimitEnable, shall convey two flags that separately enable and disable reporting of high limit and low limit offnormal events and their return to normal. This property is required if intrinsic reporting is supported by this object.

12.2.23 Event_Enable

This property, of type BACnetEventTransitionBits, shall convey three flags that separately enable and disable reporting of TO-OFFNORMAL, TO-FAULT, and TO-NORMAL events. In the context of Analog Input objects, transitions to High_Limit and Low_Limit Event_States are considered to be "offnormal" events. This property is required if intrinsic reporting is supported by this object.

12.2.24 Acked_Transitions

This property, of type BACnetEventTransitionBits, shall convey three flags that separately indicate the receipt of acknowledgments for TO-OFFNORMAL, TO-FAULT, and TO-NORMAL events. In the context of Analog Input objects, transitions to High_Limit and Low_Limit Event_States are considered to be "offnormal" events. These flags shall be cleared upon the occurrence of the corresponding event and set under any of these conditions:

(a) upon receipt of the corresponding acknowledgment;
(b) upon the occurrence of the event if the corresponding flag is not set in the Event_Enable property (meaning event notifications will not be generated for this condition and thus no acknowledgment is expected);
(c) upon the occurrence of the event if the corresponding flag is set in the Event_Enable property and the corresponding flag in the Ack_Required property of the Notification Class object implicitly referenced by the Notification_Class property of this object is not set (meaning no acknowledgment is expected).

This property is required if intrinsic reporting is supported by this object.

12.2.25 Notify_Type

This property, of type BACnetNotifyType, shall convey whether the notifications generated by the object should be Events or Alarms. This property is required if intrinsic reporting is supported by this object.

12.2.26 Event_Time_Stamps

This optional property, of type BACnetARRAY[3] of BACnetTimeStamp, shall convey the times of the last event notifications for TO-OFFNORMAL, TO-FAULT, and TO-NORMAL events, respectively. Time stamps of type Time or Date shall have 'FF' in each octet and Sequence number time stamps shall have the value 0 if no event notification of that type has been generated since the object was created. This property is required if intrinsic reporting is supported by this object.

12.2.27 Profile_Name

This optional property, of type CharacterString, is the name of an object profile to which this object conforms. To ensure uniqueness, a profile name must begin with a vendor identifier code (see Clause 23) in base-10 integer format, followed by a dash. All subsequent characters are administered by the organization registered with that vendor identifier code. The vendor identifier code that prefixes the profile name shall indicate the organization that publishes and maintains the profile document named by the remainder of the profile name. This vendor identifier need not have any relationship to the vendor identifier of the device within which the object resides.

A profile defines a set of additional properties, behavior, and/or requirements for this object beyond those specified here. This standard defines only the format of the names of profiles. The definition of the profiles themselves is outside the scope of this standard.

12.3 Analog Output Object Type

The Analog Output object type defines a standardized object whose properties represent the externally visible characteristics of an analog output. The object and its properties are summarized in Table 12-3 and described in detail in this subclause.

Table 12-3. Properties of the Analog Output Object Type

Property Identifier	Property Datatype	Conformance Code
Object_Identifier	BACnetObjectIdentifier	R
Object_Name	CharacterString	R
Object_Type	BACnetObjectType	R
Present_Value	REAL	W
Description	CharacterString	O
Device_Type	CharacterString	O
Status_Flags	BACnetStatusFlags	R
Event_State	BACnetEventState	R
Reliability	BACnetReliability	O
Out_Of_Service	BOOLEAN	R
Units	BACnetEngineeringUnits	R
Min_Pres_Value	REAL	O
Max_Pres_Value	REAL	O
Resolution	REAL	O
Priority_Array	BACnetPriorityArray	R
Relinquish_Default	REAL	R
COV_Increment	REAL	O[1]
Time_Delay	Unsigned	O[2]
Notification_Class	Unsigned	O[2]
High_Limit	REAL	O[2]
Low_Limit	REAL	O[2]
Deadband	REAL	O[2]
Limit_Enable	BACnetLimitEnable	O[2]
Event_Enable	BACnetEventTransitionBits	O[2]
Acked_Transitions	BACnetEventTransitionBits	O[2]
Notify_Type	BACnetNotifyType	O[2]
Event_Time_Stamps	BACnetARRAY[3] of BACnetTimeStamp	O[2]
Profile_Name	CharacterString	O

[1] This property is required if the object supports COV reporting.
[2] These properties are required if the object supports intrinsic reporting.

12.3.1 Object_Identifier

This property, of type BACnetObjectIdentifier, is a numeric code that is used to identify the object. It shall be unique within the BACnet Device that maintains it.

12.3.2 Object_Name

This property, of type CharacterString, shall represent a name for the object that is unique within the BACnet Device that maintains it. The minimum length of the string shall be one character. The set of characters used in the Object_Name shall be restricted to printable characters.

12.3.3 Object_Type

This property, of type BACnetObjectType, indicates membership in a particular object type class. The value of this property shall be ANALOG_OUTPUT.

12.3.4 Present_Value (Commandable)

This property, of type REAL, indicates the current value, in engineering units, of the output.

12.3.5 Description

This property, of type CharacterString, is a string of printable characters whose content is not restricted.

12.3.6 Device_Type

This property, of type CharacterString, is a text description of the physical device connected to the analog output. It will typically be used to describe the type of device attached to the analog output**.**

12.3.7 Status_Flags

This property, of type BACnetStatusFlags, represents four Boolean flags that indicate the general "health" of an analog output. Three of the flags are associated with the values of other properties of this object. A more detailed status could be determined by reading the properties that are linked to these flags. The relationship between individual flags is not defined by the protocol. The four flags are

{IN_ALARM, FAULT, OVERRIDDEN, OUT_OF_SERVICE}

where:

IN_ALARM — Logical FALSE (0) if the Event_State property has a value of NORMAL, otherwise logical TRUE (1).

FAULT — Logical TRUE (1) if the Reliability property is present and does not have a value of NO_FAULT_DETECTED, otherwise logical FALSE (0).

OVERRIDDEN — Logical TRUE (1) if the point has been overridden by some mechanism local to the BACnet Device. In this context "overridden" is taken to mean that the physical output is no longer tracking changes to the Present_Value property and the Reliability property is no longer a reflection of the physical output. Otherwise, the value is logical FALSE (0).

OUT_OF_SERVICE — Logical TRUE (1) if the Out_Of_Service property has a value of TRUE, otherwise logical FALSE (0**).**

12.3.8 Event_State

The Event_State property, of type BACnetEventState, is included in order to provide a way to determine if this object has an active event state associated with it. If the object supports intrinsic reporting, then the Event_State property shall indicate the event state of the object. If the object does not support intrinsic reporting, then the value of this property shall be NORMAL. If the Reliability property is present and does not have a value of NO_FAULT_DETECTED, then the value of the Event_State property shall be FAULT. Changes in the Event_State property to the value FAULT are considered to be "fault" events.

12.3.9 Reliability

The Reliability property, of type BACnetReliability, provides an indication of whether the Present_Value or the operation of the physical output in question is "reliable" as far as the BACnet Device or operator can determine and, if not, why. The Reliability property for this object type may have any of the following values:

{NO_FAULT_DETECTED,OPEN_LOOP,SHORTED_LOOP,NO_OUTPUT,UNRELIABLE_OTHER}

12.3.10 Out_Of_Service

The Out_Of_Service property, of type BOOLEAN, is an indication whether (TRUE) or not (FALSE) the physical point that the object represents is not in service. This means that changes to the Present_Value property are decoupled from the physical output when the value of Out_Of_Service is TRUE. In addition, the Reliability property and the corresponding state of the FAULT flag of the Status_Flags property shall be decoupled from the physical output when Out_Of_Service is TRUE. While the Out_Of_Service property is TRUE, the Present_Value and Reliability properties may still be changed to any value as a means of simulating specific fixed conditions or for testing purposes. Other functions that depend on the state of the Present_Value or Reliability properties shall respond to changes made to these properties while Out_Of_Service is TRUE, as if those changes had occurred to the physical output. The Present_Value property shall still be controlled by the BACnet command prioritization mechanism if Out_Of_Service is TRUE. See Clause 19.

12.3.11 Units

This property, of type BACnetEngineeringUnits, indicates the measurement units of this object. See the BACnetEngineeringUnits ASN.1 production in Clause 21 for a list of engineering units defined by this standard.

12.3.12 Min_Pres_Value

This property, of type REAL, indicates the lowest number that can be reliably used for the Present_Value property of this object.

12.3.13 Max_Pres_Value

This property, of type REAL, indicates the highest number that can be reliably used for the Present_Value property of this object.

12.3.14 Resolution

This property, of type REAL, indicates the smallest recognizable change in Present_Value in engineering units (read-only).

12.3.15 Priority_Array

This property is a read-only array of prioritized values. See Clause 19 for a description of the prioritization mechanism.

12.3.16 Relinquish_Default

This property is the default value to be used for the Present_Value property when all command priority values in the Priority_Array property have a NULL value. See Clause 19.

12.3.17 COV_Increment

This property, of type REAL, shall specify the minimum change in Present_Value that will cause a COVNotification to be issued to subscriber COV-clients. This property is required if COV reporting is supported by this object.

12.3.18 Time_Delay

This property, of type Unsigned, shall specify the minimum period of time in seconds the Present_Value must remain outside the band defined by the High_Limit and Low_Limit properties before a TO-OFFNORMAL event is generated or within the same band, including the Deadband property, before a TO-NORMAL event is generated. This property is required if intrinsic reporting is supported by this object.

12.3.19 Notification_Class

This property, of type Unsigned, shall specify the notification class to be used when handling and generating event notifications for this object. The Notification_Class property implicitly refers to a Notification Class object that has a Notification_Class property with the same value. This property is required if intrinsic reporting is supported by this object.

12.3.20 High_Limit

This property, of type REAL, shall specify a limit that the Present_Value must exceed before an event is generated. This property is required if intrinsic reporting is supported by this object.

12.3.20.1 Conditions for Generating a TO-OFFNORMAL Event

A TO-OFFNORMAL event is generated under these conditions:

(a) the Present_Value must exceed the High_Limit for a minimum period of time, specified in the Time_Delay property, and
(b) the HighLimitEnable flag must be set in the Limit_Enable property, and
(c) the TO-OFFNORMAL flag must be set in the Event_Enable property.

12.3.20.2 Conditions for Generating a TO-NORMAL Event

Once exceeded, the Present_Value must fall below the High_Limit minus the Deadband before a TO-NORMAL event is generated under these conditions:

(a) the Present_Value must fall below the High_Limit minus the Deadband for a minimum period of time, specified in the Time_Delay property, and

(b) the HighLimitEnable flag must be set in the Limit_Enable property, and
(c) the TO-NORMAL flag must be set in the Event_Enable property.

12.3.21 Low_Limit

This property, of type REAL, shall specify a limit below which the Present_Value must fall before an event is generated. . This property is required if intrinsic reporting is supported by this object .

12.3.21.1 Conditions for Generating a TO-OFFNORMAL Event

A TO-OFFNORMAL event is generated under these conditions:

(a) the Present_Value must fall below the Low_Limit for a minimum period of time, specified in the Time_Delay property, and
(b) the LowLimitEnable flag must be set in the Limit_Enable property, and
(c) the TO-OFFNORMAL flag must be set in the Event_Enable property.

12.3.21.2 Conditions for Generating a TO-NORMAL Event

Once the Present_Value has fallen below the Low_Limit, the Present_Value must exceed the Low_Limit plus the Deadband before a TO-NORMAL event is generated under these conditions:

(a) the Present_Value must exceed the Low_Limit plus the Deadband for a minimum period of time, specified in the Time_Delay property, and
(b) the LowLimitEnable flag must be set in the Limit_Enable property, and
(c) the TO-NORMAL flag must be set in the Event_Enable property.

12.3.22 Deadband

This property, of type REAL, shall specify a range between the High_Limit and Low_Limit properties within which the Present_Value must remain for a TO-NORMAL event to be generated under these conditions:

(a) the Present_Value must fall below the High_Limit minus Deadband, and
(b) the Present_Value must exceed the Low_Limit plus the Deadband, and
(c) the Present_Value must remain within this range for a minimum period of time, specified in the Time_Delay property, and
(d) either the HighLimitEnable or LowLimitEnable flag must be set in the Limit_Enable property, and
(e) the TO-NORMAL flag must be set in the Event_Enable property.

This property is required if intrinsic reporting is supported by this object.

12.3.23 Limit_Enable

This property, of type BACnetLimitEnable, shall convey two flags that separately enable and disable reporting of high limit and low limit offnormal events and their return to normal. This property is required if intrinsic reporting is supported by this object.

12.3.24 Event_Enable

This property, of type BACnetEventTransitionBits, shall convey three flags that separately enable and disable reporting of TO-OFFNORMAL, TO-FAULT, and TO-NORMAL events. In the context of Analog Output objects, transitions to High_Limit and Low_Limit Event_States are considered to be "offnormal" events. This property is required if intrinsic reporting is supported by this object.

12.3.25 Acked_Transitions

This property, of type BACnetEventTransitionBits, shall convey three flags that separately indicate the receipt of acknowledgments for TO-OFFNORMAL, TO-FAULT, and TO-NORMAL events. In the context of Analog Output objects, transitions to High_Limit and Low_Limit Event_States are considered to be "offnormal" events. These flags shall be cleared upon the occurrence of the corresponding event and set under any of these conditions:

(a) upon receipt of the corresponding acknowledgment;
(b) upon the occurrence of the event if the corresponding flag is not set in the Event_Enable property (meaning event notifications will not be generated for this condition and thus no acknowledgment is expected);
(c) upon the occurrence of the event if the corresponding flag is set in the Event_Enable property and the corresponding flag in the Ack_Required property of the Notification Class object implicitly referenced by the Notification_Class property of this object is not set (meaning no acknowledgment is expected).

This property is required if intrinsic reporting is supported by this object.

12.3.26 Notify_Type

This property, of type BACnetNotifyType, shall convey whether the notifications generated by the object should be Events or Alarms. This property is required if intrinsic reporting is supported by this object.

12.3.27 Event_Time_Stamps

This optional property, of type BACnetARRAY[3] of BACnetTimeStamp, shall convey the times of the last event notifications for TO-OFFNORMAL, TO-FAULT, and TO-NORMAL events, respectively. Time stamps of type Time or Date shall have 'FF' in each octet and Sequence number time stamps shall have the value 0 if no event notification of that type has been generated since the object was created. This property is required if intrinsic reporting is supported by this object.

12.3.28 Profile_Name

This optional property, of type CharacterString, is the name of an object profile to which this object conforms. To ensure uniqueness, a profile name must begin with a vendor identifier code (see Clause 23) in base-10 integer format, followed by a dash. All subsequent characters are administered by the organization registered with that vendor identifier code. The vendor identifier code that prefixes the profile name shall indicate the organization that publishes and maintains the profile document named by the remainder of the profile name. This vendor identifier need not have any relationship to the vendor identifier of the device within which the object resides.

A profile defines a set of additional properties, behavior, and/or requirements for this object beyond those specified here. This standard defines only the format of the names of profiles. The definition of the profiles themselves is outside the scope of this standard.

12.4 Analog Value Object Type

The Analog Value object type defines a standardized object whose properties represent the externally visible characteristics of an analog value. An "analog value" is a control system parameter residing in the memory of the BACnet Device. The object and its properties are summarized in Table 12-4 and described in detail in this subclause.

Table 12-4. Properties of the Analog Value Object Type

Property Identifier	Property Datatype	Conformance Code
Object_Identifier	BACnetObjectIdentifier	R
Object_Name	CharacterString	R
Object_Type	BACnetObjectType	R
Present_Value	REAL	R[4]
Description	CharacterString	O
Status_Flags	BACnetStatusFlags	R
Event_State	BACnetEventState	R
Reliability	BACnetReliability	O
Out_Of_Service	BOOLEAN	R
Units	BACnetEngineeringUnits	R
Priority_Array	BACnetPriorityArray	O[1]
Relinquish_Default	REAL	O[1]
COV_Increment	REAL	O[2]
Time_Delay	Unsigned	O[3]
Notification_Class	Unsigned	O[3]
High_Limit	REAL	O[3]
Low_Limit	REAL	O[3]
Deadband	REAL	O[3]
Limit_Enable	BACnetLimitEnable	O[3]
Event_Enable	BACnetEventTransitionBits	O[3]
Acked_Transitions	BACnetEventTransitionBits	O[3]
Notify_Type	BACnetNotifyType	O[3]
Event_Time_Stamps	BACnetARRAY[3] of BACnetTimeStamp	O[3]
Profile_Name	CharacterString	O

[1] If Present_Value is commandable, then both of these properties shall be present.
[2] This property is required if the object supports COV reporting.
[3] These properties are required if the object supports intrinsic reporting.
[4] If Present_Value is commandable, then it is required to be writable. This property is required to be writable when Out_Of_Service is TRUE.

12.4.1 Object_Identifier

This property, of type BACnetObjectIdentifier, is a numeric code that is used to identify the object. It shall be unique within the BACnet Device that maintains it.

12.4.2 Object_Name

This property, of type CharacterString, shall represent a name for the object that is unique within the BACnet Device that maintains it. The minimum length of the string shall be one character. The set of characters used in the Object_Name shall be restricted to printable characters.

12.4.3 Object_Type

This property, of type BACnetObjectType, indicates membership in a particular object type class. The value of this property shall be ANALOG_VALUE.

12.4.4 Present_Value

This property, of type REAL, indicates the current value, in engineering units, of the analog value. Present_Value shall be optionally commandable. If Present_Value is commandable for a given object instance, then the Priority_Array and

Relinquish_Default properties shall also be present for that instance. The Present_Value property shall be writable when Out_Of_Service is TRUE.

12.4.5 Description

This property, of type CharacterString, is a string of printable characters whose content is not restricted.

12.4.6 Status_Flags

This property, of type BACnetStatusFlags, represents four Boolean flags that indicate the general "health" of an analog value. Three of the flags are associated with the values of other properties of this object. A more detailed status could be determined by reading the properties that are linked to these flags. The relationship between individual flags is not defined by the protocol. The four flags are

{IN_ALARM, FAULT, OVERRIDDEN, OUT_OF_SERVICE}

where:

IN_ALARM — Logical FALSE (0) if the Event_State property has a value of NORMAL, otherwise logical TRUE (1).

FAULT — Logical TRUE (1) if the Reliability property is present and does not have a value of NO_FAULT_DETECTED, otherwise logical FALSE (0).

OVERRIDDEN — Logical TRUE (1) if the point has been overridden by some mechanism local to the BACnet Device. In this context "overridden" is taken to mean that the Present_Value property is not changeable through BACnet services. Otherwise, the value is logical FALSE (0).

OUT_OF_SERVICE — Logical TRUE (1) if the Out_Of_Service property has a value of TRUE, otherwise logical FALSE (0).

12.4.7 Event_State

The Event_State property, of type BACnetEventState, is included in order to provide a way to determine if this object has an active event state associated with it. If the object supports intrinsic reporting, then the Event_State property shall indicate the event state of the object. If the object does not support intrinsic reporting, then the value of this property shall be NORMAL. If the Reliability property is present and does not have a value of NO_FAULT_DETECTED, then the value of the Event_State property shall be FAULT. Changes in the Event_State property to the value FAULT are considered to be "fault" events.

12.4.8 Reliability

The Reliability property, of type BACnetReliability, provides an indication of whether the Present_Value is "reliable" as far as the BACnet Device can determine and, if not, why. The Reliability property for this object type may have any of the following values:

{NO_FAULT_DETECTED, OVER_RANGE, UNDER_RANGE, UNRELIABLE_OTHER}.

12.4.9 Out_Of_Service

The Out-Of-Service property, of type BOOLEAN, is an indication whether (TRUE) or not (FALSE) the Present_Value of the Analog Value object is prevented from being modified by software local to the BACnet device in which the object resides. When Out_Of_Service is TRUE, the Present_Value property may be written to freely. If the Priority_Array and Relinquish_Default properties are present, then writing to the Present_Value property shall be controlled by the BACnet command prioritization mechanism. See Clause 19.

12.4.10 Units

This property, of type BACnetEngineeringUnits, indicates the measurement units of this object. See the BACnetEngineeringUnits ASN.1 production in Clause 21 for a list of engineering units defined by this standard.

12.4.11 Priority_Array

This property is a read-only array that contains prioritized commands that are in effect for this object. See Clause 19 for a description of the prioritization mechanism. If either the Priority_Array property or the Relinquish_Default property are present, then both of them shall be present. If Present_Value is commandable, then Priority_Array and Relinquish_Default shall both be present.

12.4.12 Relinquish_Default

This property is the default value to be used for the Present_Value property when all command priority values in the Priority_Array property have a NULL value. See Clause 19. If either the Relinquish_Default property or the Priority_Array property are present, then both of them shall be present. If Present_Value is commandable, then Priority_Array and Relinquish_Default shall both be present.

12.4.13 COV_Increment

This property, of type REAL, shall specify the minimum change in Present_Value that will cause a COVNotification to be issued to subscriber COV-clients. This property is required if COV reporting is supported by this object.

12.4.14 Time_Delay

This property, of type Unsigned, shall specify the minimum period of time during which the Present_Value must remain outside the band defined by the High_Limit and Low_Limit properties before a TO-OFFNORMAL event is generated or within the same band, including the Deadband property, before a TO-NORMAL event is generated. This property is required if intrinsic reporting is supported by this object.

12.4.15 Notification_Class

This property, of type Unsigned, shall specify the notification class to be used when handling and generating event notifications for this object. The Notification_Class property implicitly refers to a Notification Class object that has a Notification_Class property with the same value. This property is required if intrinsic reporting is supported by this object.

12.4.16 High_Limit

This property, of type REAL, shall specify a limit that the Present_Value must exceed before an event is generated. This property is required if intrinsic reporting is supported by this object.

12.4.16.1 Conditions for Generating a TO-OFFNORMAL Event

A TO-OFFNORMAL event is generated under these conditions:

(a) the Present_Value must exceed the High_Limit for a minimum period of time, specified in the Time_Delay property, and
(b) the HighLimitEnable flag must be set in the Limit_Enable property, and
(c) the TO-OFFNORMAL flag must be set in the Event_Enable property.

12.4.16.2 Conditions for Generating a TO-NORMAL Event

Once exceeded, the Present_Value must fall below the High_Limit minus the Deadband before a TO-NORMAL event is generated under these conditions:

(a) the Present_Value must fall below the High_Limit minus the Deadband for a minimum period of time, specified in the Time_Delay property, and
(b) the HighLimitEnable flag must be set in the Limit_Enable property, and
(c) the TO-NORMAL flag must be set in the Event_Enable property.

12.4.17 Low_Limit

This property, of type REAL, shall specify a limit below which the Present_Value must fall before an event is generated. This property is required if intrinsic reporting is supported by this object.

12.4.17.1 Conditions for Generating a TO-OFFNORMAL Event

A TO-OFFNORMAL event is generated under these conditions:

(a) the Present_Value must fall below the Low_Limit for a minimum period of time, specified in the Time_Delay property, and
(b) the LowLimitEnable flag must be set in the Limit_Enable property, and
(c) the TO-OFFNORMAL flag must be set in the Event_Enable property.

12.4.17.2 Conditions for Generating a TO-NORMAL Event

Once the Present_Value has fallen below the Low_Limit, the Present_Value must exceed the Low_Limit plus the Deadband before a TO-NORMAL event is generated under these conditions:

(a) the Present_Value must exceed the Low_Limit plus the Deadband for a minimum period of time, specified in the Time_Delay property, and
(b) the LowLimitEnable flag must be set in the Limit_Enable property, and
(c) the TO-NORMAL flag must be set in the Event_Enable property.

12.4.18 Deadband

This property, of type REAL, shall specify a range between the High_Limit and Low_Limit properties within which the Present_Value must remain for a TO-NORMAL event to be generated under these conditions:

(a) the Present_Value must fall below the High_Limit minus Deadband, and
(b) the Present_Value must exceed the Low_Limit plus the Deadband, and
(c) the Present_Value must remain within this range for a minimum period of time, specified in the Time_Delay property, and
(d) either the HighLimitEnable or LowLimitEnable flag must be set in the Limit_Enable property, and
(e) the TO-NORMAL flag must be set in the Event_Enable property.

This property is required if intrinsic reporting is supported by this object.

12.4.19 Limit_Enable

This property, of type BACnetLimitEnable, shall convey two flags that separately enable and disable reporting of high limit and low limit offnormal events and their return to normal. This property is required if intrinsic reporting is supported by this object.

12.4.20 Event_Enable

This property, of type BACnetEventTransitionBits, shall convey three flags that separately enable and disable reporting of TO-OFFNORMAL, TO-FAULT, and TO-NORMAL events. In the context of Analog Value objects, transitions to High_Limit and Low_Limit Event_States are considered to be "offnormal" events. This property is required if intrinsic reporting is supported by this object.

12.4.21 Acked_Transitions

This property, of type BACnetEventTransitionBits, shall convey three flags that separately indicate the receipt of acknowledgments for TO-OFFNORMAL, TO-FAULT, and TO-NORMAL events. In the context of Analog Value objects, transitions to High_Limit and Low_Limit Event_States are considered to be "offnormal" events. These flags shall be cleared upon the occurrence of the corresponding event and set under any of these conditions:

(a) upon receipt of the corresponding acknowledgment;
(b) upon the occurrence of the event if the corresponding flag is not set in the Event_Enable property (meaning event notifications will not be generated for this condition and thus no acknowledgment is expected);
(c) upon the occurrence of the event if the corresponding flag is set in the Event_Enable property and the corresponding flag in the Ack_Required property of the Notification Class object implicitly referenced by the Notification_Class property of this object is not set (meaning no acknowledgment is expected).

This property is required if intrinsic reporting is supported by this object.

12.4.22 Notify_Type

This property, of type BACnetNotifyType, shall convey whether the notifications generated by the object should be Events or Alarms. This property is required if intrinsic reporting is supported by this object.

12.4.23 Event_Time_Stamps

This optional property, of type BACnetARRAY[3] of BACnetTimeStamp, shall convey the times of the last event notifications for TO-OFFNORMAL, TO-FAULT, and TO-NORMAL events, respectively. Time stamps of type Time or Date shall have 'FF' in each octet and Sequence number time stamps shall have the value 0 if no event notification of that type has been generated since the object was created. This property is required if intrinsic reporting is supported by this object.

12.4.24 Profile_Name

This optional property, of type CharacterString, is the name of an object profile to which this object conforms. To ensure uniqueness, a profile name must begin with a vendor identifier code (see Clause 23) in base-10 integer format, followed by a dash. All subsequent characters are administered by the organization registered with that vendor identifier code. The vendor identifier code that prefixes the profile name shall indicate the organization that publishes and maintains the profile document named by the remainder of the profile name. This vendor identifier need not have any relationship to the vendor identifier of the device within which the object resides.

A profile defines a set of additional properties, behavior, and/or requirements for this object beyond those specified here. This standard defines only the format of the names of profiles. The definition of the profiles themselves is outside the scope of this standard.

12.5 Averaging Object Type

The Averaging object type defines a standardized object whose properties represent the externally visible characteristics of a value that is sampled periodically over a specified time interval. The Averaging object records the minimum, maximum and average value over the interval, and makes these values visible as properties of the Averaging object. The sampled value may be the value of any BOOLEAN, INTEGER, Unsigned, Enumerated or REAL property value of any object within the BACnet Device in which the object resides. Optionally, the object property to be sampled may exist in a different BACnet Device. The Averaging object shall use a "sliding window" technique that maintains a buffer of *N* samples distributed over the specified interval. Every (time interval/N) seconds a new sample is recorded displacing the oldest sample from the buffer. At this time, the minimum, maximum and average are recalculated. The buffer shall maintain an indication for each sample that permits the average calculation and minimum/maximum algorithm to determine the number of valid samples in the buffer.

The Averaging object type and its properties are summarized in Table 12-5 and described in detail in this subclause.

Table 12-5. Properties of the Averaging Object Type

Property Identifier	Property Datatype	Conformance Code
Object_Identifier	BACnetObjectIdentifier	R
Object_Name	CharacterString	R
Object_Type	BACnetObjectType	R
Minimum_Value	REAL	R
Minimum_Value_Timestamp	BACnetDateTime	O
Average_Value	REAL	R
Variance_Value	REAL	O
Maximum_Value	REAL	R
Maximum_Value_Timestamp	BACnetDateTime	O
Description	CharacterString	O
Attempted_Samples	Unsigned	W[1]
Valid_Samples	Unsigned	R
Object_Property_Reference	BACnetDeviceObjectPropertyReference	R[1]
Window_Interval	Unsigned	W[1]
Window_Samples	Unsigned	W[1]
Profile_Name	CharacterString	O

[1] If any of these properties are written to using BACnet services, then all of the buffer samples shall become invalid, 'Attempted_Samples' shall become zero, 'Valid_Samples' shall become zero, 'Minimum_Value' shall become INF, 'Average_Value' shall become NaN and 'Maximum_Value' shall become -INF.

12.5.1 Object_Identifier

This property, of type BACnetObjectIdentifier, is a numeric code that is used to identify the object. It shall be unique within the BACnet Device that maintains it.

12.5.2 Object_Name

This property, of type CharacterString, shall represent a name for the object that is unique within the BACnet Device that maintains it. The minimum length of the string shall be one character. The set of characters used in the Object_Name shall be restricted to printable characters.

12.5.3 Object_Type

This property, of type BACnetObjectType, indicates membership in a particular object type class. The value of this property shall be AVERAGING.

12.5.4 Minimum_Value

This property, of type REAL, shall reflect the lowest value contained within the buffer window for the most recent 'Window_Samples' samples, or the actual number of samples ('Valid_Samples') if less than 'Window_Samples' samples have

been taken. After a device restart, or after 'Attempted_Samples', 'Object_Property_Reference', 'Window_Samples' or 'Window_Interval' are written to using BACnet services, until a sample is taken, 'Minimum_Value' shall have the value **INF**.

12.5.5 Minimum_Value_Timestamp

This optional property, of type BACnetDateTime, indicates the date and time at which the value stored in Minimum_Value was sampled.

12.5.6 Average_Value

This property, of type REAL, shall reflect the average value contained within the buffer window for the most recent 'Window_Samples' samples, or the actual number of samples ('Valid_Samples') if less than 'Window_Samples' samples have been taken. The average shall be calculated by taking the arithmetic sum of all non-missed buffer samples and dividing by the number of non-missed buffer samples. After a device restart, or after 'Attempted_Samples', 'Object_Property_Reference', 'Window_Samples' or 'Window_Interval' are written to using BACnet services, until a sample is taken, 'Average_Value' shall have the value **NaN**.

12.5.7 Variance_Value

This optional property, of type REAL, shall reflect the variance value contained within the buffer window for the most recent 'Window_Samples' samples, or the actual number of samples ('Valid_Samples') if less than 'Window_Samples' samples have been taken. After a device restart, or after 'Attempted_Samples', 'Object_Property_Reference', 'Window_Samples' or 'Window_Interval' are written to using BACnet services, until a sample is taken, 'Variance_Value' shall have the value **NaN**.

12.5.8 Maximum_Value

This property, of type REAL, shall reflect the highest value contained within the buffer window for the most recent 'Window_Samples' samples, or the actual number of samples ('Valid_Samples') if less than 'Window_Samples' samples have been taken. After a device restart, or after 'Attempted_Samples', 'Object_Property_Reference', 'Window_Samples' or 'Window_Interval' are written to using BACnet services, until a sample is taken, 'Maximum_Value' shall have the value **-INF**.

12.5.9 Maximum_Value_Timestamp

This optional property, of type BACnetDateTime, indicates the date and time at which the value stored in Maximum_Value was sampled.

12.5.10 Description

This property, of type CharacterString, is a string of printable characters whose content is not restricted.

12.5.11 Attempted_Samples

This property, of type Unsigned, indicates the number of samples that have been attempted to be collected for the current window. The only acceptable value that may be written to this property shall be zero. If 'Attempted_Samples' is less than 'Window_Samples' then a period of time less than 'Window_Interval' has elapsed since a device restart, or 'Attempted_Samples', 'Object_Property_Reference', 'Window_Samples' or 'Window_Interval' have been written to using BACnet services. The number of missed samples in the current window can be calculated by subtracting 'Valid_Samples' from 'Attempted_Samples.' After a device restart, or after 'Attempted_Samples', 'Object_Property_Reference', 'Window_Samples' or 'Window_Interval' are written to using BACnet services, until a sample is taken, 'Attempted_Samples' shall have the value zero.

12.5.12 Valid_Samples

This read-only property, of type Unsigned, indicates the number of samples that have been successfully collected for the current window. This value can be used to determine whether any of the samples in the current 'Window_Interval' are missing. The number of missed samples in the current window can be calculated by subtracting 'Valid_Samples' from 'Attempted_Samples.' A result greater than zero indicates the number of samples that encountered an error when the sample was being recorded. After a device restart, or after 'Attempted_Samples', 'Object_Property_Reference', 'Window_Samples' or 'Window_Interval' are written to using BACnet services until a sample is taken, 'Valid_Samples' shall have the value **zero**.

12.5.13 Object_Property_Reference

This property, of type BACnetDeviceObjectPropertyReference, shall identify the object and property whose value is to be sampled during the 'Window_Interval'. The object referenced may be located within the device containing the Averaging

object, or optionally the Averaging object may support the referencing of object properties in other devices. External references may be restricted to a particular set of BACnet devices. The referenced object property must have any of the numeric datatypes BOOLEAN, INTEGER, Unsigned, Enumerated or REAL. All sampled data shall be converted to REAL for calculation purposes. BOOLEAN FALSE shall be considered to be zero and TRUE shall be considered to be one. Enumerated datatypes shall be treated as Unsigned values. If an implementation supports writing to 'Object_Property_Reference', then if 'Object_Property_Reference' is written to using BACnet services, then all of the buffer samples shall become invalid, 'Attempted_Samples' shall become zero, 'Valid_Samples' shall become zero, 'Minimum_Value' shall become INF, 'Average_Value' shall become NaN and 'Maximum_Value' shall become -INF.

12.5.14 Window_Interval

This property, of type Unsigned, shall indicate the period of time in seconds over which the minimum, maximum and average values are calculated. The minimum acceptable value for 'Window_Interval' shall be a local matter. Every 'Window_Interval' divided by 'Window_Samples' seconds a new sample shall be taken by reading the value of the property referenced by the 'Object_Property_Reference'. Whether the sample represents an instantaneous "snapshot" or a continuously calculated sample shall be a local matter. If 'Window_Interval' is written to using BACnet services, then all of the buffer samples shall become invalid, 'Attempted_Samples' shall become zero, 'Valid_Samples' shall become zero, 'Minimum_Value' shall become INF, 'Average_Value' shall become NaN and 'Maximum_Value' shall become -INF.

12.5.15 Window_Samples

This property, of type Unsigned, shall indicate the number of samples to be taken during the period of time specified by the 'Window_Interval' property. 'Window_Samples' must be greater than zero and all implementations shall support at least 15 samples. Every 'Window_Interval' divided by 'Window_Samples' seconds a new sample shall be taken by reading the value of the property referenced by the 'Object_Property_Reference'. Whether the sample represents an instantaneous "snapshot" or a continuously calculated sample shall be a local matter. If 'Window_Samples' is written to using BACnet services, then all of the buffer samples shall become invalid, 'Attempted_Samples' shall become zero, 'Valid_Samples' shall become zero, 'Minimum_Value' shall become INF, 'Average_Value' shall become NaN and 'Maximum_Value' shall become -INF.

12.5.16 Profile_Name

This optional property, of type CharacterString, is the name of an object profile to which this object conforms. To ensure uniqueness, a profile name must begin with a vendor identifier code (see Clause 23) in base-10 integer format, followed by a dash. All subsequent characters are administered by the organization registered with that vendor identifier code. The vendor identifier code that prefixes the profile name shall indicate the organization that publishes and maintains the profile document named by the remainder of the profile name. This vendor identifier need not have any relationship to the vendor identifier of the device within which the object resides.

A profile defines a set of additional properties, behavior, and/or requirements for this object beyond those specified here. This standard defines only the format of the names of profiles. The definition of the profiles themselves is outside the scope of this standard.

12.6 Binary Input Object Type

The Binary Input object type defines a standardized object whose properties represent the externally visible characteristics of a binary input. A "binary input" is a physical device or hardware input that can be in only one of two distinct states. In this description, those states are referred to as ACTIVE and INACTIVE. A typical use of a binary input is to indicate whether a particular piece of mechanical equipment, such as a fan or pump, is running or idle. The state ACTIVE corresponds to the situation when the equipment is on or running, and INACTIVE corresponds to the situation when the equipment is off or idle.

In some applications, electronic circuits may reverse the relationship between the application-level logical states ACTIVE and INACTIVE and the physical state of the underlying hardware. For example, a normally open relay contact may result in an ACTIVE state when the relay is energized, while a normally closed relay contact may result in an INACTIVE state when the relay is energized. The Binary Input object provides for this possibility by including a Polarity property. See 12.6.4 and 12.6.11.

The object and its properties are summarized in Table 12-6 and described in detail in this subclause.

Table 12-6. Properties of the Binary Input Object Type

Property Identifier	Property Datatype	Conformance Code
Object_Identifier	BACnetObjectIdentifier	R
Object_Name	CharacterString	R
Object_Type	BACnetObjectType	R
Present_Value	BACnetBinaryPV	R[1]
Description	CharacterString	O
Device_Type	CharacterString	O
Status_Flags	BACnetStatusFlags	R
Event_State	BACnetEventState	R
Reliability	BACnetReliability	O
Out_Of_Service	BOOLEAN	R
Polarity	BACnetPolarity	R
Inactive_Text	CharacterString	O[2]
Active_Text	CharacterString	O[2]
Change_Of_State_Time	BACnetDateTime	O[3]
Change_Of_State_Count	Unsigned	O[3]
Time_Of_State_Count_Reset	BACnetDateTime	O[3]
Elapsed_Active_Time	Unsigned32	O[4]
Time_Of_Active_Time_Reset	BACnetDateTime	O[4]
Time_Delay	Unsigned	O[5]
Notification_Class	Unsigned	O[5]
Alarm_Value	BACnetBinaryPV	O[5]
Event_Enable	BACnetEventTransitionBits	O[5]
Acked_Transitions	BACnetEventTransitionBits	O[5]
Notify_Type	BACnetNotifyType	O[5]
Event_Time_Stamps	BACnetARRAY[3] of BACnetTimeStamp	O[5]
Profile_Name	CharacterString	O

[1] This property is required to be writable when Out_Of_Service is TRUE.
[2] If one of the optional properties Inactive_Text or Active_Text is present, then both of these properties shall be present.
[3] If one of the optional properties Change_Of_State_Time, Change_Of_State_Count, or Time_Of_State_Count_Reset is present, then all of these properties shall be present.
[4] If one of the optional properties Elapsed_Active_Time or Time_Of_Active_Time_Reset is present, then both of these properties shall be present.
[5] These properties are required if the object supports intrinsic reporting.

12.6.1 Object_Identifier

This property, of type BACnetObjectIdentifier, is a numeric code that is used to identify the object. It shall be unique within the BACnet Device that maintains it.

12.6.2 Object_Name

This property, of type CharacterString, shall represent a name for the object that is unique within the BACnet Device that maintains it. The minimum length of the string shall be one character. The set of characters used in the Object_Name shall be restricted to printable characters.

12.6.3 Object_Type

This property, of type BACnetObjectType, indicates membership in a particular object-type class. The value of this property shall be BINARY_INPUT.

12.6.4 Present_Value

This property, of type BACnetBinaryPV, reflects the logical state of the Binary Input. The logical state of the Input shall be either INACTIVE or ACTIVE. The relationship between the Present_Value and the physical state of the Input is determined by the Polarity property. The possible states are summarized in Table 12-7.

Table 12-7. BACnet Polarity Relationships

Present_Value	Polarity	Physical State of Input	Physical State of Device
INACTIVE	NORMAL	OFF or INACTIVE	not running
ACTIVE	NORMAL	ON or ACTIVE	Running
INACTIVE	REVERSE	ON or ACTIVE	not running
ACTIVE	REVERSE	OFF or INACTIVE	running

The Present_Value property shall be writable when Out_Of_Service is TRUE.

12.6.5 Description

This property, of type CharacterString, is a string of printable characters whose content is not restricted.

12.6.6 Device_Type

This property, of type CharacterString, is a text description of the physical device connected to the binary input. It will typically be used to describe the type of device attached to the binary input.

12.6.7 Status_Flags

This property, of type BACnetStatusFlags, represents four Boolean flags that indicate the general "health" of a binary input. Three of the flags are associated with the values of other properties of this object. A more detailed status could be determined by reading the properties that are linked to these flags. The relationship between individual flags is not defined by the protocol. The four flags are

{IN_ALARM, FAULT, OVERRIDDEN, OUT_OF_SERVICE}

where:

IN_ALARM — Logical FALSE (0) if the Event_State property has a value of NORMAL, otherwise logical TRUE (1).

FAULT — Logical TRUE (1) if the Reliability property is present and does not have a value of NO_FAULT_DETECTED, otherwise logical FALSE (0).

OVERRIDDEN — Logical TRUE (1) if the point has been overridden by some mechanism local to the BACnet Device. In this context "overridden" is taken to mean that the Present_Value and Reliability properties are no longer tracking changes to the physical input. Otherwise, the value is logical FALSE (0).

OUT_OF_SERVICE — Logical TRUE (1) if the Out_Of_Service property has a value of TRUE, otherwise logical FALSE (0).

12.6.8 Event_State

The Event_State property, of type BACnetEventState, is included in order to provide a way to determine if this object has an active event state associated with it. If the object supports intrinsic reporting, then the Event_State property shall indicate the event state of the object. If the object does not support intrinsic reporting, then the value of this property shall be NORMAL. If the Reliability property is present and does not have a value of NO_FAULT_DETECTED, then the value of the Event_State property shall be FAULT. Changes in the Event_State property to the value FAULT are considered to be "fault" events.

12.6.9 Reliability

The Reliability property, of type BACnetReliability, provides an indication of whether the Present_Value or the operation of the physical input in question is "reliable" as far as the BACnet Device or operator can determine and, if not, why. The Reliability property for this object may have any of the following values:

{NO_FAULT_DETECTED, NO_SENSOR, OPEN_LOOP, SHORTED_LOOP, UNRELIABLE_OTHER}.

12.6.10 Out_Of_Service

The Out_Of_Service property, of type BOOLEAN, is an indication whether (TRUE) or not (FALSE) the physical input the object represents is not in service. This means that the Present_Value property is decoupled from the physical input and will not track changes to the physical input when the value of Out_Of_Service is TRUE. In addition, the Reliability property and the corresponding state of the FAULT flag of the Status_Flags property shall be decoupled from the physical input when Out_Of_Service is TRUE. While the Out_Of_Service property is TRUE, the Present_Value and Reliability properties may be changed to any value as a means of simulating specific fixed conditions or for testing purposes. Other functions that depend on the state of the Present_Value or Reliability properties shall respond to changes made to these properties while Out_Of_Service is TRUE, as if those changes had occurred in the physical input.

12.6.11 Polarity

This property, of type BACnetPolarity, indicates the relationship between the physical state of the Input and the logical state represented by the Present_Value property. If the Polarity property is NORMAL, then the ACTIVE state of the Present_Value property is also the ACTIVE or ON state of the physical Input as long as Out_Of_Service is FALSE. If the Polarity property is REVERSE, then the ACTIVE state of the Present_Value property is the INACTIVE or OFF state of the physical Input as long as Out_Of_Service is FALSE. See Table 12-7. Therefore, when Out_Of_Service is FALSE for a constant physical input state, a change in the Polarity property shall produce a change in the Present_Value property. If Out_Of_Service is TRUE, then the Polarity property shall have no effect on the Present_Value property.

12.6.12 Inactive_Text

This property, of type CharacterString, characterizes the intended effect of the INACTIVE state of the Present_Value property from the human operator's viewpoint. The content of this string is a local matter, but it is intended to represent a human-readable description of the INACTIVE state. For example, if the physical input is connected to a switch contact, then the Inactive_Text property might be assigned a value such as "Fan 1 Off". If either the Inactive_Text property or the Active_Text property are present, then both of them shall be present.

12.6.13 Active_Text

This property, of type CharacterString, characterizes the intended effect of the ACTIVE state of the Present_Value property from the human operator's viewpoint. The content of this string is a local matter, but it is intended to represent a human-readable description of the ACTIVE state. For example, if the physical input is a switch contact, then the Active_Text property might be assigned a value such as "Fan 1 On". If either the Active_Text property or the Inactive_Text property are present, then both of them shall be present.

12.6.14 Change_Of_State_Time

This property, of type BACnetDateTime, represents the date and time at which the most recent change of state occurred. A "change of state" shall be defined as any event that alters the Present_Value property. When Out_Of_Service is FALSE, a change to the Polarity property shall alter Present_Value and thus be considered a change of state. When Out_Of_Service is TRUE, changes to Polarity shall not cause changes of state. If one of the optional properties Change_Of_State_Time, Change_Of_State_Count, or Time_Of_State_Count_Reset is present, then all of these properties shall be present.

12.6.15 Change_Of_State_Count

This property, of type Unsigned, represents the number of times that the Present_Value property has changed state since the Change_Of_State_Count property was most recently set to a zero value. The Change_Of_State_Count property shall have a range of 0-65535 or greater. A "change of state" shall be defined as any event that alters the Present_Value property. When Out_Of_Service is FALSE, a change to the Polarity property shall alter Present_Value and thus be considered a change of state. When Out_Of_Service is TRUE, changes to Polarity shall not cause changes of state. If one of the optional properties Change_Of_State_Time, Change_Of_State_Count, or Time_Of_State_Count_Reset is present, then all of these properties shall be present.

12.6.16 Time_Of_State_Count_Reset

This property, of type BACnetDateTime, represents the date and time at which the Change_Of_State_Count property was most recently set to a zero value. If one of the optional properties Change_Of_State_Time, Change_Of_State_Count, or Time_Of_State_Count_Reset is present, then all of these properties shall be present.

12.6.17 Elapsed_Active_Time

This property, of type Unsigned32, represents the accumulated number of seconds that the Present_Value property has had the value ACTIVE since the Elapsed_Active_Time property was most recently set to a zero value. If one of the optional properties Elapsed_Active_Time or Time_Of_Active_Time_Reset are present, then both of these properties shall be present.

12.6.18 Time_Of_Active_Time_Reset

This property, of type BACnetDateTime, represents the date and time at which the Elapsed_Active_Time property was most recently set to a zero value. If one of the optional properties Elapsed_Active_Time or Time_Of_Active_Time_Reset are present, then both of these properties shall be present.

12.6.19 Time_Delay

This property, of type Unsigned, shall specify the minimum period of time in seconds during which the Present_Value must remain equal to the Alarm_Value property before a TO-OFFNORMAL event is generated, or remain not equal to the Alarm_Value property before a TO-NORMAL event is generated. This property is required if intrinsic reporting is supported by this object.

12.6.20 Notification_Class

This property, of type Unsigned, shall specify the notification class to be used when handling and generating event notifications for this object. The Notification_Class property implicitly refers to a Notification Class object that has a Notification_Class property with the same value. This property is required if intrinsic reporting is supported by this object.

12.6.21 Alarm_Value

This property, of type BACnetBinaryPV, shall specify the value that the Present_Value must have before an event is generated. This property is required if intrinsic reporting is supported by this object.

12.6.21.1 Conditions for Generating a TO-OFFNORMAL Event

A TO-OFFNORMAL event is generated under these conditions:

(a) the Present_Value must maintain the value specified by Alarm_Value for a minimum period of time, specified in the Time_Delay property, and
(b) the TO-OFFNORMAL flag must be enabled in the Event_Enable property.

12.6.21.2 Conditions for Generating a TO-NORMAL Event

Once equal, the Present_Value must become not equal to this property before a TO-NORMAL event is generated under these conditions:

(a) the Present_Value must remain not equal to the Alarm_Value for a minimum period of time, specified by the Time_Delay property, and
(b) the TO-NORMAL flag must be enabled in the Event_Enable property.

12.6.22 Event_Enable

This property, of type BACnetEventTransitionBits, shall convey three flags that separately enable and disable reporting of TO-OFFNORMAL, TO-FAULT, and TO-NORMAL events. This property is required if intrinsic reporting is supported by this object.

12.6.23 Acked_Transitions

This property, of type BACnetEventTransitionBits, shall convey three flags that separately indicate the receipt of acknowledgments for TO-OFFNORMAL, TO-FAULT, and TO-NORMAL events. These flags shall be cleared upon the occurrence of the corresponding event and set under any of these conditions:

(a) upon receipt of the corresponding acknowledgment;
(b) upon the occurrence of the event if the corresponding flag is not set in the Event_Enable property (meaning event notifications will not be generated for this condition and thus no acknowledgment is expected);
(c) upon the occurrence of the event if the corresponding flag is set in the Event_Enable property and the corresponding flag in the Ack_Required property of the Notification Class object implicitly referenced by the Notification_Class property of this object is not set (meaning no acknowledgment is expected).

This property is required if intrinsic reporting is supported by this object.

12.6.24 Notify_Type

This property, of type BACnetNotifyType, shall convey whether the notifications generated by the object should be Events or Alarms. This property is required if intrinsic reporting is supported by this object.

12.6.25 Event_Time_Stamps

This optional property, of type BACnetARRAY[3] of BACnetTimeStamp, shall convey the times of the last event notifications for TO-OFFNORMAL, TO-FAULT, and TO-NORMAL events, respectively. Time stamps of type Time or Date shall have 'FF' in each octet and Sequence number time stamps shall have the value 0 if no event notification of that type has been generated since the object was created. This property is required if intrinsic reporting is supported by this object.

12.6.26 Profile_Name

This optional property, of type CharacterString, is the name of an object profile to which this object conforms. To ensure uniqueness, a profile name must begin with a vendor identifier code (see Clause 23) in base-10 integer format, followed by a dash. All subsequent characters are administered by the organization registered with that vendor identifier code. The vendor identifier code that prefixes the profile name shall indicate the organization that publishes and maintains the profile document named by the remainder of the profile name. This vendor identifier need not have any relationship to the vendor identifier of the device within which the object resides.

A profile defines a set of additional properties, behavior, and/or requirements for this object beyond those specified here. This standard defines only the format of the names of profiles. The definition of the profiles themselves is outside the scope of this standard.

12.7 Binary Output Object Type

The Binary Output object type defines a standardized object whose properties represent the externally visible characteristics of a binary output. A "binary output" is a physical device or hardware output that can be in only one of two distinct states. In this description, those states are referred to as ACTIVE and INACTIVE. A typical use of a binary output is to switch a particular piece of mechanical equipment, such as a fan or pump, on or off. The state ACTIVE corresponds to the situation when the equipment is on or running, and INACTIVE corresponds to the situation when the equipment is off or idle.

In some applications, electronic circuits may reverse the relationship between the application-level logical states, ACTIVE and INACTIVE, and the physical state of the underlying hardware. For example, a normally open relay contact may result in an ACTIVE state (device energized) when the relay is energized, while a normally closed relay contact may result in an ACTIVE state (device energized) when the relay is not energized. The Binary Output object provides for this possibility by including a Polarity property. See 12.7.4 and 12.7.11.

The object and its properties are summarized in Table 12-8 and described in detail in this subclause.

Table 12-8. Properties of the Binary Output Object Type

Property Identifier	Property Datatype	Conformance Code
Object_Identifier	BACnetObjectIdentifier	R
Object_Name	CharacterString	R
Object_Type	BACnetObjectType	R
Present_Value	BACnetBinaryPV	W
Description	CharacterString	O
Device_Type	CharacterString	O
Status_Flags	BACnetStatusFlags	R
Event_State	BACnetEventState	R
Reliability	BACnetReliability	O
Out_Of_Service	BOOLEAN	R
Polarity	BACnetPolarity	R
Inactive_Text	CharacterString	O[1]
Active_Text	CharacterString	O[1]
Change_Of_State_Time	BACnetDateTime	O[2]
Change_Of_State_Count	Unsigned	O[2]
Time_Of_State_Count_Reset	BACnetDateTime	O[2]
Elapsed_Active_Time	Unsigned32	O[3]
Time_Of_Active_Time_Reset	BACnetDateTime	O[3]
Minimum_Off_Time	Unsigned32	O
Minimum_On_Time	Unsigned32	O
Priority_Array	BACnetPriorityArray	R
Relinquish_Default	BACnetBinaryPV	R
Time_Delay	Unsigned	O[4]
Notification_Class	Unsigned	O[4]
Feedback_Value	BACnetBinaryPV	O[4]
Event_Enable	BACnetEventTransitionBits	O[4]
Acked_Transitions	BACnetEventTransitionBits	O[4]
Notify_Type	BACnetNotifyType	O[4]
Event_Time_Stamps	BACnetARRAY[3] of BACnetTimeStamp	O[4]
Profile_Name	CharacterString	O

[1] If one of the optional properties Inactive_Text or Active_Text is present, then both of these properties shall be present.
[2] If one of the optional properties Change_Of_State_Time, Change_Of_State_Count, or Time_Of_State_Count_Reset is present, then all of these properties shall be present.
[3] If one of the optional properties Elapsed_Active_Time or Time_Of_Active_Time_Reset is present, then both of these properties shall be present.
[4] These properties are required if the object supports intrinsic reporting.

12.7.1 Object_Identifier

This property, of type BACnetObjectIdentifier, is a numeric code that is used to identify the object. It shall be unique within the BACnet Device that maintains it.

12.7.2 Object_Name

This property, of type CharacterString, shall represent a name for the object that is unique within the BACnet Device that maintains it. The minimum length of the string shall be one character. The set of characters used in the Object_Name shall be restricted to printable characters.

12.7.3 Object_Type

This property, of type BACnetObjectType, indicates membership in a particular object type class. The value of this property shall be BINARY_OUTPUT.

12.7.4 Present_Value (Commandable)

This property, of type BACnetBinaryPV, reflects the logical state of the Binary Output. The logical state of the output shall be either INACTIVE or ACTIVE. The relationship between the Present_Value and the physical state of the output is determined by the Polarity property. The possible states are summarized in Table 12-9.

Table 12-9. BACnet Polarity Relationships

Present_Value	Polarity	Physical State of Output	Physical State of Device
INACTIVE	NORMAL	OFF or INACTIVE	not running
ACTIVE	NORMAL	ON or ACTIVE	running
INACTIVE	REVERSE	ON or ACTIVE	not running
ACTIVE	REVERSE	OFF or INACTIVE	running

12.7.5 Description

This property, of type CharacterString, is a string of printable characters whose content is not restricted.

12.7.6 Device_Type

This property, of type CharacterString, is a text description of the physical device connected to the binary output. It will typically be used to describe the type of device attached to the binary output.

12.7.7 Status_Flags

This property, of type BACnetStatusFlags, represents four Boolean flags that indicate the general "health" of a binary output. Three of the flags are associated with the values of other properties of this object. A more detailed status could be determined by reading the properties that are linked to these flags. The relationship between individual flags is not defined by the protocol. The four flags are

{IN_ALARM, FAULT, OVERRIDDEN, OUT_OF_SERVICE}

where:

IN_ALARM — Logical FALSE (0) if the Event_State property has a value of NORMAL, otherwise logical TRUE (1).

FAULT — Logical TRUE (1) if the Reliability property is present and does not have a value of NO_FAULT_DETECTED, otherwise logical FALSE (0).

OVERRIDDEN	Logical TRUE (1) if the point has been overridden by some mechanism local to the BACnet Device. In this context "overridden" is taken to mean that the physical output is no longer tracking changes to the Present_Value property and the Reliability property is no longer a reflection of the physical output. Otherwise, the value is logical FALSE (0).
OUT_OF_SERVICE	Logical TRUE (1) if the Out_Of_Service property has a value of TRUE, otherwise logical FALSE(0).

12.7.8 Event_State

The Event_State property, of type BACnetEventState, is included in order to provide a way to determine if this object has an active event state associated with it. If the object supports intrinsic reporting, then the Event_State property shall indicate the event state of the object. If the object does not support intrinsic reporting, then the value of this property shall be NORMAL. If the Reliability property is present and does not have a value of NO_FAULT_DETECTED, then the value of the Event_State property shall be FAULT. Changes in the Event_State property to the value FAULT are considered to be "fault" events.

12.7.9 Reliability

The Reliability property, of type BACnetReliability, provides an indication of whether the Present_Value or the operation of the physical output in question is "reliable" as far as the BACnet Device or operator can determine and, if not, why. The Reliability property for this object type may have any of the following values:

{NO_FAULT_DETECTED, NO_OUTPUT, OPEN_LOOP, SHORTED_LOOP, UNRELIABLE_OTHER}.

12.7.10 Out_Of_Service

The Out_Of_Service property, of type BOOLEAN, is an indication whether (TRUE) or not (FALSE) the physical point the object represents is not in service. This means that changes to the Present_Value property are decoupled from the physical output when the value of Out_Of_Service is TRUE. In addition, the Reliability property and the corresponding state of the FAULT flag of the Status_Flags property shall be decoupled from the physical output when Out_Of_Service is TRUE. While the Out_Of_Service property is TRUE, the Present_Value and Reliability properties may still be changed to any value as a means of simulating specific fixed conditions or for testing purposes. Other functions that depend on the state of the Present_Value or Reliability properties shall respond to changes made to these properties while Out_Of_Service is TRUE, as if those changes had occurred to the physical output. The Present_Value property shall still be controlled by the BACnet command prioritization mechanism if Out_Of_Service is TRUE. See Clause 19.

12.7.11 Polarity

This property, of type BACnetPolarity, indicates the relationship between the physical state of the output and the logical state represented by the Present_Value property. If the Polarity property is NORMAL, then the ACTIVE state of the Present_Value property is also the ACTIVE or ON state of the physical output as long as Out_Of_Service is FALSE. If the Polarity property is REVERSE, then the ACTIVE state of the Present_Value property is the INACTIVE or OFF state of the physical output as long as Out_Of_Service is FALSE. See Table 12-9. If Out_Of_Service is TRUE, then the Polarity property shall have no effect on the physical output state**.**

12.7.12 Inactive_Text

This property, of type CharacterString, characterizes the intended effect, from the human operator's viewpoint, of the INACTIVE state of the Present_Value property on the final device that is ultimately controlled by the output. The content of this string is a local matter, but it is intended to represent a human-readable description of the INACTIVE state. For example, if the physical output is a relay contact that turns on a light, then the Inactive_Text property might be assigned a value such as "Light Off". If one of the optional properties Inactive_Text or Active_Text is present, then both of these properties shall be present.

12.7.13 Active_Text

This property, of type CharacterString, characterizes the intended effect, from the human operator's viewpoint, of the ACTIVE state of the Present_Value property on the final device that is ultimately controlled by the output. The content of this string is a local matter, but it is intended to represent a human-readable description of the ACTIVE state. For example, if the physical output is a relay contact that turns on a light, then the Active_Text property might be assigned a value such as

"Light On". If one of the optional properties Inactive_Text or Active_Text is present, then both of these properties shall be present.

12.7.14 Change_Of_State_Time

This property, of type BACnetDateTime, represents the date and time at which the most recent change of state occurred. A "change of state" shall be defined as any event that alters the Present_Value property. Changes to Polarity shall not cause changes of state. If one of the optional properties Change_Of_State_Time, Change_Of_State_Count, or Time_Of_State_Count_Reset is present, then all of these properties shall be present.

12.7.15 Change_Of_State_Count

This property, of type Unsigned, represents the number of times that the Present_Value property has changed state since the Change_Of_State_Count property was most recently set to a zero value. The Change_Of_State_Count property shall have a range of 0-65535 or greater. A "change of state" shall be defined as any event that alters the Present_Value property. Changes to Polarity shall not cause changes of state. If one of the optional properties Change_Of_State_Time, Change_Of_State_Count, or Time_Of_State_Count_Reset is present, then all of these properties shall be present.

12.7.16 Time_Of_State_Count_Reset

This property, of type BACnetDateTime, represents the date and time at which the Change_Of_State_Count property was most recently set to a zero value. If one of the optional properties Change_Of_State_Time, Change_Of_State_Count, or Time_Of_State_Count_Reset is present, then all of these properties shall be present.

12.7.17 Elapsed_Active_Time

This property, of type Unsigned32, represents the accumulated number of seconds that the Present_Value property has had the value ACTIVE since this property was most recently set to a zero value. If one of the optional properties Elapsed_Active_Time or Time_Of_Active_Time_Reset is present, then both of these properties shall be present.

12.7.18 Time_Of_Active_Time_Reset

This property, of type BACnetDateTime, represents the date and time at which the Elapsed_Active_Time property was most recently set to a zero value. If one of the optional properties Elapsed_Active_Time or Time_Of_Active_Time_Reset is present, then both of these properties shall be present.

12.7.19 Minimum_Off_Time

This property, of type Unsigned32, represents the minimum number of seconds that the Present_Value shall remain in the INACTIVE state after a write to the Present_Value property causes that property to assume the INACTIVE state.

The mechanism by which this is accomplished is described in 19.2.3.

12.7.20 Minimum_On_Time

This property, of type Unsigned32, represents the minimum number of seconds that the Present_Value shall remain in the ACTIVE state after a write to the Present_Value property causes that property to assume the ACTIVE state.

The mechanism by which this is accomplished is described in 19.2.3.

12.7.21 Priority_Array

This property is a read-only array that contains prioritized commands that are in effect for this object. See Clause 19 for a description of the prioritization mechanism.

12.7.22 Relinquish_Default

This property is the default value to be used for the Present_Value property when all command priority values in the Priority_Array property have a NULL value. See Clause 19.

12.7.23 Time_Delay

This property, of type Unsigned, shall specify the minimum period of time in seconds during which the Present_Value must be different from the Feedback_Value property before a TO-OFFNORMAL event is generated or must remain equal to the Feedback_Value property before a TO-NORMAL event is generated. This property is required if intrinsic reporting is supported by this object.

12.7.24 Notification_Class

This property, of type Unsigned, shall specify the notification class to be used when handling and generating event notifications for this object. The Notification_Class property implicitly refers to a Notification Class object that has a Notification_Class property with the same value. This property is required if intrinsic reporting is supported by this object.

12.7.25 Feedback_Value

This property, of type BACnetBinaryPV, shall indicate the status of a feedback value from which the Present_Value must differ before an event is generated. This property is required if intrinsic reporting is supported by this object. The manner by which the Feedback_Value is determined shall be a local matter.

12.7.25.1 Conditions for Generating a TO-NORMAL Event

A TO-OFFNORMAL event is generated under these conditions:

(a) the Present_Value must be different from the Feedback_Value for a minimum period of time, specified by the Time_Delay property, and
(b) the TO-OFFNORMAL flag must be enabled in the Event_Enable property.

12.7.25.2 Conditions for Generating a TO-NORMAL Event

Once different, the Present_Value must become equal to this property before a TO-NORMAL event is generated under these conditions:

(a) the Present_Value must remain equal to the Feedback_Value for a minimum period of time, specified by the Time_Delay property, and
(b) the TO-NORMAL flag must be enabled in the Event_Enable property.

12.7.26 Event_Enable

This property, of type BACnetEventTransitionBits, shall convey three flags that separately enable and disable reporting of TO-OFFNORMAL, TO-FAULT, and TO-NORMAL events. This property is required if intrinsic reporting is supported by this object.

12.7.27 Acked_Transitions

This property, of type BACnetEventTransitionBits, shall convey three flags that separately indicate the receipt of acknowledgments for TO-OFFNORMAL, TO-FAULT, and TO-NORMAL events. These flags shall be cleared upon the occurrence of the corresponding event and set under any of these conditions:

(a) upon receipt of the corresponding acknowledgment;
(b) upon the occurrence of the event if the corresponding flag is not set in the Event_Enable property (meaning event notifications will not be generated for this condition and thus no acknowledgment is expected);
(c) upon the occurrence of the event if the corresponding flag is set in the Event_Enable property and the corresponding flag in the Ack_Required property of the Notification Class object implicitly referenced by the Notification_Class property of this object is not set (meaning no acknowledgment is expected).

This property is required if intrinsic reporting is supported by this object.

12.7.28 Notify_Type

This property, of type BACnetNotifyType, shall convey whether the notifications generated by the object should be Events or Alarms. This property is required if intrinsic reporting is supported by this object.

12.7.29 Event_Time_Stamps

This optional property, of type BACnetARRAY[3] of BACnetTimeStamp, shall convey the times of the last event notifications for TO-OFFNORMAL, TO-FAULT, and TO-NORMAL events, respectively. Time stamps of type Time or Date shall have 'FF' in each octet and Sequence number time stamps shall have the value 0 if no event notification of that type has been generated since the object was created. This property is required if intrinsic reporting is supported by this object.

12.7.30 Profile_Name

This optional property, of type CharacterString, is the name of an object profile to which this object conforms. To ensure uniqueness, a profile name must begin with a vendor identifier code (see Clause 23) in base-10 integer format, followed by a dash. All subsequent characters are administered by the organization registered with that vendor identifier code. The vendor identifier code that prefixes the profile name shall indicate the organization that publishes and maintains the profile document named by the remainder of the profile name. This vendor identifier need not have any relationship to the vendor identifier of the device within which the object resides.

A profile defines a set of additional properties, behavior, and/or requirements for this object beyond those specified here. This standard defines only the format of the names of profiles. The definition of the profiles themselves is outside the scope of this standard.

12.8 Binary Value Object Type

The Binary Value object type defines a standardized object whose properties represent the externally visible characteristics of a binary value. A "binary value" is a control system parameter residing in the memory of the BACnet Device. This parameter may assume only one of two distinct states. In this description, those states are referred to as ACTIVE and INACTIVE. The Binary Value object and its properties are summarized in Table 12-10 and described in detail in this subclause.

Table 12-10. Properties of the Binary Value Object Type

Property Identifier	Property Datatype	Conformance Code
Object_Identifier	BACnetObjectIdentifier	R
Object_Name	CharacterString	R
Object_Type	BACnetObjectType	R
Present_Value	BACnetBinaryPV	R[1]
Description	CharacterString	O
Status_Flags	BACnetStatusFlags	R
Event_State	BACnetEventState	R
Reliability	BACnetReliability	O
Out_Of_Service	BOOLEAN	R
Inactive_Text	CharacterString	O[2]
Active_Text	CharacterString	O[2]
Change_Of_State_Time	BACnetDateTime	O[3]
Change_Of_State_Count	Unsigned32	O[3]
Time_Of_State_Count_Reset	BACnetDateTime	O[3]
Elapsed_Active_Time	Unsigned32	O[4]
Time_Of_Active_Time_Reset	BACnetDateTime	O[4]
Minimum_Off_Time	Unsigned32	O
Minimum_On_Time	Unsigned32	O
Priority_Array	BACnetPriorityArray	O[5]
Relinquish_Default	BACnetBinaryPV	O[5]
Time_Delay	Unsigned	O[6]
Notification_Class	Unsigned	O[6]
Alarm_Value	BACnetBinaryPV	O[6]
Event_Enable	BACnetEventTransitionBits	O[6]
Acked_Transitions	BACnetEventTransitionBits	O[6]
Notify_Type	BACnetNotifyType	O[6]
Event_Time_Stamps	BACnetARRAY[3] of BACnetTimeStamp	O[6]
Profile_Name	CharacterString	O

[1] If Present_Value is commandable, then it is required to be writable. This property is required to be writable when Out_Of_Service is TRUE.
[2] If one of the optional properties Inactive_Text or Active_Text is present, then both of these properties shall be present.
[3] If one of the optional properties Change_Of_State_Time, Change_Of_State_Count, or Time_Of_State_Count_Reset is present, then all of these properties shall be present.
[4] If one of the optional properties Elapsed_Active_Time or Time_Of_Active_Time_Reset is present, then both of these properties shall be present.
[5] If Present_Value is commandable, then both of these properties shall be present.
[6] These properties are required if the object supports intrinsic reporting.

12.8.1 Object_Identifier

This property, of type BACnetObjectIdentifier, is a numeric code that is used to identify the object. It shall be unique within the BACnet Device that maintains it.

12.8.2 Object_Name

This property, of type CharacterString, shall represent a name for the object that is unique within the BACnet Device that maintains it. The minimum length of the string shall be one character. The set of characters used in the Object_Name shall be restricted to printable characters.

12.8.3 Object_Type

This property, of type BACnetObjectType, indicates membership in a particular object type class. The value of this property shall be BINARY_VALUE.

12.8.4 Present_Value

This property, of type BACnetBinaryPV, reflects the logical state of the Binary Value. The logical state shall be either INACTIVE or ACTIVE. Present_Value shall be optionally commandable. If Present_Value is commandable for a given instance, then the Priority_Array and Relinquish_Default properties shall also be present for that instance. The Present_Value property shall be writable when Out_Of_Service is TRUE.

12.8.5 Description

This property, of type CharacterString, is a string of printable characters whose content is not restricted**.**

12.8.6 Status_Flags

This property, of type BACnetStatusFlags, represents four Boolean flags that indicate the general "health" of a binary value object. Three of the flags are associated with the values of other properties of this object. A more detailed status could be determined by reading the properties that are linked to these flags. The relationship between individual flags is not defined by the protocol. The four flags are

{IN_ALARM, FAULT, OVERRIDDEN, OUT_OF_SERVICE}

where:

IN_ALARM	Logical FALSE (0) if the Event_State property has a value of NORMAL, otherwise logical TRUE (1).
FAULT	Logical TRUE (1) if the Reliability property is present and does not have a value of NO_FAULT_DETECTED, otherwise logical FALSE (0).
OVERRIDDEN	Logical TRUE (1) if the point has been overridden by some mechanism local to the BACnet Device. In this context "overridden" is taken to mean that the Present_Value property is not changeable through BACnet services. Otherwise, the value is logical FALSE (0).
OUT_OF_SERVICE	Logical TRUE (1) if the Out_Of_Service property has a value of TRUE, otherwise logical FALSE (0).

12.8.7 Event_State

The Event_State property, of type BACnetEventState, is included in order to provide a way to determine if this object has an active event state associated with it. If the object supports intrinsic reporting, then the Event_State property shall indicate the event state of the object. If the object does not support intrinsic reporting, then the value of this property shall be NORMAL. If the Reliability property is present and does not have a value of NO_FAULT_DETECTED, then the value of the Event_State property shall be FAULT. Changes in the Event_State property to the value FAULT are considered to be "fault" events.

12.8.8 Reliability

The Reliability property, of type BACnetReliability, provides an indication of whether the Present_Value is "reliable" as far as the BACnet Device or operator can determine. The Reliability property for this object type may have any of the following values:

{NO_FAULT_DETECTED, UNRELIABLE_OTHER}.

12.8.9 Out_Of_Service

The Out_Of_Service property, of type BOOLEAN, is an indication whether (TRUE) or not (FALSE) the Present_Value of the Binary Value object is prevented from being modified by software local to the BACnet device in which the object resides. When Out_Of_Service is TRUE, the Present_Value property may be written to freely. If the Priority_Array and

Relinquish_Default properties are present, then writing to the Present_Value property shall be controlled by the BACnet command prioritization mechanism. See Clause 19.

12.8.10 Inactive_Text

This property, of type CharacterString, characterizes the intended effect of the INACTIVE state of the Binary Value. The content of this string is a local matter, but it is intended to represent a human-readable description of the INACTIVE state. If one of the optional properties Inactive_Text or Active_Text is present, then both of these properties shall be present.

12.8.11 Active_Text

This property, of type CharacterString, characterizes the intended effect of the ACTIVE state of the Binary Value. The content of this string is a local matter, but it is intended to represent a human-readable description of the ACTIVE state. If one of the optional properties Inactive_Text or Active_Text is present, then both of these properties shall be present.

12.8.12 Change_Of_State_Time

This property, of type BACnetDateTime, represents the date and time at which the most recent change of state occurred. A "change of state" shall be defined as any event that alters the logical state of the Binary Value. If one of the optional properties Change_Of_State_Time, Change_Of_State_Count, or Time_Of_State_Count_Reset is present, then all of these properties shall be present.

12.8.13 Change_Of_State_Count

This property, of type Unsigned32, represents the number of times that the state of the Binary Value has changed since this property was most recently set to a zero value. The Change_Of_State_Count property shall have a range of 0-65535 or greater. If one of the optional properties Change_Of_State_Time, Change_Of_State_Count, or Time_Of_State_Count_Reset is present, then all of these properties shall be present.

12.8.14 Time_Of_State_Count_Reset

This property, of type BACnetDateTime, represents the date and time at which the Change_Of_State_Count property was most recently set to a zero value. If one of the optional properties Change_Of_State_Time, Change_Of_State_Count, or Time_Of_State_Count_Reset is present, then all of these properties shall be present.

12.8.15 Elapsed_Active_Time

This property, of type Unsigned32, represents the accumulated number of seconds that the Present_Value property has had the value ACTIVE since this property was most recently set to a zero value. If one of the optional properties Elapsed_Active_Time or Time_Of_Active_Time_Reset is present, then both of these properties shall be present.

12.8.16 Time_Of_Active_Time_Reset

This property, of type BACnetDateTime, represents the date and time at which the Elapsed_Active_Time property was most recently set to a zero value. If one of the optional properties Elapsed_Active_Time or Time_Of_Active_Time_Reset is present, then both of these properties shall be present.

12.8.17 Minimum_Off_Time

This property, of type Unsigned32, represents the minimum number of seconds that the Present_Value shall remain in the INACTIVE state after a write to the Present_Value property causes that property to assume the INACTIVE state.

If the Present_Value is commandable according to Clause 19, then the mechanism by which this is accomplished is described in 19.2.3. Otherwise, the mechanism is a local matter.

12.8.18 Minimum_On_Time

This property, of type Unsigned32, represents the minimum number of seconds that the Present_Value shall remain in the ACTIVE state after a write to the Present_Value property causes that property to assume the ACTIVE state.

If the Present_Value is commandable according to Clause 19, then the mechanism by which this is accomplished is described in 19.2.3. Otherwise, the mechanism is a local matter.

12.8.19 Priority_Array

This property is a read-only array that contains prioritized commands that are in effect for this object. See Clause 19 for a description of the prioritization mechanism. If either the Relinquish_Default property or the Priority_Array property are present, then both of them shall be present. If Present_Value is commandable, then Priority_Array and Relinquish_Default shall both be present and Present_Value shall be required to be writable.

12.8.20 Relinquish_Default

This property is the default value to be used for the Present_Value property when all command priority values in the Priority_Array property have a NULL value. See Clause 19. If either the Relinquish_Default property or the Priority_Array property are present, then both of them shall be present. If Present_Value is commandable, then Priority_Array and Relinquish_Default shall both be present and Present_Value shall be required to be writable.

12.8.21 Time_Delay

This property, of type Unsigned, shall specify the minimum period of time in seconds during which the Present_Value must be different from the Alarm_Value property before a TO-OFFNORMAL event is generated or must remain equal to the Alarm_Value property before a TO-NORMAL event is generated. This property is required if intrinsic reporting is supported by this object.

12.8.22 Notification_Class

This property, of type Unsigned, shall specify the notification class to be used when handling and generating event notifications for this object. The Notification_Class property implicitly refers to a Notification Class object that has a Notification_Class property with the same value. This property is required if intrinsic reporting is supported by this object.

12.8.23 Alarm_Value

This property, of type BACnetBinaryPV, shall specify the value that the Present_Value property must have before a TO-OFFNORMAL event is generated. This property is required if intrinsic reporting is supported by this object.

12.8.23.1 Conditions for Generating a TO-OFFNORMAL Event

A TO-OFFNORMAL event is generated under these conditions:

(a) the Present_Value must maintain the value specified by Alarm_Value for a minimum period of time, specified by the Time_Delay property, and
(b) the TO-OFFNORMAL flag must be enabled in the Event_Enable property.

12.8.23.2 Conditions for Generating a TO-NORMAL Event

Once equal, the Present_Value must become not equal to this property before a TO-NORMAL event is generated under these conditions:

(a) the Present_Value must remain not equal to the Alarm_Value for a minimum period of time, specified by the Time_Delay property, and
(b) the TO-NORMAL flag must be enabled in the Event_Enable property.

12.8.24 Event_Enable

This property, of type BACnetEventTransitionBits, shall convey three flags that separately enable and disable reporting of TO-OFFNORMAL, TO-FAULT, and TO-NORMAL events. This property is required if intrinsic reporting is supported by this object.

12.8.25 Acked_Transitions

This property, of type BACnetEventTransitionBits, shall convey three flags that separately indicate the receipt of acknowledgments for TO-OFFNORMAL, TO-FAULT, and TO-NORMAL events. These flags shall be cleared upon the occurrence of the corresponding event and set under any of these conditions:

(a) upon receipt of the corresponding acknowledgment;
(b) upon the occurrence of the event if the corresponding flag is not set in the Event_Enable property (meaning event notifications will not be generated for this condition and thus no acknowledgment is expected);

(c) upon the occurrence of the event if the corresponding flag is set in the Event_Enable property and the corresponding flag in the Ack_Required property of the Notification Class object implicitly referenced by the Notification_Class property of this object is not set (meaning no acknowledgment is expected).

This property is required if intrinsic reporting is supported by this object.

12.8.26 Notify_Type

This property, of type BACnetNotifyType, shall convey whether the notifications generated by the object should be Events or Alarms. This property is required if intrinsic reporting is supported by this object.

12.8.27 Event_Time_Stamps

This optional property, of type BACnetARRAY[3] of BACnetTimeStamp, shall convey the times of the last event notifications for TO-OFFNORMAL, TO-FAULT, and TO-NORMAL events, respectively. Time stamps of type Time or Date shall have 'FF' in each octet and Sequence number time stamps shall have the value 0 if no event notification of that type has been generated since the object was created. This property is required if intrinsic reporting is supported by this object.

12.8.28 Profile_Name

This optional property, of type CharacterString, is the name of an object profile to which this object conforms. To ensure uniqueness, a profile name must begin with a vendor identifier code (see Clause 23) in base-10 integer format, followed by a dash. All subsequent characters are administered by the organization registered with that vendor identifier code. The vendor identifier code that prefixes the profile name shall indicate the organization that publishes and maintains the profile document named by the remainder of the profile name. This vendor identifier need not have any relationship to the vendor identifier of the device within which the object resides.

A profile defines a set of additional properties, behavior, and/or requirements for this object beyond those specified here. This standard defines only the format of the names of profiles. The definition of the profiles themselves is outside the scope of this standard.

12.9 Calendar Object Type

The Calendar object type defines a standardized object used to describe a list of calendar dates, which might be thought of as "holidays," "special events," or simply as a list of dates. The object and its properties are summarized in Table 12-11 and described in detail in this subclause.

Table 12-11. Properties of the Calendar Object Type

Property Identifier	Property Datatype	Conformance Code
Object_Identifier	BACnetObjectIdentifier	R
Object_Name	CharacterString	R
Object_Type	BACnetObjectType	R
Description	CharacterString	O
Present_Value	BOOLEAN	R
Date_List	List of BACnetCalendarEntry	R
Profile_Name	CharacterString	O

12.9.1 Object_Identifier

This property, of type BACnetObjectIdentifier, is a numeric code that is used to identify the object. It shall be unique within the BACnet Device that maintains it.

12.9.2 Object_Name

This property, of type CharacterString, shall represent a name for the object that is unique within the BACnet Device that maintains it. The minimum length of the string shall be one character. The set of characters used in the Object_Name shall be restricted to printable characters.

12.9.3 Object_Type

This property, of type BACnetObjectType, indicates membership in a particular object type class. The value of this property shall be CALENDAR.

12.9.4 Description

This property, of type CharacterString, is a string of printable characters whose content is not restricted.

12.9.5 Present_Value

This property, of type BOOLEAN, indicates the current value of the calendar: TRUE if the current date is in the Date_List and FALSE if it is not.

12.9.6 Date_List

This property is a List of BACnetCalendarEntry, each of which is either an individual date (Date), range of dates (BACnetDateRange), or month/week-of-month/day-of-week specification (BACnetWeekNDay). If the current date matches the calendar entry criteria, the present value of the Calendar object is TRUE. Individual fields of the various constructs may also be unspecified in which case the field acts as a "wildcard" for determining if the current date results in a match. In a date range, for example, if the startDate is unspecified, it means "any date up to and including the endDate." If the endDate is unspecified, it means "any date from the startDate on."

If the calendar entry were a BACnetWeekNDay with unspecified month and week-of-month fields but with a specific day-of-week, it would mean the Calendar object would be TRUE on that day-of-week all year long. If a BACnet Device permits writing to the Date_List property, all choices in the BACnetCalendarEntry shall be permitted.

12.9.7 Profile_Name

This optional property, of type CharacterString, is the name of an object profile to which this object conforms. To ensure uniqueness, a profile name must begin with a vendor identifier code (see Clause 23) in base-10 integer format, followed by a dash. All subsequent characters are administered by the organization registered with that vendor identifier code. The vendor identifier code that prefixes the profile name shall indicate the organization that publishes and maintains the profile document named by the remainder of the profile name. This vendor identifier need not have any relationship to the vendor identifier of the device within which the object resides.

A profile defines a set of additional properties, behavior, and/or requirements for this object beyond those specified here. This standard defines only the format of the names of profiles. The definition of the profiles themselves is outside the scope of this standard.

12.10 Command Object Type

The Command object type defines a standardized object whose properties represent the externally visible characteristics of a multi-action command procedure. A Command object is used to write a set of values to a group of object properties, based on the "action code" that is written to the Present_Value of the Command object. Whenever the Present_Value property of the Command object is written to, it triggers the Command object to take a set of actions that change the values of a set of other objects' properties.

The Command object would typically be used to represent a complex context involving multiple variables. The Command object is particularly useful for representing contexts that have multiple states. For example, a particular zone of a building might have three states: UNOCCUPIED, WARMUP, and OCCUPIED. To establish the operating context for each state, numerous objects' properties may need to be changed to a collection of known values. For example, when unoccupied, the temperature setpoint might be 65°F and the lights might be off. When occupied, the setpoint might be 72°F and the lights turned on, etc.

The Command object defines the relationship between a given state and those values that shall be written to a collection of different objects' properties to realize that state. Normally, a Command object is passive. Its In_Process property is FALSE, indicating that the Command object is waiting for its Present_Value property to be written with a value. When Present_Value is written, the Command object shall begin a sequence of actions. The In_Process property shall be set to TRUE, indicating that the Command object has begun processing one of a set of action sequences that is selected based on the particular value written to the Present_Value property. If an attempt is made to write to the Present_Value property through WriteProperty services while In_Process is TRUE, then a Result(-) shall be issued rejecting the write.

The new value of the Present_Value property determines which sequence of actions the Command object shall take. These actions are specified in an array of action lists indexed by this value. The Action property contains these lists. A given list may be empty, in which case no action takes place, except that In_Process is returned to FALSE and All_Writes_Successful is set to TRUE. If the list is not empty, then for each action in the list the Command object shall write a particular value to a particular property of a particular object in a particular BACnet Device. Note, however, that the capability to write to remote devices is not required.

Note also that the Command object does not guarantee that every write will be successful, and no attempt is made by the Command object to "roll back" successfully written properties to their previous values in the event that one or more writes fail. If any of the writes fail, then the All_Writes_Successful property is set to FALSE and the Write_Successful flag for that BACnetActionCommand is set to FALSE. If the Quit_On_Failure flag is TRUE for the failed BACnetActionCommand, then all subsequent BACnetActionCommands in the list shall have their Write_Successful flag set to FALSE. If an individual write succeeds, then the Write_Successful flag for that BACnetActionCommand shall be set to TRUE. If all the writes are successful, then the All_Writes_Successful property is set to TRUE. Once all the writes have been processed to completion by the Command object, the In_Process property is set back to FALSE and the Command object becomes passive again, waiting for another command.

It is important to note that the particular value that is written to the Present_Value property is not what triggers the action, but the act of writing itself. Thus if the Present_Value property has the value 5 and it is again written with the value 5, then the 5th list of actions will be performed again. Writing zero to the Present_Value causes no action to be taken and is the same as invoking an empty list of actions.

The Command object is a powerful concept with many beneficial applications. However, there are unique aspects of the Command object that can cause confusing or destructive side effects if the Command object is improperly configured. Since the Command object can manipulate other objects' properties, it is possible that a Command object could be configured to command itself. In such a case, the In_Process property acts as an interlock and protects the Command object from self-oscillation. However, it is also possible for a Command object to command another Command object that commands the first Command object and so on. The possibility exists for Command objects that command GROUP objects. In these cases of "circular referencing," it is possible for confusing side effects to occur. When references occur to objects in other BACnet Devices, there is an increased possibility of time delays, which could cause oscillatory behavior between Command objects that are improperly configured in such a circular manner. Caution should be exercised when configuring Command objects that reference objects outside the BACnet device that contains them.

The Command object and its properties are summarized in Table 12-12 and described in detail in this subclause.

Table 12-12. Properties of the Command Object Type

Property Identifier	Property Datatype	Conformance Code
Object_Identifier	BACnetObjectIdentifier	R
Object_Name	CharacterString	R
Object_Type	BACnetObjectType	R
Description	CharacterString	O
Present_Value	Unsigned	W
In_Process	BOOLEAN	R
All_Writes_Successful	BOOLEAN	R
Action	BACnetARRAY[N] of BACnetActionList	R
Action_Text	BACnetARRAY[N] of CharacterString	O
Profile_Name	CharacterString	O

12.10.1 Object_Identifier

This property, of type BACnetObjectIdentifier, is a numeric code that is used to identify the object. It shall be unique within the BACnet Device that maintains it.

12.10.2 Object_Name

This property, of type CharacterString, shall represent a name for the object that is unique within the BACnet Device that maintains it. The minimum length of the string shall be one character. The set of characters used in the Object_Name shall be restricted to printable characters.

12.10.3 Object_Type

This property, of type BACnetObjectType, indicates membership in a particular object type class. The value of this property shall be COMMAND.

12.10.4 Description

This property, of type CharacterString, is a string of printable characters whose content is not restricted.

12.10.5 Present_Value

This property, of type Unsigned, indicates which action the Command object is to take or has already taken. Whenever the Present_Value property is written to, it triggers the Command object to take a set of actions that change the values of a set of other objects' properties.

The Present_Value may be written to with any value from 0 to the maximum number of actions supported by the Action property. When the Present_Value is written to, the Command object begins a sequence of actions. The new value of the Present_Value property determines which list of actions the Command object shall take. These actions are specified in the Action property, which is an array of lists of actions to take. The array is indexed by the value being written. A given list may be empty, in which case no action takes place. If the list is not empty, then for each action in the list, the Command object shall write a particular value to a particular property of a particular object in a particular BACnet Device.

12.10.6 In_Process

This property, of type BOOLEAN, shall be set to TRUE when a value is written to the Present_Value property. This TRUE value indicates that the Command object has begun processing one of a set of action sequences. Once all of the writes have been attempted by the Command object, the In_Process property shall be set back to FALSE.

12.10.7 All_Writes_Successful

This property, of type BOOLEAN, indicates the success or failure of the sequence of actions that are triggered when the Present_Value property is written to. At that time, In_Process is set to TRUE and All_Writes_Successful is set to FALSE. If after the list has been executed, all of the writes have succeeded, then All_Writes_Successful is set to TRUE at the same time that In_Process is set to FALSE. Therefore, while In_Process is TRUE, the value of All_Writes_Successful is not a valid indication of the current or previous operation.

12.10.8 Action

This property, of type BACnetARRAY of BACnetActionList, specifies an array of "action lists." These action lists are indexed by the value that is written to the Present_Value property. A given list may be empty, in which case no action takes place, except that In_Process is returned to FALSE and All_Writes_Successful is set to TRUE. If the list is not empty, then for each action in the list, the Command object shall write a particular value to a particular property of a particular object in a particular BACnet Device based on the specifications in each BACnetActionCommand. Each write shall occur in the order that BACnetActionCommand list elements would appear if the list was read using the ReadProperty service. The value zero is a special case that takes no action and behaves like an empty list. The number of defined action lists may be found by reading the Action property with an array index of zero. If the size of this array is changed, the size of the Action_Text array, if present, shall also be changed to the same size.

Each BACnetActionCommand is a specification of a single value to be written to a single property of a single object. BACnetActionCommands have nine parts: an optional BACnet device identifier, an object identifier, a property identifier, a conditional property array index, a value to be written, a conditional priority, an optional post-writing delay time, a premature quit flag, and a write success flag. The components and their datatypes are shown below.

Component	Datatype
Device_Identifier	BACnetObjectIdentifier (Optional)
Object_Identifier	BACnetObjectIdentifier
Property_Identifier	BACnetPropertyIdentifier
Property_Array_Index	Unsigned (Conditional)
Property_Value	Any
Priority	Unsigned (1..16) (Conditional)
Post_Delay	Unsigned (Optional)
Quit_On_Failure	BOOLEAN
Write_Successful	BOOLEAN

If the Device_Identifier is not present, then the write shall be performed on objects residing in the device that contains the Command object. A device that supports the Command object type is not required to support writing outside the device. If the Property_Identifier refers to an array property, then the Property_Array_Index shall also be present to specify the index within the array of the property to be written. If the property being written is a commandable property, then a priority value shall be supplied; otherwise it shall be omitted. If the Quit_On_Failure flag is TRUE, then if the write fails for any reason, the device shall terminate the execution of the action list prematurely. Otherwise, writing shall continue after each failure with the next element of the action list. In either case, All_Writes_Successful shall remain FALSE throughout the execution of the list. After each write, whether successful or not, if the Post_Delay is present, it shall represent a delay in seconds prior to the execution of the next write or the completion of all writing and the setting of In_Process to FALSE.

If the write fails for any reason, then the Write_Successful flag shall be set to FALSE. If the Quit_On_Failure flag is TRUE, then the first write that fails shall also terminate the execution list prematurely. In this case, the Write_Successful flag in subsequent entries in the same list shall be set to FALSE. If the write succeeds, then the Write_Successful flag shall be set to TRUE.

12.10.9 Action_Text

This property, of type BACnetARRAY of CharacterString, shall be used to indicate a text string description for each of the possible values of the Present_Value property. The content of these strings is not restricted. If the size of this array is changed, the size of the Action array shall also be changed to the same size.

12.10.10 Profile_Name

This optional property, of type CharacterString, is the name of an object profile to which this object conforms. To ensure uniqueness, a profile name must begin with a vendor identifier code (see Clause 23) in base-10 integer format, followed by a dash. All subsequent characters are administered by the organization registered with that vendor identifier code. The vendor identifier code that prefixes the profile name shall indicate the organization that publishes and maintains the profile document named by the remainder of the profile name. This vendor identifier need not have any relationship to the vendor identifier of the device within which the object resides.

A profile defines a set of additional properties, behavior, and/or requirements for this object beyond those specified here. This standard defines only the format of the names of profiles. The definition of the profiles themselves is outside the scope of this standard.

12.11 Device Object Type

The Device object type defines a standardized object whose properties represent the externally visible characteristics of a BACnet Device. There shall be exactly one Device object in each BACnet Device. A Device object is referenced by its Object_Identifier property, which is not only unique to the BACnet Device that maintains this object but is also unique throughout the BACnet internetwork. The Device object type and its properties are summarized in Table 12-13 and described in detail in this subclause.

Table 12-13. Properties of the Device Object Type

Property Identifier	Property Datatype	Conformance Code
Object_Identifier	BACnetObjectIdentifier	R
Object_Name	CharacterString	R
Object_Type	BACnetObjectType	R
System_Status	BACnetDeviceStatus	R
Vendor_Name	CharacterString	R
Vendor_Identifier	Unsigned16	R
Model_Name	CharacterString	R
Firmware_Revision	CharacterString	R
Application_Software_Version	CharacterString	R
Location	CharacterString	O
Description	CharacterString	O
Protocol_Version	Unsigned	R
Protocol_Revision	Unsigned	R
Protocol_Services_Supported	BACnetServicesSupported	R
Protocol_Object_Types_Supported	BACnetObjectTypesSupported	R
Object_List	BACnetARRAY[N]of BACnetObjectIdentifier	R
Max_APDU_Length_Accepted	Unsigned	R
Segmentation_Supported	BACnetSegmentation	R
Max_Segments_Accepted	Unsigned	O[1]
VT_Classes_Supported	List of BACnetVTClass	O[2]
Active_VT_Sessions	List of BACnetVTSession	O[2]
Local_Time	Time	O[3,4]
Local_Date	Date	O[3,4]
UTC_Offset	INTEGER	O[4]
Daylight_Savings_Status	BOOLEAN	O[4]
APDU_Segment_Timeout	Unsigned	O[1]
APDU_Timeout	Unsigned	R
Number_Of_APDU_Retries	Unsigned	R
List_Of_Session_Keys	List of BACnetSessionKey	O
Time_Synchronization_Recipients	List of BACnetRecipient	O[5]
Max_Master	Unsigned(1..127)	O[6]
Max_Info_Frames	Unsigned	O[6]
Device_Address_Binding	List of BACnetAddressBinding	R
Database_Revision	Unsigned	R
Configuration_Files	BACnetARRAY[N] of BACnetObjectIdentifier	O[7]
Last_Restore_Time	BACnetTimeStamp	O[7]
Backup_Failure_Timeout	Unsigned16	O[8]
Active_COV_Subscriptions	List of BACnetCOVSubscription	O[9]
Slave_Proxy_Enable	BACnetArray[N] of BOOLEAN	O[10]
Manual_Slave_Address_Binding	List of BACnetAddressBinding	O[10]
Auto_Slave_Discovery	BACnetArray[N] of BOOLEAN	O[11]
Slave_Address_Binding	List of BACnetAddressBinding	O[12]
Profile_Name	CharacterString	O

[1] Required if segmentation of any kind is supported.
[2] If one of the properties VT_Classes_Supported or Active_VT_Sessions is present, then both of these properties shall be present. Both properties are required if support for VT Services is indicated in the PICS.

[3] If the device supports the execution of the TimeSynchronization service, then these properties shall be present.
[4] If the device supports the execution of the UTCTimeSynchronization service, then these properties shall be present.
[5] Required if PICS indicates that this device is a Time Master. If present, this property shall be writable.
[6] These properties are required if the device is an MS/TP master node.
[7] These properties are required if the device supports the backup and restore procedures.
[8] This property must be present and writable if the device supports the backup and restore procedures.
[9] This property is required if the device supports execution of either the SubscribeCOV or SubscribeCOVProperty service.
[10] This property shall be present and writable if the device is capable of being a Slave-Proxy device.
[11] This property shall be present if the device is capable of being a Slave-Proxy device that implements automatic discovery of slaves.
[12] This property shall be present if the device is capable of being a Slave-Proxy device.

12.11.1 Object_Identifier

This property, of type BACnetObjectIdentifier, is a numeric code that is used to identify the object. For the Device object, the object identifier shall be unique internetwork-wide.

12.11.2 Object_Name

This property, of type CharacterString, shall represent a name for the object that is unique internetwork-wide. The minimum length of the string shall be one character. The set of characters used in the Object_Name shall be restricted to printable characters.

12.11.3 Object_Type

This property, of type BACnetObjectType, indicates membership in a particular object type class. The value of this property shall be DEVICE.

12.11.4 System_Status

This property, of type BACnetDeviceStatus, reflects the current physical and logical status of the BACnet Device. The values that may be taken on by this property are

{OPERATIONAL,OPERATIONAL_READ_ONLY,DOWNLOAD_REQUIRED, DOWNLOAD_IN_PROGRESS, NON_OPERATIONAL, BACKUP_IN_PROGRESS}.

The exact meaning of these states, except for BACKUP_IN_PROGRESS, in a given device and their synchronization with other internal operations of the device or the execution of BACnet services by the device are local matters and are not defined by this standard.

12.11.5 Vendor_Name

This property, of type CharacterString, identifies the manufacturer of the BACnet Device.

12.11.6 Vendor_Identifier

This property, of type Unsigned16, is a unique vendor identification code, assigned by ASHRAE, which is used to distinguish proprietary extensions to the protocol. See Clause 23.

12.11.7 Model_Name

This property, of type CharacterString, is assigned by the vendor to represent the model of the BACnet Device.

12.11.8 Firmware_Revision

This property, of type CharacterString, is assigned by the vendor to represent the level of firmware installed in the BACnet Device.

12.11.9 Application_Software_Version

This property, of type CharacterString, identifies the version of application software installed in the machine. The content of this string is a local matter, but it could be a date-and-time stamp, a programmer's name, a host file version number, etc.

12.11.10 Location

This property, of type CharacterString, indicates the physical location of the BACnet Device.

12.11.11 Description

This property, of type CharacterString, is a string of printable characters that may be used to describe the application being carried out by the BACnet Device or other locally desired descriptive information.

12.11.12 Protocol_Version

This property, of type Unsigned, represents the version of the BACnet protocol supported by this BACnet Device. Every major revision of BACnet shall increase this version number by one. The initial release of BACnet shall be version 1.

12.11.13 Protocol_Revision

This property, of type Unsigned, shall indicate the minor revision level of the BACnet standard. This value shall start at 1 and be incremented for any substantive change(s) to the BACnet standard that affect device communication or behavior. This value shall revert to zero upon each change to the Protocol_Version property. Changes to the values for Protocol_Version and Protocol_Revision are recorded in the History of Revisions at the end of this standard.

This property is required for all devices implementing BACnet Protocol_Version 1, Protocol_Revision 1 and above. Absence of this property shall indicate a device implemented to a version of the standard prior to the definition of the Protocol_Revision property.

12.11.14 Protocol_Services_Supported

This property, of type BACnetServicesSupported, indicates which standardized protocol services are supported by this device's protocol implementation.

12.11.15 Protocol_Object_Types_Supported

This property, of type BACnetObjectTypesSupported, indicates which standardized object types are supported by this device's protocol implementation. The list of properties supported for a particular object may be acquired by use of the ReadPropertyMultiple service with a property reference of ALL (see 15.7.3.1.2).

12.11.16 Object_List

This read only property is a BACnetARRAY of Object_Identifiers, one Object_Identifier for each object within the device that is accessible through BACnet services.

12.11.17 Max_APDU_Length_Accepted

This property, of type Unsigned, is the maximum number of octets that may be contained in a single, indivisible application layer protocol data unit. The value of this property shall be greater than or equal to 50. The value of this property is also constrained by the underlying data link technology. See Clauses 6 through 11.

12.11.18 Segmentation_Supported

This property, of type BACnetSegmentation, indicates whether the BACnet Device supports segmentation of messages and, if so, whether it supports segmented transmission, reception, or both:

{SEGMENTED_BOTH, SEGMENTED_TRANSMIT, SEGMENTED_RECEIVE, NO_SEGMENTATION}.

12.11.19 Max_Segments_Accepted

The Max_Segments_Accepted property, of type Unsigned, shall indicate the maximum number of segments of an APDU that this device will accept.

12.11.20 VT_Classes_Supported

The VT_Classes_Supported property is a List of BACnetVTClass each of which is an enumeration indicating a particular set of terminal characteristics. A given BACnet Device may support multiple types of behaviors for differing types of terminals or differing types of operator interface programs. At a minimum, such devices shall support the "Default-terminal" VT-class defined in 17.5.

If one of the properties VT_Classes_Supported or Active_VT_Sessions is present, then both of these properties shall be present. Both properties are required if support for VT Services is indicated in the PICS.

12.11.21 Active_VT_Sessions

The Active_VT_Sessions property is a List of BACnetVTSession each of which consists of a Local VT Session Identifier, a Remote VT Session Identifier, and Remote VT Address. This property provides a network-visible indication of those virtual terminal sessions (VT-Sessions) that are active at any given time. Whenever a virtual terminal session is created with the VT-Open service, a new entry is added to the Active_VT_Sessions list. Similarly, whenever a VT-session is terminated, the corresponding entry shall be removed from the Active_VT_Sessions list.

If one of the properties VT_Classes_Supported or Active_VT_Sessions is present, then both of these properties shall be present. Both properties are required if support for VT Services is indicated in the PICS.

12.11.22 Local_Time

The Local_Time property, of type Time, shall indicate the time of day to the best of the device's knowledge. If the BACnet Device does not have any knowledge of time or date, then the Local_Time property shall be omitted.

12.11.23 Local_Date

The Local_Date property, of type Date, shall indicate the date to the best of the device's knowledge. If the BACnet Device does not have any knowledge of time or date, then the Local_Date property shall be omitted.

12.11.24 UTC_Offset

The UTC_Offset property, of type INTEGER, shall indicate the number of minutes (-780 to +780) offset between local standard time and Universal Time Coordinated. The time zones to the west of the zero degree meridian shall be positive values, and those to the east shall be negative values. The value of the UTC_Offset property is subtracted from the UTC received in UTCTimeSynchronization service requests to calculate the correct local standard time.

12.11.25 Daylight_Savings_Status

The Daylight_Savings_Status property, of type BOOLEAN, shall indicate whether daylight savings time is in effect (TRUE) or not (FALSE) at the BACnet Device's location.

12.11.26 APDU_Segment_Timeout

The APDU_Segment_Timeout property, of type Unsigned, shall indicate the amount of time in milliseconds between retransmission of an APDU segment. The default value for this property shall be 2000 milliseconds. This value shall be non-zero if the Device object property called Number_Of_APDU_Retries is non-zero. See 5.3. If segmentation of any kind is supported, then the APDU_Segment_Timeout property shall be present.

In order to achieve reliable communication, it is recommended that the values of the APDU_Segment_Timeout properties of the Device objects of all intercommunicating devices should contain the same value.

12.11.27 APDU_Timeout

The APDU_Timeout property, of type Unsigned, shall indicate the amount of time in milliseconds between retransmissions of an APDU requiring acknowledgment for which no acknowledgment has been received. The default value for this property shall be 3,000 milliseconds for devices that permit modification of this parameter. Otherwise, the default value shall be 60,000 milliseconds. This value shall be non-zero if the Device object property called Number_Of_APDU_Retries is non-zero. See 5.3.

In order to achieve reliable communication, it is recommended that the values of the APDU_Timeout properties of the Device objects of all intercommunicating devices should contain the same value.

12.11.28 Number_Of_APDU_Retries

The Number_Of_APDU_Retries property, of type Unsigned, shall indicate the maximum number of times that an APDU shall be retransmitted. The default value for this property shall be 3. If this device does not perform retries, then this property shall be set to zero. If the value of this property is greater than zero, a non-zero value shall be placed in the Device object APDU_Timeout property. See 5.3.

12.11.29 List_Of_Session_Keys

This property is a List of BACnetSessionKey each of which is one of the cryptographic keys used to communicate with other security-conscious BACnet Devices. This property shall not be readable or writable by any device except a device designated the "Key Server." A session key shall consist of a 56-bit encryption key and a BACnet Address of the peer with which secure communications is requested.

12.11.30 Time_Synchronization_Recipients

The Time_Synchronization_Recipients property is used to control the restrictions placed on a device's use of the TimeSynchronization service. The value of this property shall be a list of zero or more BACnetRecipients. If the list is of length zero, a device is prohibited from automatically sending a TimeSynchronization request. If the list is of length one or more, a device may automatically send a TimeSynchronization request but only to the devices or addresses listed. If it is present, this property shall be writable. If the PICS indicates that this device is a Time Master, then the Time_Synchronization_Recipients property shall be present.

12.11.31 Max_Master

The Max_Master property, of type Unsigned, shall be present if the device is a master node on an MS/TP network. The value of Max_Master specifies the highest possible address for master nodes and shall be less than or equal to 127. If the Max_Master property is not writeable via BACnet services, its value shall be 127. See 9.5.3.

12.11.32 Max_Info_Frames

The Max_Info_Frames property, of type Unsigned, shall be present if the device is a node on an MS/TP network. The value of Max_Info_Frames specifies the maximum number of information frames the node may send before it must pass the token. If Max_Info_Frames is not writable or otherwise user configurable, its value shall be one. See 9.5.3.

12.11.33 Device_Address_Binding

The Device_Address_Binding property is a List of BACnetAddressBinding each of which consists of a BACnet Object_Identifier of a BACnet Device object and a BACnet device address in the form of a BACnetAddress. Entries in the list identify the actual device addresses that will be used when the remote device must be accessed via a BACnet service request. A value of zero shall be used for the network-number portion of BACnetAddress entries for other devices residing on the same network as this device. The list may be empty if no device identifier-device address bindings are currently known to the device.

12.11.34 Database_Revision

This property, of type Unsigned, is a logical revision number for the device's database. It is incremented when an object is created, an object is deleted, an object's name is changed, an object's Object_Identifier property is changed, or a restore is performed.

12.11.35 Configuration_Files

This optional property is a BACnet Array of BACnetObjectIdentifier. Entries in the array identify the files within the device that define the device's image that can be backed up. The contents of this property is only required to be valid during the backup procedure. This property must be supported if the device supports the BACnet backup and restore procedure as described in 19.1.

12.11.36 Last_Restore_Time

This optional property, of type BACnetTimeStamp, is the time at which the device's image was last restored as described in 19.1. This property must be supported if the device supports the BACnet backup and restore procedures as described in 19.1.

12.11.37 Backup_Failure_Timeout

This optional property, of type Unsigned16, is the time, in seconds, that the device being backed up or restored must wait before unilaterally ending the backup or restore procedure. This property must be writable with the intent that the device performing the backup, or the human operator, will configure this with an appropriate timeout.

12.11.38 Active_COV_Subscriptions

The Active_COV_Subscriptions property is a List of BACnetCOVSubscription, each of which consists of a Recipient, a Monitored Property Reference, an Issue Confirmed Notifications flag, a Time Remaining value and an optional COV

Increment. This property provides a network-visible indication of those COV subscriptions that are active at any given time. Whenever a COV Subscription is created with the SubscribeCOV or SubscribeCOVProperty service, a new entry is added to the Active_COV_Subscriptions list. Similarly, whenever a COV Subscription is terminated, the corresponding entry shall be removed from the Active_COV_Subscriptions list.

This property is required if the device supports execution of either SubscribeCOV or SubscribeCOVProperty service.

12.11.39 Slave_Proxy_Enable

This property, of type BACnetArray of BOOLEAN, is an indication whether (TRUE) or not (FALSE) the device will perform Slave-Proxy functions for each of the MS/TP ports represented by each array element. This property shall be present if the device is capable of performing the functions of a Slave-Proxy device. The value of this property shall be retained over a device reset.

12.11.40 Manual_Slave_Address_Binding

This property, of type List of BACnetAddressBinding, describes the manually configured set of slave devices for which this device is acting as a Slave Proxy as described in 16.10.2. This property shall be present if the device is capable of performing the functions of a Slave-Proxy device. If present, and the device is directly attached to an MS/TP network, then this property shall be writable.

This property is used to manually configure a set of slave devices for which this device will be a proxy. This property allows a Slave Proxy that does not support automatic slave discovery be configured with a set of slaves for which this device will be a proxy. It also allows a Slave-Proxy device to be a proxy for Slave devices that do not support the special object instance of 4194303 as described in Clause 12. The value of this property shall be retained over a device reset. When enabled, the Slave-Proxy device shall periodically check each device that is in this list, and not in the Slave_Address_Binding list, by reading the device's Protocol_Services_Supported property from the device's Device object using the ReadProperty service. If the device responds and indicates that it does not execute the Who-Is service, it shall be added to the Slave_Address_Binding property. The period at which the devices are checked is a local matter.

12.11.41 Auto_Slave_Discovery

This property, of type BACnetArray of BOOLEAN, is an indication whether (TRUE) or not (FALSE) the device will perform automatic slave detection functions for each of the MS/TP ports represented by each array element. This property shall be present if the device is capable of performing the functions of a Slave-Proxy device. The value of this property shall be retained over a device reset.

Slave detection shall be accomplished by the proxy device using ReadProperty services to read, at a minimum, the Device object's Protocol_Services_Supported property for each MAC address on each port where Auto_Slave_Discovery for that port is TRUE. The ReadProperty service shall use the special object instance of 4194303 as described in Clause 12. If the device is found to support execution of the Who-Is service, it is ignored; otherwise, the device shall be added to the Slave_Address_Binding property. The slave detection algorithm shall be repeated periodically. The period at which it is repeated is a local matter.

12.11.42 Slave_Address_Binding

This property, of type List of BACnetAddressBinding, describes the set of slave devices for which this device is acting as a Slave-Proxy as described in 16.10.2. This property shall be present if the device is capable of performing the functions of a Slave-Proxy device. If present, and the device is directly attached to an MS/TP network, then this property shall be writable.

The set of devices described by the Slave_Address_Binding property consists of those devices described in the Manual_Slave_Address_Binding and those devices that are automatically discovered. When enabled, the Slave-Proxy device shall periodically check each device in this list by reading the device's Protocol_Services_Supported property from the device's Device object using the ReadProperty service. If the device fails to respond, or indicates that it executes Who-Is, it shall be removed from the list. The period at which the devices are checked is a local matter.

12.11.43 Profile_Name

This optional property, of type CharacterString, is the name of an object profile to which this object conforms. To ensure uniqueness, a profile name must begin with a vendor identifier code (see Clause 23) in base-10 integer format, followed by a dash. All subsequent characters are administered by the organization registered with that vendor identifier code. The vendor

identifier code that prefixes the profile name shall indicate the organization that publishes and maintains the profile document named by the remainder of the profile name. This vendor identifier need not have any relationship to the vendor identifier of the device within which the object resides.

A profile defines a set of additional properties, behavior, and/or requirements for this object beyond those specified here. This standard defines only the format of the names of profiles. The definition of the profiles themselves is outside the scope of this standard.

12.12 Event Enrollment Object Type

The Event Enrollment object type defines a standardized object that represents and contains the information required for managing events within BACnet systems. "Events" are changes of value of any property of any object that meet certain predetermined criteria. The primary purpose for Event Enrollment objects is to define an event and to provide a connection between the occurrence of an event and the transmission of a notification message to one or more recipients.

The Event Enrollment object contains the event-type description, the parameters needed to determine if the event has occurred, and a device to be notified. Alternatively, a Notification Class object may serve to identify the recipients of event notifications. A device is considered to be "enrolled for event notification" if it is the recipient to be notified or one of the recipients in a Notification Class object referenced by the Event Enrollment object.

Clause 13 describes the interaction between Event Enrollment objects and the Alarm and Event application services. The Event Enrollment object and its properties are summarized in Table 12-14 and described in detail in this subclause.

Table 12-14. Properties of the Event Enrollment Object Type

Property Identifier	Property Datatype	Conformance Code
Object_Identifier	BACnetObjectIdentifier	R
Object_Name	CharacterString	R
Object_Type	BACnetObjectType	R
Description	CharacterString	O
Event_Type	BACnetEventType	R
Notify_Type	BACnetNotifyType	R
Event_Parameters	BACnetEventParameter	R
Object_Property_Reference	BACnetDeviceObjectPropertyReference	R
Event_State	BACnetEventState	R
Event_Enable	BACnetEventTransitionBits	R
Acked_Transitions	BACnetEventTransitionBits	R
Notification_Class	Unsigned	R
Event_Time_Stamps	BACnetARRAY[3] of BACnetTimeStamp	R
Profile_Name	CharacterString	O

12.12.1 Object_Identifier

This property, of type BACnetObjectIdentifier, is a numeric code that is used to identify the object. It shall be unique within the device that maintains it.

12.12.2 Object_Name

This property, of type CharacterString, shall represent a name for the object that is unique within the BACnet Device that maintains it. The minimum length of the string shall be one character. The set of characters used in the Object_Name shall be restricted to printable characters.

12.12.3 Object_Type

This property, of type BACnetObjectType, indicates membership in a particular object type class. The value of this property shall be EVENT_ENROLLMENT.

12.12.4 Description

This property, of type CharacterString, is a string of printable characters whose content is not restricted.

12.12.5 Event_Type

This read only property, of type BACnetEventType, indicates the type of event algorithm that is to be used to detect the occurrence of events and report to enrolled devices. This parameter is an enumerated type that may have any of the following values:

{CHANGE_OF_BITSTRING, CHANGE_OF_STATE, CHANGE_OF_VALUE, COMMAND_FAILURE, FLOATING_LIMIT, OUT_OF_RANGE, BUFFER_READY, CHANGE_OF_LIFE_SAFETY, EXTENDED}.

There is a specific relationship between each event algorithm, the parameter list, and the event types that are valid for the event. The Event_Type reflects the algorithm that is used to determine the state of an event. The algorithm for each Event_Type is specified in Clause 13. The Event_Parameters property provides the parameters needed by the algorithm.

The valid combinations of Event_Type, Event_State, and Event_Parameters values are summarized in Table 12-15.

Table 12-15. Event_Types, Event_States, and their Parameters

Event_Type	Event_State	Event_Parameters
CHANGE_OF_BITSTRING	NORMAL OFFNORMAL	Time_Delay Bitmask List_Of_Bitstring_Values
CHANGE_OF_STATE	NORMAL OFFNORMAL	Time_Delay List_Of_Values
CHANGE_OF_VALUE	NORMAL OFFNORMAL	Time_Delay Bitmask Referenced_Property_Increment
COMMAND_FAILURE	NORMAL OFFNORMAL	Time_Delay Feedback_Property_Reference
FLOATING_LIMIT	NORMAL HIGH_LIMIT LOW_LIMIT	Time_Delay Setpoint_Reference Low_Diff_Limit High_Diff_Limit Deadband
OUT_OF_RANGE	NORMAL HIGH_LIMIT LOW_LIMIT	Time_Delay Low_Limit High_Limit Deadband
BUFFER_READY	NORMAL	Notification_Threshold
CHANGE_OF_LIFE_SAFETY	NORMAL OFFNORMAL LIFE_SAFETY_ALARM	Time_Delay List_Of_Alarm_Values List_Of_Life_Safety_Alarm_Values Mode_Property_Reference

12.12.6 Notify_Type

This property, of type BACnetNotifyType, shall convey whether the notifications generated by the monitoring algorithm specified by the Event_Type property should be Events or Alarms.

12.12.7 Event_Parameters

The Event_Parameters property, of type BACnetEventParameter, determines the algorithm used to monitor the referenced object and provides the parameter values needed for this algorithm. The meaning of each value in the Event_Parameters, depends on the algorithm as indicated by the Event_Type column in Table 12-15. Each of the possible parameters is described below.

Bitmask — This parameter, of type BIT STRING, applies to the CHANGE_OF_BITSTRING event algorithm and the CHANGE_OF_VALUE event algorithm in the special case where the referenced property is a BIT STRING datatype. It represents a bitmask that is used to indicate which bits in the referenced property are to be monitored by the algorithm. A value of one in a bit position indicates that the bit in this position in the referenced property is to be monitored by the algorithm. A value of zero in a bit position indicates that the bit in this position in the referenced property is not significant for the purpose of detecting this CHANGE_OF_BITSTRING or CHANGE_OF_VALUE.

List_Of_Bitstring_Values	This parameter is a list of bitstrings that apply to the CHANGE_OF_BITSTRING event algorithm. This list of bitstrings defines the set of states for which the referenced property is OFFNORMAL. Only the bits indicated by the Bitmask are significant. If the value of the referenced property changes to one of the values in the List_Of_Bitstring_Values, then the Event_State property of the Event Enrollment object makes a transition TO-OFFNORMAL and appropriate notifications are sent.
List_Of_Values	This parameter is a list of BACnetPropertyStates that apply to the CHANGE_OF_STATE event algorithm. This event algorithm applies to referenced properties that have discrete or enumerated values. The List_Of_Values is a subset of the possible values that the property may have. If the value of the referenced property changes to one of the values in the List_Of_Values, then the Event_State property of the Event Enrollment object makes a transition TO-OFFNORMAL and appropriate notifications are sent.
Referenced_Property_Increment	This parameter, of type REAL, applies to the CHANGE_OF_VALUE event algorithm. It represents the increment by which the referenced property must change in order for the event to occur.
Time_Delay	This parameter, of type Unsigned, applies to all event types and represents the time, in seconds, that the conditions monitored by the event algorithm must persist before an event notification is issued.
Feedback_Property_Reference	This parameter, of type BACnetObjectPropertyReference, applies to the COMMAND_FAILURE algorithm. It identifies the object and property that provides the feedback to ensure that the commanded property has changed value. This property may reference only object properties that have enumerated values or are of type BOOLEAN.
Setpoint_Reference	This parameter, of type BACnetObjectPropertyReference, applies to the FLOATING_LIMIT event algorithm. It indicates the setpoint reference for the reference property interval.
Deadband, High_Diff_Limit, Low_Diff_Limit, High_Limit, Low_Limit	These parameters, of type REAL, apply to the FLOATING_LIMIT and OUT_OF_RANGE event algorithms. Their use is described in the algorithms for these types in Clause 13.
Notification_Threshold	This parameter, of type Unsigned, applies to the BUFFER_READY algorithm. It specifies the value of Records_Since_Notification at which notification occurs.
List_Of_Life_Safety_Alarm_Values	This parameter is a list of BACnetLifeSafetyState that applies to the CHANGE_OF_LIFE_SAFETY algorithm. If the value of the referenced property changes to one of the values in the List_Of_Life_Safety_Alarm_Values, then the Event_State property of the Event Enrollment object makes a transition TO-OFFNORMAL and appropriate notifications are sent. The resulting event state is LIFE_SAFETY_ALARM
List_Of_Alarm_Values	This parameter is a list of BACnetLifeSafetyState that applies to the CHANGE_OF_LIFE_SAFETY algorithm. If the value of the referenced property changes to one of the values in the List_Of_Alarm_Values, then the Event_State property of the Event Enrollment object makes a transition TO-OFFNORMAL and appropriate notifications are sent. The resulting event state is OFFNORMAL.

Mode_Property_Reference	This parameter, of type BACnetDeviceObjectPropertyReference, applies to the CHANGE_OF_LIFE_SAFETY algorithm. It identifies the object and property that provides the operating mode of the referenced object providing life safety functionality(normally the Mode property). This parameter may reference only object properties that are of type BACnetLifeSafetyMode.

12.12.8 Object_Property_Reference

This property, of type BACnetDeviceObjectPropertyReference, designates the particular object and property referenced by this Event Enrollment object. The algorithm specified by the Event_Type property is applied to the referenced property in order to determine the Event_State of the event.

If this property is writable, it may be restricted to only support references to objects inside of the device containing the Event Enrollment object. If the property is restricted to referencing objects within the containing device, an attempt to write a reference to an object outside the containing device into this property shall cause a Result(-) to be returned with an error class of PROPERTY and an error code of OPTIONAL_FUNCTIONALITY_NOT_SUPPORTED.

If this property is set to reference an object outside the device containing the Event Enrollment object, the method used for acquisition of the referenced property value for the purpose of monitoring is a local matter. The only restriction on the method of data acquisition is that the monitoring device be capable of using ReadProperty for this purpose so as to be interoperable with all BACnet devices.

12.12.9 Event_State

This property, of type BACnetEventState, contains the current state of the event. The permitted values for Event_State are specific to the Event_Type. See Table 12-15. The value of the Event_State property is independent of the Event_Enable flags. See 12.12.10.

12.12.10 Event_Enable

This property, of type BACnetEventTransitionBits, conveys three flags that determine whether notifications are enabled for TO-OFFNORMAL, TO-FAULT, and TO-NORMAL transitions. When a flag is set, this means that the corresponding transition would cause notification to be sent to all enrolled devices. When a flag is cleared, this means that the corresponding transition would not be reported. The object's Event_State property shall be continuously updated regardless of the value of the Event_Enable property.

12.12.11 Acked_Transitions

This property, of type BACnetEventTransitionBits, shall convey three separate flags that each indicate whether the most recent TO-OFFNORMAL, TO-FAULT, or TO-NORMAL event transitions have been acknowledged, if acknowledgment is required for that transition.

12.12.12 Notification_Class

This property, of type Unsigned, implicitly references a Notification Class object in the device containing the Event Enrollment object. The class is used to specify the handling, reporting, and acknowledgment characteristics for one or more event-initiating objects.

12.12.13 Event_Time_Stamps

This property, of type BACnetARRAY[3] of BACnetTimeStamp, shall convey the times of the last event notifications for TO-OFFNORMAL, TO-FAULT, and TO-NORMAL events, respectively. Time stamps of type Time or Date shall have 'FF' in each octet and Sequence number time stamps shall have the value 0 if no event notification of that type has been generated since the object was created.

12.12.14 Profile_Name

This optional property, of type CharacterString, is the name of an object profile to which this object conforms. To ensure uniqueness, a profile name must begin with a vendor identifier code (see Clause 23) in base-10 integer format, followed by a dash. All subsequent characters are administered by the organization registered with that vendor identifier code. The vendor identifier code that prefixes the profile name shall indicate the organization that publishes and maintains the profile document

named by the remainder of the profile name. This vendor identifier need not have any relationship to the vendor identifier of the device within which the object resides.

A profile defines a set of additional properties, behavior, and/or requirements for this object beyond those specified here. This standard defines only the format of the names of profiles. The definition of the profiles themselves is outside the scope of this standard.

12.13 File Object Type

The File object type defines a standardized object that is used to describe properties of data files that may be accessed using File Services (see Clause 14). The file object type and its properties are summarized in Table 12-16 and described in detail in this subclause.

Table 12-16. Properties of the File Object Type

Property Identifier	Property Datatype	Conformance Code
Object_Identifier	BACnetObjectIdentifier	R
Object_Name	CharacterString	R
Object_Type	BACnetObjectType	R
Description	CharacterString	O
File_Type	CharacterString	R
File_Size	Unsigned	R[1]
Modification_Date	BACnetDateTime	R
Archive	BOOLEAN	W
Read_Only	BOOLEAN	R
File_Access_Method	BACnetFileAccessMethod	R
Record_Count	Unsigned	O[2]
Profile_Name	CharacterString	O

[1] If the file size can be changed by writing to the file, and File_Access_Method is STREAM_ACCESS, then this property shall be writable.
[2] This property shall be present only if File_Access_Method is RECORD_ACCESS. If the number of records can be changed by writing to the file, then this property shall be writable.

12.13.1 Object_Identifier

This property, of type BACnetObjectIdentifier, is a numeric code that is used to identify the object. It shall be unique within the BACnet Device that maintains it.

12.13.2 Object_Name

This property, of type CharacterString, shall represent a name for the object that is unique within the BACnet Device that maintains it. The minimum length of the string shall be one character. The set of characters used in the Object_Name shall be restricted to printable characters.

12.13.3 Object_Type

This property, of type BACnetObjectType, indicates membership in a particular object type class. The value of this property shall be FILE.

12.13.4 Description

This property, of type CharacterString, is a string of printable characters whose content is not restricted.

12.13.5 File_Type

This property, of type CharacterString, identifies the intended use of this file.

12.13.6 File_Size

This property, of type Unsigned, indicates the size of the file data in octets. If the size of the file can be changed by writing to the file, and File_Access_Method is STREAM_ACCESS, then this property shall be writable.

Writing to the File_Size property with a value less than the current size of the file shall truncate the file at the specified position. Writing a File_Size of 0 shall delete all of the file data but not the File object itself. Writing to the File_Size property with a value greater than the current size of the file shall expand the size of the file but the value of the new octets of the file shall be a local matter.

Devices may restrict the allowed values for writes to the File_Size. Specifically, devices may allow deletion of the file contents by writing a value of zero, but not necessarily allow arbitrary truncation or expansion.

12.13.7 Modification_Date

This property, of type BACnetDateTime, indicates the last time this object was modified. A File object shall be considered modified when it is created or written to.

12.13.8 Archive

This property, of type BOOLEAN, indicates whether the File object has been saved for historical or backup purposes. This property shall be logical TRUE only if no changes have been made to the file data by internal processes or through File Access Services since the last time the object was archived.

12.13.9 Read_Only

This property, of type BOOLEAN, indicates whether (FALSE) or not (TRUE) the file data may be changed through the use of a BACnet AtomicWriteFile service.

12.13.10 File_Access_Method

This property, of type BACnetFileAccessMethod, indicates the type(s) of file access supported for this object. The possible values for File_Access_Method are:

{RECORD_ACCESS, STREAM_ACCESS}.

12.13.11 Record_Count

This property, of type Unsigned, indicates the size of the file data in records. The Record_Count property shall be present only if File_Access_Type is RECORD_ACCESS. If the number of records can be changed by writing to the file, then this property shall be writable.

Writing to the Record_Count property with a value less than the current size of the file shall truncate the file at the specified position. Writing a Record_Count of 0 shall delete all of the file data but not the File object itself. Writing to the Record_Count property with a value greater than the current size of the file shall expand the size of the file but the value of the new octets of the file shall be a local matter.

Devices may restrict the allowed values for writes to the Record_Count. Specifically, devices may allow deletion of the file contents by writing a value of zero, but not necessarily allow arbitrary truncation or expansion.

12.13.12 Profile_Name

This optional property, of type CharacterString, is the name of an object profile to which this object conforms. To ensure uniqueness, a profile name must begin with a vendor identifier code (see Clause 23) in base-10 integer format, followed by a dash. All subsequent characters are administered by the organization registered with that vendor identifier code. The vendor identifier code that prefixes the profile name shall indicate the organization that publishes and maintains the profile document named by the remainder of the profile name. This vendor identifier need not have any relationship to the vendor identifier of the device within which the object resides.

A profile defines a set of additional properties, behavior, and/or requirements for this object beyond those specified here. This standard defines only the format of the names of profiles. The definition of the profiles themselves is outside the scope of this standard.

12.14 Group Object Type

The Group object type defines a standardized object whose properties represent a collection of other objects and one or more of their properties. A group object is used to simplify the exchange of information between BACnet Devices by providing a shorthand way to specify all members of the group at once. A group may be formed using any combination of object types. The group object and its properties are summarized in Table 12-17 and described in detail in this subclause.

Table 12-17. Properties of the Group Object Type

Property Identifier	Property Datatype	Conformance Code
Object_Identifier	BACnetObjectIdentifier	R
Object_Name	CharacterString	R
Object_Type	BACnetObjectType	R
Description	CharacterString	O
List_Of_Group_Members	List of ReadAccessSpecification	R
Present_Value	List of ReadAccessResult	R
Profile_Name	CharacterString	O

12.14.1 Object_Identifier

This property, of type BACnetObjectIdentifier, is a numeric code that is used to identify the object. It shall be unique within the BACnet Device that maintains it.

12.14.2 Object_Name

This property, of type CharacterString, shall represent a name for the object that is unique within the BACnet Device that maintains it. The minimum length of the string shall be one character. The set of characters used in the Object_Name shall be restricted to printable characters.

12.14.3 Object_Type

This property, of type BACnetObjectType, indicates membership in a particular object type class. The value of this property shall be GROUP.

12.14.4 Description

This property, of type CharacterString, is a string of printable characters whose content is not restricted.

12.14.5 List_Of_Group_Members

This property is a list of one or more read access specifications, which defines the members of the group that shall be referenced when this object is specified in a protocol transaction. Each read access specification shall consist of two parts: 1) an Object_Identifier and 2) a List Of Property References. All members of the group shall be objects that reside in the same device that maintains the Group object. See the ASN.1 production for ReadAccessSpecification in Clause 21.

Nesting of group objects is not permitted; that is, the List_Of_Group_Members shall not refer to the Present_Value property of a Group object.

12.14.6 Present_Value

This property is a list that contains the values of all the properties specified in the List_Of_Group_Members. This is a "read only" property; it cannot be used to write a set of values to the members of the group. The Present_Value list shall be reconstructed each time the property is read by fetching the member properties. *(NOTE: This requirement is to reduce concurrency problems that could result if the Present_Value were stored.)*

12.14.7 Profile_Name

This optional property, of type CharacterString, is the name of an object profile to which this object conforms. To ensure uniqueness, a profile name must begin with a vendor identifier code (see Clause 23) in base-10 integer format, followed by a dash. All subsequent characters are administered by the organization registered with that vendor identifier code. The vendor identifier code that prefixes the profile name shall indicate the organization that publishes and maintains the profile document named by the remainder of the profile name. This vendor identifier need not have any relationship to the vendor identifier of the device within which the object resides.

A profile defines a set of additional properties, behavior, and/or requirements for this object beyond those specified here. This standard defines only the format of the names of profiles. The definition of the profiles themselves is outside the scope of this standard.

12.15 Life Safety Point Object Type

The Life Safety Point object type defines a standardized object whose properties represent the externally visible characteristics associated with initiating and indicating devices in fire, life safety and security applications. The condition of a Life Safety Point object is represented by a *mode* and a *state*.

Mode changes determine the object's inner logic and, consequently, influence the evaluation of the state. Typically, the operating *mode* would be under operator control.

The *state* of the object represents the condition of the controller according to the logic internal to the device. The implementation of the logic applied to such controllers to determine the various possible states is a local matter.

Typical applications of the Life Safety Point object include automatic fire detectors, pull stations, sirens, supervised printers, etc. Similar objects can be identified in security control panels.

The Life Safety Point object type and its properties are summarized in Table 12-18 and described in detail in this subclause.

NOTE: Do not confuse the Present_Value state with the Event_State property, which reflects the offnormal state of the Life Safety Point object.

Table 12-18. Properties of the Life Safety Point Object Type

Property Identifier	Property Datatype	Conformance Code
Object_Identifier	BACnetObjectIdentifier	R
Object_Name	CharacterString	R
Object_Type	BACnetObjectType	R
Present_Value	BACnetLifeSafetyState	R[1]
Tracking_Value	BACnetLifeSafetyState	O
Description	CharacterString	O
Device_Type	CharacterString	O
Status_Flags	BACnetStatusFlags	R
Event_State	BACnetEventState	R
Reliability	BACnetReliability	R[1]
Out_Of_Service	BOOLEAN	R
Mode	BACnetLifeSafetyMode	W
Accepted_Modes	List of BACnetLifeSafetyMode	R
Time_Delay	Unsigned	O[2]
Notification_Class	Unsigned	O[2]
Life_Safety_Alarm_Values	List of BACnetLifeSafetyState	O[2]
Alarm_Values	List of BACnetLifeSafetyState	O[2]
Fault_Values	List of BACnetLifeSafetyState	O[2]
Event_Enable	BACnetEventTransitionBits	O[2]
Acked_Transitions	BACnetEventTransitionBits	O[2]
Notify_Type	BACnetNotifyType	O[2]
Event_Time_Stamps	BACnetARRAY [3] of BACnetTimeStamp	O[2]
Silenced	BACnetSilencedState	R
Operation_Expected	BACnetLifeSafetyOperation	R
Maintenance_Required	BACnetMaintenance	O
Setting	Unsigned8	O
Direct_Reading	REAL	O[3]
Units	BACnetEngineeringUnits	O[3]
Member_Of	List of BACnetDeviceObjectReference	O
Profile_Name	CharacterString	O

[1] These properties are required to be writable when Out_Of_Service is TRUE.

[2] These properties are required if the object supports intrinsic alarming.
[3] If either of these properties is present, then both must be present.

12.15.1 Object_Identifier

This property, of type BACnetObjectIdentifier, is a numeric code that is used to identify the object. It shall be unique within the BACnet Device that maintains it.

12.15.2 Object_Name

This property, of type CharacterString, shall represent a name for the object that is unique within the BACnet Device that maintains it. The minimum length of the string shall be one character. The set of characters used in the Object_Name shall be restricted to printable characters.

12.15.3 Object_Type

This property, of type BACnetObjectType, indicates membership in a particular object type class. The value of this property shall be LIFE_SAFETY_POINT.

12.15.4 Present_Value

This property, of type BACnetLifeSafetyState, reflects the state of the Life Safety Point object. The means of deriving the Present_Value shall be a local matter. Present_Value may latch non-NORMAL state values until reset. The Present_Value property shall be writable when Out_Of_Service is TRUE.

12.15.5 Tracking_Value

This optional property, of type BACnetLifeSafetyState, reflects the non-latched state of the Life Safety Point object. The means of deriving the state shall be a local matter. Unlike Present_Value, which may latch non-NORMAL state values until reset, Tracking_Value shall continuously track changes in the state.

12.15.6 Description

This optional property, of type CharacterString, is a string of printable characters whose content is not restricted.

12.15.7 Device_Type

This optional property, of type CharacterString, is a text description of the physical device that the Life Safety Point object represents.

12.15.8 Status_Flags

This property, of type BACnetStatusFlags, represents four Boolean flags that indicate the general "health" of the Life Safety Point object. Three of the flags are associated with the values of other properties of this object. A more detailed status could be determined by reading the properties that are linked to these flags. The relationship between individual flags is not defined by the protocol. The four flags are

{IN_ALARM, FAULT, OVERRIDDEN, OUT_OF_SERVICE}

where:

IN_ALARM	Logical FALSE (0) if the Event_State property has a value of NORMAL, otherwise logical TRUE (1).
FAULT	Logical TRUE (1) if the Reliability property does not have a value of NO_FAULT_DETECTED, otherwise logical FALSE (0).
OVERRIDDEN	Logical TRUE (1) if the point has been overridden by some mechanism local to the BACnet Device. In this context "overridden" is taken to mean that the physical input(s) are no longer tracking changes to the Present_Value property and the Reliability property is no longer a reflection of the physical input(s). Otherwise, the value is logical FALSE (0).
OUT_OF_SERVICE	Logical TRUE (1) if the Out_Of_Service property has a value of TRUE, otherwise logical FALSE (0).

12.15.9 Event_State

The Event_State property, of type BACnetEventState, is included in order to provide a way to determine if this object has an active event state associated with it. The Event_State property shall indicate the event state of the object. If the object does not support intrinsic reporting, then the value of this property shall be NORMAL.

12.15.10 Reliability

The reliability property, of type BACnetReliability, provides an indication of whether the Present_Value or the operation of the physical input(s) in question are "reliable" as far as the BACnet Device or operator can determine and, if not, why. The Reliability property for this object type may have any of the following values:

{NO_FAULT_DETECTED, OPEN_LOOP, SHORTED_LOOP, MULTI_STATE_FAULT, UNRELIABLE_OTHER}.

12.15.10.1 Conditions for Generating a TO-FAULT Event

A TO-FAULT event is generated under these conditions:

(a) the TO-FAULT flag must be enabled in the Event_Enable property, and
(b) the Present_Value must equal at least one of the values in the Fault_Values list.

12.15.11 Out_Of_Service

The Out_Of_Service property, of type BOOLEAN, is an indication whether (TRUE) or not (FALSE) the input(s) or process the object represents is not in service. This means that changes to the Present_Value property are decoupled from the input(s) or process when the value of Out_Of_Service is TRUE. In addition, the Reliability property and the corresponding state of the FAULT flag of the Status_Flags property shall be decoupled when Out_Of_Service is TRUE. While the Out_Of_Service property is TRUE, the Present_Value and Reliability properties may be changed to any value as a means of simulating specific fixed conditions or for testing purposes. Other functions that depend on the state of the Present_Value or Reliability properties shall respond to changes made to these properties while Out_Of_Service is TRUE, as if those changes had occurred to the input(s) or process.

12.15.12 Mode

This writable property, of type BACnetLifeSafetyMode, shall convey the desired operating mode for the Life Safety Point object. The Life Safety Point object shall generate CHANGE_OF_LIFE_SAFETY event notifications for any mode transition if the respective flags TO-OFFNORMAL, TO-FAULT or TO-NORMAL are set in the Event_Enable property (see 12.15.16.1, 12.15.16.2, 12.15.17.1, 12.15.17.2, 12.15.18.1, and 12.15.18.2).

12.15.13 Accepted_Modes

This read-only property, of type List of BACnetLifeSafetyMode, shall specify all values the Mode property accepts when written to using BACnet services. Even though a mode is listed in this property, the write may be denied by the object due to the internal state of the object at that time. The value of the Accepted_Modes property does not depend on the internal state of the object and shall not change when the internal state changes. If a write is denied, a Result(-) specifying an 'Error Class' of PROPERTY and an 'Error Code' of VALUE_OUT_OF_RANGE shall be returned. Internal computation in the object may set the Mode property to a value other than one of those listed in the Accepted_Modes property.

12.15.14 Time_Delay

This optional property, of type Unsigned, shall specify the minimum period of time in seconds that the Present_Value must remain:

(a) equal to any of the values in the Life_Safety_Alarm_Values property before a TO-OFFNORMAL event is generated, or
(b) equal to any one of the values in the Alarm_Values property before a TO-OFFNORMAL event is generated, or
(c) not equal to any of the values in the Life_Safety_Alarm_Values property, Alarm_Values property, or Fault_Values property before a TO-NORMAL event is generated.

This property is required if intrinsic reporting is supported by this object.

12.15.15 Notification_Class

This optional property, of type Unsigned, shall specify the notification class to be used when handling and generating event notifications for this object. The Notification_Class property implicitly refers to a Notification Class object that has a Notification_Class property with the same value. This property is required if intrinsic reporting is supported by this object.

12.15.16 Life_Safety_Alarm_Values

This optional property, of type List of BACnetLifeSafetyState, shall specify any states the Present_Value must equal before a TO-OFFNORMAL event is generated and event state LIFE_SAFETY_ALARM is entered. This property is required if intrinsic reporting is supported by this object.

12.15.16.1 Conditions for Generating a TO-OFFNORMAL Event

A TO-OFFNORMAL event is generated under these conditions:

(a) the TO-OFFNORMAL flag must be enabled in the Event_Enable property, and
(b) the Present_Value must equal any of the values in the Life_Safety_Alarm_Values list, and
(c) the Present_Value must remain within the Life_Safety_Alarm_Values list for a minimum period of time, specified by the Time_Delay property.

New events shall be generated upon a change of Mode if the TO-OFFNORMAL flag is set in the Event_Enable property.

12.15.16.2 Conditions for Generating a TO-NORMAL Event

Once equal, the Present_Value must become not equal to any of the states in the Life_Safety_Alarm_Values property, and not equal to any of the states in the Alarm_Values property, and not equal to any of the states in the Fault_Values property, before a TO-NORMAL event is generated under these conditions:

(a) the TO-NORMAL flag must be enabled in the Event_Enable property, and
(b) the Present_Value must remain not equal to any of the states in the Life_Safety_Alarm_Values property, and
(c) the Present_Value must remain not equal to any of the states in the Alarm_Values property, and
(d) the Present_Value must remain not equal to any of the states in the Fault_Values property, and
(e) the Present_Value must remain equal to the same value for a minimum period of time, specified by the Time_Delay property.

New events shall be generated upon a change of Mode if the TO-NORMAL flag is set in the Event_Enable property.

12.15.17 Alarm_Values

This optional property, of type List of BACnetLifeSafetyState, shall specify any states the Present_Value must equal before a TO-OFFNORMAL event is generated and event state OFFNORMAL is entered. This property is required if intrinsic reporting is supported by this object.

12.15.17.1 Conditions for Generating a TO-OFFNORMAL Event

A TO-OFFNORMAL event is generated under these conditions:

(a) the TO-OFFNORMAL flag must be enabled in the Event_Enable property, and
(b) the Present_Value must equal any of the values in the Alarm_Values list, and
(c) the Present_Value must remain within the Alarm_Values list for a minimum period of time specified by the Time_Delay property.

New events shall be generated upon a change of Mode if the TO-OFFNORMAL flag is set in the Event_Enable property.

12.15.17.2 Conditions for Generating a TO-NORMAL Event

Conditions for generating a TO-NORMAL event are defined in 12.15.16.2.

12.15.18 Fault_Values

This optional property, of type List of BACnetLifeSafetyState, shall specify any states the Present_Value must equal before a TO-FAULT event is generated. If Present_Value becomes equal to any of the states in the Fault_Values list, and no physical

fault has been detected for any inputs that the Present_Value represents, then the Reliability property shall have the value MULTI_STATE_FAULT. The Fault_Values property is required if intrinsic reporting is supported by this object.

New events shall be generated upon a change of Mode if the TO-FAULT flag is set in the Event_Enable property.

12.15.18.1 Conditions for Generating a TO-FAULT Event

A TO-FAULT event is generated under these conditions:

(a) the TO-FAULT flag must be enabled in the Event_Enable property, and
(b) the Present_Value must equal at least one of the values in the Fault_Values list.

12.15.18.2 Conditions for Generating a TO-NORMAL Event

Conditions for generating a TO-NORMAL event are defined in 12.15.16.2.

12.15.19 Event_Enable

This optional property, of type BACnetEventTransitionBits, shall convey three flags that separately enable and disable reporting of TO-OFFNORMAL, TO-FAULT, and TO-NORMAL events that are based on Present_Value and/or Mode changes. This property is required if intrinsic reporting is supported by this object.

12.15.20 Acked_Transitions

This optional property, of type BACnetEventTransitionBits, shall convey three flags that separately indicate the receipt of acknowledgments for TO-OFFNORMAL, TO-FAULT, and TO-NORMAL events. These flags shall be cleared upon the occurrence of the corresponding event and set under any of these conditions:

(a) upon receipt of the corresponding acknowledgment;
(b) upon the occurrence of the event if the corresponding flag is not set in the Event_Enable property (meaning event notifications will not be generated for this condition and thus no acknowledgment is expected);
(c) upon the occurrence of the event if the corresponding flag is set in the Event_Enable property and the corresponding flag in the Ack_Required property of the Notification Class object implicitly referenced by the Notification_Class property of this object is not set (meaning no acknowledgment is expected).

This property is required if intrinsic reporting is supported by this object.

12.15.21 Notify_Type

This optional property, of type BACnetNotifyType, shall convey whether the notifications generated by the object should be Events or Alarms. This property is required if intrinsic reporting is supported by this object.

12.15.22 Event_Time_Stamps

This optional property, of type BACnetARRAY[3] of BACnetTimeStamp, shall convey the times of the last event notifications for TO-OFFNORMAL, TO-FAULT, and TO-NORMAL events, respectively. Time stamps of type Time or Date shall have 'FF' in each octet, and Sequence number time stamps shall have the value 0 if no event notification of that type has been generated since the object was created. This property is required if intrinsic reporting is supported by this object.

12.15.23 Silenced

This property, of type BACnetSilencedState, shall indicate whether the most recently occurring transition for this object that has produced an audible or visual indication has been silenced by the receipt of a LifeSafetyOperation service request or a local process.

12.15.24 Operation_Expected

The Operation_Expected property, of type BACnetLifeSafetyOperation, shall specify the next operation expected by this object to handle a specific life safety situation.

12.15.25 Maintenance_Required

This optional property, of type BACnetMaintenance, shall indicate the type of maintenance required for the life safety point. This may be periodic maintenance, or a "parameter-determined" maintenance, such as dirtiness value for an associated detector, and shall be determined locally.

12.15.26 Setting

This optional property, of type Unsigned8, shall be used to convey the desired setting of the input(s) or process used to determine the logical state of the Present_Value. The range of the Setting property shall be from 0 (least sensitive) to 100 (most sensitive). The interpretation of the setting and the actual number of useful settings for a given Life Safety Point object shall be a local matter.

12.15.27 Direct_Reading

This optional property, of type REAL, shall indicate an analog quantity that reflects the measured or calculated reading from an initiating device. The manner in which this reading is used to determine the logical state of the object shall be a local matter. If this property is present, then the Units property shall also be present.

12.15.28 Units

This optional property, of type BACnetEngineeringUnits, shall indicate the units of the quantity represented by the Direct_Reading property. If this property is present, then the Direct_Reading property shall also be present.

12.15.29 Member_Of

This optional property, of type List of BACnetDeviceObjectReference, shall indicate those Life Safety Zone objects of which this Life Safety Point object is considered to be a zone member. Each object in the Member_Of list shall be a Life Safety Zone object.

This property may be restricted to only support references to objects inside of the device containing the Life Safety Point object. If the property is writable and is restricted to referencing objects within the containing device, an attempt to write a reference to an object outside the containing device into this property shall cause a Result(-) to be returned with an error class of PROPERTY and an error code of OPTIONAL_FUNCTIONALITY_NOT_SUPPORTED.

12.15.30 Profile_Name

This optional property, of type CharacterString, is the name of an object profile to which this object conforms. To ensure uniqueness, a profile name must begin with a vendor identifier code (see Clause 23) in base-10 integer format, followed by a dash. All subsequent characters are administered by the organization registered with that vendor identifier code. The vendor identifier code that prefixes the profile name shall indicate the organization that publishes and maintains the profile document named by the remainder of the profile name. This vendor identifier need not have any relationship to the vendor identifier of the device within which the object resides.

A profile defines a set of additional properties, behavior, and/or requirements for this object beyond those specified here. This standard defines only the format of the names of profiles. The definition of the profiles themselves is outside the scope of this standard.

12.16 Life Safety Zone Object Type

The Life Safety Zone object type defines a standardized object whose properties represent the externally visible characteristics associated with an arbitrary group of BACnet Life Safety Point and Life Safety Zone objects in fire, life safety and security applications. The condition of a Life Safety Zone object is represented by a *mode* and a *state*.

Mode changes determine the object's inner logic and, consequently, influence the evaluation of the state. Typically, the operating *mode* would be under operator control.

The *state* of the object represents the condition of the controller according to the logic internal to the device. The implementation of the logic applied to such controllers to determine the various possible states is a local matter.

Typical applications of the Life Safety Zone object include fire zones, panel zones, detector lines, extinguishing controllers, remote transmission controllers, etc. Similar objects can be identified in security control panels.

The Life Safety Zone object type and its properties are summarized in Table 12-19 and described in detail in this subclause.

NOTE: Do not confuse the Present_Value state with the Event_State property, which reflects the offnormal state of the Life Safety Zone object.

Table 12-19. Properties of the Life Safety Zone Object Type

Property Identifier	Property Datatype	Conformance Code
Object_Identifier	BACnetObjectIdentifier	R
Object_Name	CharacterString	R
Object_Type	BACnetObjectType	R
Present_Value	BACnetLifeSafetyState	R[1]
Tracking_Value	BACnetLifeSafetyState	O
Description	CharacterString	O
Device_Type	CharacterString	O
Status_Flags	BACnetStatusFlags	R
Event_State	BACnetEventState	R
Reliability	BACnetReliability	R[1]
Out_Of_Service	BOOLEAN	R
Mode	BACnetLifeSafetyMode	W
Accepted_Modes	List of BACnetLifeSafetyMode	R
Time_Delay	Unsigned	O[2]
Notification_Class	Unsigned	O[2]
Life_Safety_Alarm_Values	List of BACnetLifeSafetyState	O[2]
Alarm_Values	List of BACnetLifeSafetyState	O[2]
Fault_Values	List of BACnetLifeSafetyState	O[2]
Event_Enable	BACnetEventTransitionBits	O[2]
Acked_Transitions	BACnetEventTransitionBits	O[2]
Notify_Type	BACnetNotifyType	O[2]
Event_Time_Stamps	BACnetARRAY [3] of BACnetTimeStamp	O[2]
Silenced	BACnetSilencedState	R
Operation_Expected	BACnetLifeSafetyOperation	R
Maintenance_Required	BOOLEAN	O
Zone_Members	List of BACnetDeviceObjectReference	R
Member_Of	List of BACnetDeviceObjectReference	O
Profile_Name	CharacterString	O

[1] These properties are required to be writable when Out_Of_Service is TRUE.
[2] These properties are required if the object supports intrinsic alarming.

12.16.1 Object_Identifier

This property, of type BACnetObjectIdentifier, is a numeric code that is used to identify the object. It shall be unique within the BACnet Device that maintains it.

12.16.2 Object_Name

This property, of type CharacterString, shall represent a name for the object that is unique within the BACnet Device that maintains it. The minimum length of the string shall be one character. The set of characters used in the Object_Name shall be restricted to printable characters.

12.16.3 Object_Type

This property, of type BACnetObjectType, indicates membership in a particular object type class. The value of this property shall be LIFE_SAFETY_ZONE.

12.16.4 Present_Value

This property, of type BACnetLifeSafetyState, reflects the state of the Life Safety Zone object. The means of deriving the Present_Value shall be a local matter. Present_Value may latch non-NORMAL state values until reset. The Present_Value property shall be writable when Out_Of_Service is TRUE.

12.16.5 Tracking_Value

This optional property, of type BACnetLifeSafetyState, reflects the non-latched state of the Life Safety Zone object. The means of deriving the state shall be a local matter. Unlike Present_Value, which may latch non-NORMAL state values until reset, Tracking_Value shall continuously track changes in the state.

12.16.6 Description

This optional property, of type CharacterString, is a string of printable characters whose content is not restricted.

12.16.7 Device_Type

This optional property, of type CharacterString, is a text description of the physical zone or area that the Life Safety Zone object represents. It will typically be used to describe the locale of the Life Safety Point objects that are Zone_Members of the Life Safety Zone object.

12.16.8 Status_Flags

This property, of type BACnetStatusFlags, represents four Boolean flags that indicate the general "health" of the Life Safety Zone object. Three of the flags are associated with the values of other properties of this object. A more detailed status could be determined by reading the properties that are linked to these flags. The relationship between individual flags is not defined by the protocol. The four flags are

{IN_ALARM, FAULT, OVERRIDDEN, OUT_OF_SERVICE}

where:

IN_ALARM	Logical FALSE (0) if the Event_State property has a value of NORMAL, otherwise logical TRUE (1).
FAULT	Logical TRUE (1) if the Reliability property does not have a value of NO_FAULT_DETECTED, otherwise logical FALSE (0).
OVERRIDDEN	Logical TRUE (1) if the point has been overridden by some mechanism local to the BACnet Device. In this context "overridden" is taken to mean that the physical input(s) are no longer tracking changes to the Present_Value property and the Reliability property is no longer a reflection of the physical input(s). Otherwise, the value is logical FALSE (0).
OUT_OF_SERVICE	Logical TRUE (1) if the Out_Of_Service property has a value of TRUE, otherwise logical FALSE (0).

12.16.9 Event_State

The Event_State property, of type BACnetEventState, is included in order to provide a way to determine if this object has an active event state associated with it. The Event_State property shall indicate the event state of the object.

12.16.10 Reliability

The reliability property, of type BACnetReliability, provides an indication of whether the Present_Value or the operation of the physical input(s) in question are "reliable" as far as the BACnet Device or operator can determine and, if not, why. The Reliability property for this object type may have any of the following values:

{NO_FAULT_DETECTED, OPEN_LOOP, SHORTED_LOOP, MULTI_STATE_FAULT, UNRELIABLE_OTHER}.

12.16.10.1 Conditions for Generating a TO-FAULT Event

A TO-FAULT event is generated under these conditions:

(a) the TO-FAULT flag must be enabled in the Event_Enable property, and
(b) the Present_Value must equal at least one of the values in the Fault_Values list.

12.16.11 Out_Of_Service

The Out_Of_Service property, of type BOOLEAN, is an indication whether (TRUE) or not (FALSE) the input(s) or process the object represents is not in service. This means that changes to the Present_Value property are decoupled from the input(s) or process when the value of Out_Of_Service is TRUE. In addition, the Reliability property and the corresponding state of the FAULT flag of the Status_Flags property shall be decoupled when Out_Of_Service is TRUE. While the Out_Of_Service property is TRUE, the Present_Value and Reliability properties may be changed to any value as a means of simulating specific fixed conditions or for testing purposes. Other functions that depend on the state of the Present_Value or Reliability properties shall respond to changes made to these properties while Out_Of_Service is TRUE, as if those changes had occurred to the input(s) or process.

12.16.12 Mode

This writable property, of type BACnetLifeSafetyMode, shall convey the desired operating mode for the Life Safety Point object. The Life Safety Point object shall generate CHANGE_OF_LIFE_SAFETY event notifications for any mode transition if the respective flags TO-OFFNORMAL, TO-FAULT or TO-NORMAL are set in the Event_Enable property (see 12.16.16.1, 12.16.16.2, 12.16.17.1, 12.16.17.2, 12.16.18.1, and 12.16.18.2).

12.16.13 Accepted_Modes

This read-only property, of type List of BACnetLifeSafetyMode, shall specify all values the Mode property accepts when written to using BACnet services. Even though a mode is listed in this property, the write may be denied by the object due to the internal state of the object at that time. The value of the Accepted_Modes property does not depend on the internal state of the object and shall not change when the internal state changes. If a write is denied, a Result(-) specifying an 'Error Class' of PROPERTY and an 'Error Code' of VALUE_OUT_OF_RANGE shall be returned. Internal computation in the object may set the Mode property to a value other than one of those listed in the Accepted_Modes property.

12.16.14 Time_Delay

This optional property, of type Unsigned, shall specify the minimum period of time in seconds that the Present_Value must remain:

(a) equal to any of the values in the Life_Safety_Alarm_Values property before a TO-OFFNORMAL event is generated, or
(b) equal to any one of the values in the Alarm_Values property before a TO-OFFNORMAL event is generated, or
(c) not equal to any of the values in the Life_Safety_Alarm_Values property, Alarm_Values property, or Fault_Values property before a TO-NORMAL event is generated.

This property is required if intrinsic reporting is supported by this object.

12.16.15 Notification_Class

This optional property, of type Unsigned, shall specify the notification class to be used when handling and generating event notifications for this object. The Notification_Class property implicitly refers to a Notification Class object that has a Notification_Class property with the same value.

12.16.16 Life_Safety_Alarm_Values

This optional property, of type List of BACnetLifeSafetyState, shall specify any states the Present_Value must equal before a TO-OFFNORMAL event is generated and event state LIFE_SAFETY_ALARM is entered. This property is required if intrinsic reporting is supported by this object.

12.16.16.1 Conditions for Generating a TO-OFFNORMAL Event

A TO-OFFNORMAL event is generated under these conditions:

(a) the TO-OFFNORMAL flag must be enabled in the Event_Enable property, and
(b) the Present_Value must equal any of the values in the Life_Safety_Alarm_Values list, and
(c) the Present_Value must remain within the Life_Safety_Alarm_Values list for a minimum period of time, specified by the Time_Delay property.

New events shall be generated upon a change of Mode if the TO-OFFNORMAL flag is set in the Event_Enable property.

12.16.16.2 Conditions for Generating a TO-NORMAL Event

Once equal, the Present_Value must become not equal to any of the states in the Life_Safety_Alarm_Values property, and not equal to any of the states in the Alarm_Values property, and not equal to any of the states in the Fault_Values property, before a TO-NORMAL event is generated under these conditions:

(a) the TO-NORMAL flag must be enabled in the Event_Enable property, and
(b) the Present_Value must remain not equal to any of the states in the Life_Safety_Alarm_Values property, and
(c) the Present_Value must remain not equal to any of the states in the Alarm_Values property, and
(d) the Present_Value must remain not equal to any of the states in the Fault_Values property, and
(e) the Present_Value must remain equal to the same value for a minimum period of time, specified by the Time_Delay property.

New events shall be generated upon a change of Mode if the TO-NORMAL flag is set in the Event_Enable property.

12.16.17 Alarm_Values

This optional property, of type List of BACnetLifeSafetyState, shall specify any states the Present_Value must equal before a TO-OFFNORMAL event is generated and event state OFFNORMAL is entered. This property is required if intrinsic reporting is supported by this object.

12.16.17.1 Conditions for Generating a TO-OFFNORMAL Event

A TO-OFFNORMAL event is generated under these conditions:

(a) the TO-OFFNORMAL flag must be enabled in the Event_Enable property, and
(b) the Present_Value must equal at least one of the values in the Alarm_Values list, and
(c) the Present_Value must remain equal to the same value for a minimum period of time, specified by the Time_Delay property.

New events shall be generated upon a change of Mode if the TO-OFFNORMAL flag is set in the Event_Enable property.

12.16.17.2 Conditions for Generating a TO-NORMAL Event

Conditions for generating a TO-NORMAL event are defined in 12.16.16.2.

12.16.18 Fault_Values

This optional property, of type List of BACnetLifeSafetyState, shall specify any states the Present_Value must equal before a TO-FAULT event is generated. If Present_Value becomes equal to any of the states in the Fault_Values list, and no physical

fault has been detected for any inputs that the Present_Value represents, then the Reliability property shall have the value MULTI_STATE_FAULT. The Fault_Values property is required if intrinsic reporting is supported by this object.

New events shall be generated upon a change of Mode if the TO-FAULT flag is set in the Event_Enable property.

12.16.18.1 Conditions for Generating a TO-FAULT Event

A TO-FAULT event is generated under these conditions:

(a) the TO-FAULT flag must be enabled in the Event_Enable property, and
(b) the Present_Value must equal at least one of the values in the Fault_Values list.

12.16.18.2 Conditions for Generating a TO-NORMAL Event

Conditions for generating a TO-NORMAL event are defined in 12.16.16.2.

12.16.19 Event_Enable

This optional property, of type BACnetEventTransitionBits, shall convey three flags that separately enable and disable reporting of TO-OFFNORMAL, TO-FAULT, and TO-NORMAL events that are based on Present_Value or Mode changes.

12.16.20 Acked_Transitions

This optional property, of type BACnetEventTransitionBits, shall convey three flags that separately indicate the receipt of acknowledgments for TO-OFFNORMAL, TO-FAULT, and TO-NORMAL events. These flags shall be cleared upon the occurrence of the corresponding event and set under any of these conditions:

(a) upon receipt of the corresponding acknowledgment;
(b) upon the occurrence of the event if the corresponding flag is not set in the Event_Enable property (meaning event notifications will not be generated for this condition and thus no acknowledgment is expected);
(c) upon the occurrence of the event if the corresponding flag is set in the Event_Enable property and the corresponding flag in the Ack_Required property of the Notification Class object implicitly referenced by the Notification_Class property of this object is not set (meaning no acknowledgment is expected).

12.16.21 Notify_Type

This optional property, of type BACnetNotifyType, shall convey whether the notifications generated by the object should be Events or Alarms.

12.16.22 Event_Time_Stamps

This optional property, of type BACnetARRAY[3] of BACnetTimeStamp, shall convey the times of the last event notifications for TO-OFFNORMAL, TO-FAULT, and TO-NORMAL events, respectively. Time stamps of type Time or Date shall have 'FF' in each octet, and Sequence number time stamps shall have the value 0 if no event notification of that type has been generated since the object was created. This property is required if intrinsic reporting is supported by this object.

12.16.23 Silenced

This property, of type BACnetSilencedState, shall indicate whether the most recently occurring transition for this object that has produced an audible or visual indication has been silenced by the receipt of a LifeSafetyOperation service request or a local process.

12.16.24 Operation_Expected

The Operation_Expected property, of type BACnetLifeSafetyOperation, shall specify the next operation expected by this object to handle a specific life safety situation.

12.16.25 Maintenance_Required

This optional property, of type BOOLEAN, shall indicate that maintenance is required for one or more of the life safety points that are members of this zone.

12.16.26 Zone_Members

This property, of type List of BACnetDeviceObjectReference, shall indicate which Life Safety Point and Life Safety Zone objects are members of the zone represented by this object.

This property may be restricted to only support references to objects inside of the device containing the Life Safety Zone object. If the property is writable and is restricted to referencing objects within the containing device, an attempt to write a reference to an object outside the containing device into this property shall cause a Result(-) to be returned with an error class of PROPERTY and an error code of OPTIONAL_FUNCTIONALITY_NOT_SUPPORTED.

12.16.27 Member_Of

This optional property, of type List of BACnetDeviceObjectReference, shall indicate those Life Safety Zone objects of which this Life Safety Zone object is considered to be a zone member. Each object in the Member_Of list shall be a Life Safety Zone object.

This property may be restricted to only support references to objects inside of the device containing the Life Safety Zone object. If the property is writable and is restricted to referencing objects within the containing device, an attempt to write a reference to an object outside the containing device into this property shall cause a Result(-) to be returned with an error class of PROPERTY and an error code of OPTIONAL_FUNCTIONALITY_NOT_SUPPORTED.

12.16.28 Profile_Name

This optional property, of type CharacterString, is the name of an object profile to which this object conforms. To ensure uniqueness, a profile name must begin with a vendor identifier code (see Clause 23) in base-10 integer format, followed by a dash. All subsequent characters are administered by the organization registered with that vendor identifier code. The vendor identifier code that prefixes the profile name shall indicate the organization that publishes and maintains the profile document named by the remainder of the profile name. This vendor identifier need not have any relationship to the vendor identifier of the device within which the object resides.

A profile defines a set of additional properties, behavior, and/or requirements for this object beyond those specified here. This standard defines only the format of the names of profiles. The definition of the profiles themselves is outside the scope of this standard.

12.17 Loop Object Type

The Loop object type defines a standardized object whose properties represent the externally visible characteristics of any form of feedback control loop. Flexibility is achieved by providing three independent gain constants with no assumed values for units. The appropriate gain units are determined by the details of the control algorithm, which is a local matter. The Loop object type and its properties are summarized in Table 12-20 and described in detail in this subclause. Figure 12-2 illustrates the relationship between the Loop object properties and the other objects referenced by the loop.

Table 12-20. Properties of the Loop Object Type

Property Identifier	Property Datatype	Conformance Code
Object_Identifier	BACnetObjectIdentifier	R
Object_Name	CharacterString	R
Object_Type	BACnetObjectType	R
Present_Value	REAL	R
Description	CharacterString	O
Status_Flags	BACnetStatusFlags	R
Event_State	BACnetEventState	R
Reliability	BACnetReliability	O
Out_Of_Service	BOOLEAN	R
Update_Interval	Unsigned	O
Output_Units	BACnetEngineeringUnits	R
Manipulated_Variable_Reference	BACnetObjectPropertyReference	R
Controlled_Variable_Reference	BACnetObjectPropertyReference	R
Controlled_Variable_Value	REAL	R
Controlled_Variable_Units	BACnetEngineeringUnits	R
Setpoint_Reference	BACnetSetpointReference	R
Setpoint	REAL	R
Action	BACnetAction	R
Proportional_Constant	REAL	O[1]
Proportional_Constant_Units	BACnetEngineeringUnits	O[1]
Integral_Constant	REAL	O[2]
Integral_Constant_Units	BACnetEngineeringUnits	O[2]
Derivative_Constant	REAL	O[3]
Derivative_Constant_Units	BACnetEngineeringUnits	O[3]
Bias	REAL	O
Maximum_Output	REAL	O
Minimum_Output	REAL	O
Priority_For_Writing	Unsigned	R
COV_Increment	REAL	O[4]
Time_Delay	Unsigned	O[5]
Notification_Class	Unsigned	O[5]
Error_Limit	REAL	O[5]
Event_Enable	BACnetEventTransitionBits	O[5]
Acked_Transitions	BACnetEventTransitionBits	O[5]
Notify_Type	BACnetNotifyType	O[5]
Event_Time_Stamps	BACnetARRAY[3] of BACnetTimeStamp	O[5]
Profile_Name	CharacterString	O

[1] If one of these optional properties is present, then both of these properties shall be present.
[2] If one of these optional properties is present, then both of these properties shall be present.
[3] If one of these optional properties is present, then both of these properties shall be present.
[4] This property is required if the object supports COV reporting.
[5] These properties are required if the object supports intrinsic reporting.

12.17.1 Object_Identifier

This property, of type BACnetObjectIdentifier, is a numeric code that is used to identify the object. It shall be unique within the BACnet Device that maintains it**.**

12.17.2 Object_Name

This property, of type CharacterString, shall represent a name for the object that is unique within the BACnet Device that maintains it. The minimum length of the string shall be one character. The set of characters used in the Object_Name shall be restricted to printable characters.

12.17.3 Object_Type

This property, of type BACnetObjectType, indicates membership in a particular object-type class. The value of this property shall be LOOP.

12.17.4 Present_Value

This property indicates the current output value of the loop algorithm in units of the Output_Units property.

12.17.5 Description

This property, of type CharacterString, is a string of printable characters whose content is not restricted.

12.17.6 Status_Flags

This property, of type BACnetStatusFlags, represents four Boolean flags that indicate the general "health" of the loop. Three of the flags are associated with the values of other properties of this object. A more detailed status could be determined by reading the properties that are linked to these flags. The relationship between individual flags is not defined by the protocol. The four flags are

{IN_ALARM, FAULT, OVERRIDDEN, OUT_OF_SERVICE}

where:

IN_ALARM	Logical FALSE (0) if the Event_State property has a value of NORMAL, otherwise Logical TRUE (1).
FAULT	Logical TRUE (1) if the Reliability property is present and does not have a value of NO_FAULT_DETECTED, otherwise logical FALSE (0).
OVERRIDDEN	Logical TRUE (1) if the point has been overridden by some mechanism local to the BACnet Device. In this context "overridden" is taken to mean that the Present_Value property is not changeable through BACnet services. Otherwise, the value is logical FALSE (0).
OUT_OF_SERVICE	Logical TRUE (1) if the Out_Of_Service property has a value of TRUE, otherwise logical FALSE(0).

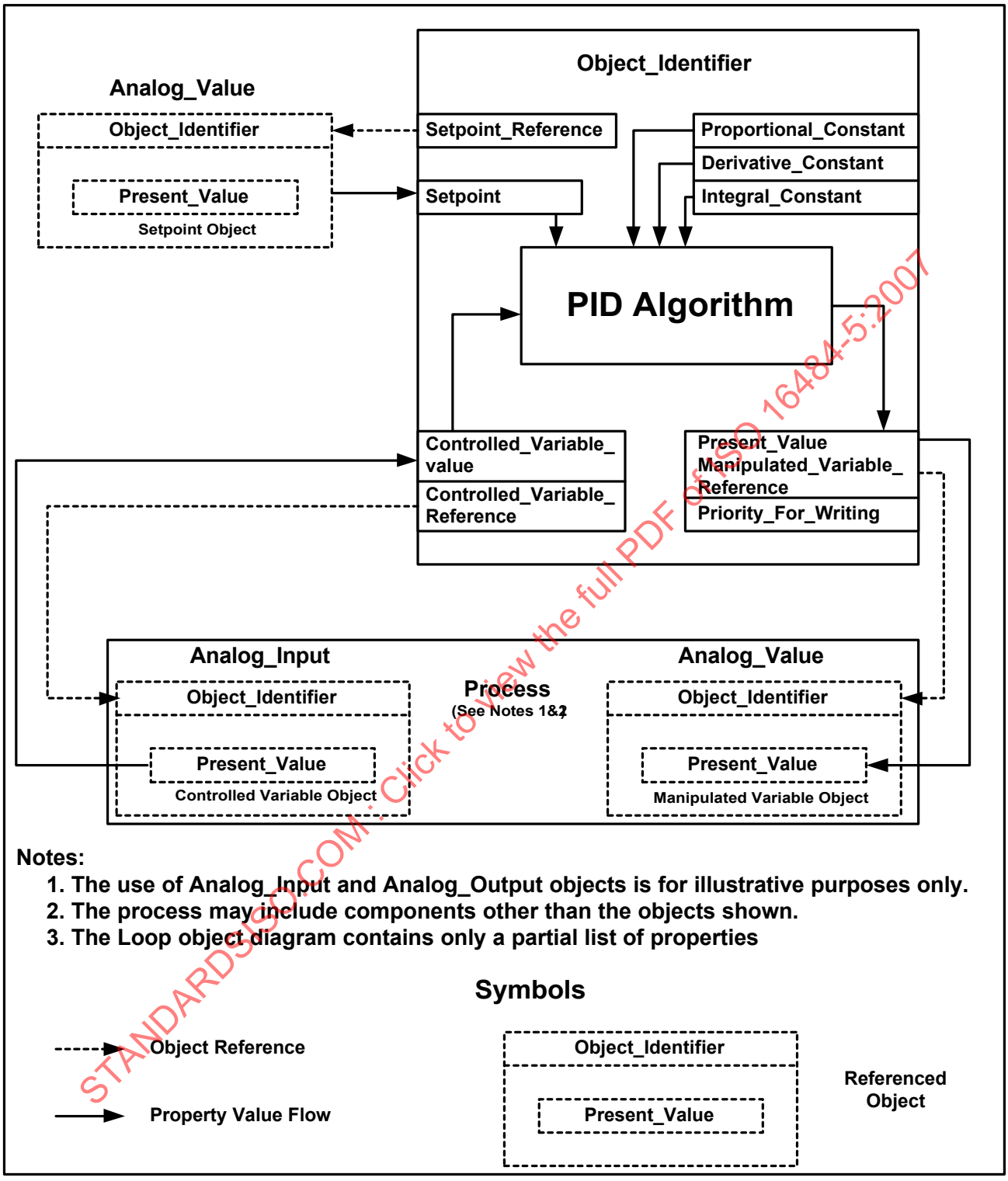


Figure 12-2. Loop object structure with its referenced objects.

12.17.7 Event_State

The Event_State property, of type BACnetEventState, is included in order to provide a way to determine if this object has an active event state associated with it. If the object supports intrinsic reporting, then the Event_State property shall indicate the

event state of the object. If the object does not support intrinsic reporting, then the value of this property shall be NORMAL. If the Reliability property is present and does not have a value of NO_FAULT_DETECTED, then the value of the Event_State property shall be FAULT. Changes in the Event_State property to the value FAULT are considered to be "fault" events.

12.17.8 Reliability

The Reliability property, of type BACnetReliability, provides an indication of whether the Present_Value of the loop in question is reliable as far as the BACnet Device or operator can determine and, if not, why. The Reliability property for this object type may have any of the following values:

{NO_FAULT_DETECTED, OPEN_LOOP, UNRELIABLE_OTHER}.

12.17.9 Out_Of_Service

The Out_Of_Service property, of type BOOLEAN, is an indication whether (TRUE) or not (FALSE) the algorithm this object represents is or is not in service.

12.17.10 Update_Interval

This property, of type Unsigned, indicates the interval in milliseconds at which the loop algorithm updates the output (Present_Value property).

NOTE: No property that represents the interval at which the process variable is sampled or the algorithm is executed is part of this object. The Update_Interval value may be the same as these other values but could also be different depending on the algorithm utilized. The sampling or execution interval is a local matter and need not be represented as part of this object.

12.17.11 Output Units

This property, of type BACnetEngineeringUnits, indicates the engineering units for the output (Present_Value property) of this control loop.

12.17.12 Manipulated_Variable_Reference

This property is of type BACnetObjectPropertyReference. The output (Present_Value) of the control loop is written to the object and property designated by the Manipulated_Variable_Reference. It is normally the Present_Value of an Analog Output object used to position a device, but it could also be another object or property, such as that used to stage multiple Binary Outputs.

12.17.13 Controlled_Variable_Reference

This property is of type BACnetObjectPropertyReference. The Controlled_Variable_Reference identifies the property used to set the Controlled_Variable_Value property of the Loop object. It is normally the Present_Value property of an Analog Input object used for measuring a process variable, temperature, for example, but it could also be another object, such as an Analog Value, which calculates a minimum or maximum of a group of Analog Inputs for use in discriminator control.

12.17.14 Controlled_Variable_Value

This property, of type REAL, is the value of the property of the object referenced by the Controlled_Variable_Reference property. This control loop compares the Controlled_Variable_Value with the Setpoint to calculate the error.

12.17.15 Controlled_Variable_Units

This property, of type BACnetEngineeringUnits, indicates the engineering units for the Controlled_Variable_Value property of this object.

12.17.16 Setpoint_Reference

This property, of type BACnetSetpointReference, is a list of references that has a length of zero or one. A length of zero indicates that the setpoint for this control loop is fixed and is contained in the Setpoint property. A length of one indicates that the property of another object contains the setpoint value used for this Loop object and specifies that property.

12.17.17 Setpoint

This property, of type REAL, is the value of the loop setpoint or of the property of the object referenced by the Setpoint_Reference, expressed in units of the Controlled_Variable_Units property.

12.17.18 Action

This property, of type BACnetAction, defines whether the loop is DIRECT or REVERSE acting.

12.17.19 Proportional_Constant

This property, of type REAL, is the value of the proportional gain parameter used by the loop algorithm. It may be used to represent any of the various forms of gain for the proportional control mode, such as overall gain, throttling range, or proportional band. If either the Proportional_Constant property or the Proportional_Constant_Units property are present, then both of these properties shall be present.

12.17.20 Proportional_Constant_Units

This property, of type BACnetEngineeringUnits, indicates the engineering units of the Proportional_Constant property of this object. If either the Proportional_Constant_Units property or the Proportional_Constant property are present, then both of these properties shall be present.

12.17.21 Integral_Constant

This property, of type REAL, is the value of the integral gain parameter used by the loop algorithm. It may be used to represent any of the various forms of gain for the integral control mode, such as reset time or rate. If either the Integral_Constant property or the Proportional_Constant_Units property are present, then both of these properties shall be present.

12.17.22 Integral_Constant_Units

This property, of type BACnetEngineeringUnits, indicates the engineering units of the Integral_Constant property of this object. If either the Integral_Constant_Units property or the Proportional_Constant property are present, then both of these properties shall be present.

12.17.23 Derivative_Constant

This property, of type REAL, is the value of the derivative gain parameter used by the loop algorithm. It may be used to represent any of the various forms of gain for the derivative control mode, such as derivative time or rate time. If either the Derivative_Constant property or the Derivative_Constant_Units property are present, then both of these properties shall be present.

12.17.24 Derivative_Constant_Units

This property, of type BACnetEngineeringUnits, indicates the engineering units of the Derivative_Constant property of this object. If either the Derivative_Constant_Units property or the Derivative_Constant property are present, then both of these properties shall be present.

12.17.25 Bias

This property, of type REAL, is the bias value used by the loop algorithm expressed in units of the Output_Units property.

12.17.26 Maximum_Output

This property, of type REAL, is the maximum value of the Present_Value property as limited by the PID loop algorithm. It is normally used to prevent the algorithm from controlling beyond the range of the controlled device and to prevent integral term "windup."

12.17.27 Minimum_Output

This property, of type REAL, is the minimum value of the Present_Value property as limited by the loop algorithm. It is normally used to prevent the algorithm from controlling beyond the range of the controlled device and to prevent integral term "windup."

12.17.28 Priority_For_Writing

Loop objects may be used to control the commandable property of an object. This property, of type Unsigned, provides a priority to be used by the command prioritization mechanism. It identifies the particular priority slot in the Priority_Array of the Controlled_Variable_Reference that is controlled by this loop. It shall have a value in the range 1-16.

12.17.29 COV_Increment

This property, of type REAL, shall specify the minimum change in Present_Value that will cause a COVNotification to be issued to subscriber COV-clients. This property is required if COV reporting is supported by this object.

12.17.30 Time_Delay

This property, of type Unsigned, shall specify the minimum period of time in seconds that the difference between the Setpoint and the Controlled_Variable_Value (the Error) must remain outside the band defined by the Error_Limit property before a TO-OFFNORMAL event is generated or within the same band before a TO-NORMAL event is generated. This property is required if intrinsic reporting is supported by this object.

12.17.31 Notification_Class

This property, of type Unsigned, shall specify the notification class to be used when handling and generating event notifications for this object. The Notification_Class property implicitly refers to a Notification Class object that has a Notification_Class property with the same value. This property is required if intrinsic reporting is supported by this object.

12.17.32 Error_Limit

This property, of type REAL, shall convey the absolute magnitude that the difference between the Setpoint and Controlled_Variable_Value (the Error) must exceed before a TO-OFFNORMAL event is generated. This property is required if intrinsic reporting is supported by this object.

12.17.33 Event_Enable

This property, of type BACnetEventTransitionBits, shall convey three flags that separately enable and disable reporting of TO-OFFNORMAL, TO-FAULT, and TO-NORMAL events. This property is required if intrinsic reporting is supported by this object.

12.17.34 Acked_Transitions

This property, of type BACnetEventTransitionBits, shall convey three flags that separately indicate the receipt of acknowledgments for TO-OFFNORMAL, TO-FAULT, and TO-NORMAL events. These flags shall be cleared upon the occurrence of the corresponding event and set under any of these conditions:

(a) upon receipt of the corresponding acknowledgment;
(b) upon the occurrence of the event if the corresponding flag is not set in the Event_Enable property (meaning event notifications will not be generated for this condition and thus no acknowledgment is expected);
(c) upon the occurrence of the event if the corresponding flag is set in the Event_Enable property and the corresponding flag in the Ack_Required property of the Notification Class object implicitly referenced by the Notification_Class property of this object is not set (meaning no acknowledgment is expected).

This property is required if intrinsic reporting is supported by this object.

12.17.35 Notify_Type

This property, of type BACnetNotifyType, shall convey whether the notifications generated by the object should be Events or Alarms. This property is required if intrinsic reporting is supported by this object.

12.17.36 Event_Time_Stamps

This optional property, of type BACnetARRAY[3] of BACnetTimeStamp, shall convey the times of the last event notifications for TO-OFFNORMAL, TO-FAULT, and TO-NORMAL events, respectively. Time stamps of type Time or Date shall have 'FF' in each octet and Sequence number time stamps shall have the value 0 if no event notification of that type has been generated since the object was created. This property is required if intrinsic reporting is supported by this object.

12.17.37 Profile_Name

This optional property, of type CharacterString, is the name of an object profile to which this object conforms. To ensure uniqueness, a profile name must begin with a vendor identifier code (see Clause 23) in base-10 integer format, followed by a dash. All subsequent characters are administered by the organization registered with that vendor identifier code. The vendor identifier code that prefixes the profile name shall indicate the organization that publishes and maintains the profile document named by the remainder of the profile name. This vendor identifier need not have any relationship to the vendor identifier of the device within which the object resides.

A profile defines a set of additional properties, behavior, and/or requirements for this object beyond those specified here. This standard defines only the format of the names of profiles. The definition of the profiles themselves is outside the scope of this standard.

12.18 Multi-state Input Object Type

The Multi-state Input object type defines a standardized object whose Present_Value represents the result of an algorithmic process within the BACnet Device in which the object resides. The algorithmic process itself is a local matter and is not defined by the protocol. For example, the Present_Value or state of the Multi-state Input object may be the result of a logical combination of multiple binary inputs or the threshold of one or more analog inputs or the result of a mathematical computation. The Present_Value property is an integer number representing the state. The State_Text property associates a description with each state. The Multi-state Input object type and its properties are summarized in Table 12-21 and described in detail in this subclause.

NOTE: Do not confuse the Present_Value state with the Event_State property, which reflects the offnormal state of the Multi-State Input.

Table 12-21. Properties of the Multi-state Input Object Type

Property Identifier	Property Datatype	Conformance Code
Object_Identifier	BACnetObjectIdentifier	R
Object_Name	CharacterString	R
Object_Type	BACnetObjectType	R
Present_Value	Unsigned	R[1]
Description	CharacterString	O
Device_Type	CharacterString	O
Status_Flags	BACnetStatusFlags	R
Event_State	BACnetEventState	R
Reliability	BACnetReliability	O[2]
Out_Of_Service	BOOLEAN	R
Number_Of_States	Unsigned	R
State_Text	BACnetARRAY[N] of CharacterString	O
Time_Delay	Unsigned	O[3]
Notification_Class	Unsigned	O[3]
Alarm_Values	List of Unsigned	O[3]
Fault_Values	List of Unsigned	O[3]
Event_Enable	BACnetEventTransitionBits	O[3]
Acked_Transitions	BACnetEventTransitionBits	O[3]
Notify_Type	BACnetNotifyType	O[3]
Event_Time_Stamps	BACnetARRAY[3] of BACnetTimeStamp	O[3]
Profile_Name	CharacterString	O

[1] This property is required to be writable when Out_Of_Service is TRUE.
[2] This property shall be required if Fault_Values is present.
[3] These properties are required if the object supports intrinsic reporting.

12.18.1 Object_Identifier

This property, of type BACnetObjectIdentifier, is a numeric code that is used to identify the object. It shall be unique within the BACnet Device that maintains it.

12.18.2 Object_Name

This property, of type CharacterString, shall represent a name for the object that is unique within the BACnet Device that maintains it. The minimum length of the string shall be one character. The set of characters used in the Object_Name shall be restricted to printable characters.

12.18.3 Object_Type

This property, of type BACnetObjectType, indicates membership in a particular object type class. The value of this property shall be MULTISTATE_INPUT.

12.18.4 Present_Value

This property, of type Unsigned, reflects the logical state of the input. The logical state of the input shall be one of 'n' states, where 'n' is the number of states defined in the Number_Of_States property. The means used to determine the current state is

a local matter. The Present_Value property shall always have a value greater than zero. The Present_Value property shall be writable when Out_Of_Service is TRUE.

12.18.5 Description

This property, of type CharacterString, is a string of printable characters whose content is not restricted.

12.18.6 Device_Type

This property, of type CharacterString, is a text description of the multi-state input. It will typically be used to describe the type of device attached to the multi-state input.

12.18.7 Status_Flags

This property, of type BACnetStatusFlags, represents four Boolean flags that indicate the general "health" of the multi-state input. Three of the flags are associated with the values of other properties of this object. A more detailed status could be determined by reading the properties that are linked to these flags. The relationship between individual flags is not defined by the protocol. The four flags are

{IN_ALARM, FAULT, OVERRIDDEN, OUT_OF_SERVICE}

where:

IN_ALARM	Logical FALSE (0) if the Event_State property has a value of NORMAL, otherwise logical TRUE (1).
FAULT	Logical TRUE (1) if the Reliability property is present and does not have a value of NO_FAULT_DETECTED, otherwise logical FALSE (0).
OVERRIDDEN	Logical TRUE (1) if the point has been overridden by some mechanism local to the BACnet Device. In this context "overridden" is taken to mean that the Present_Value and Reliability properties are no longer tracking changes to the physical input. Otherwise, the value is logical FALSE (0).
OUT_OF_SERVICE	Logical TRUE (1) if the Out_Of_Service property has a value of TRUE, otherwise logical FALSE (0).

12.18.8 Event_State

The Event_State property, of type BACnetEventState, is included in order to provide a way to determine if this object has an active event state associated with it. If the object supports intrinsic reporting, then the Event_State property shall indicate the event state of the object. If the object does not support intrinsic reporting, then:

(a) if the Reliability property is not present, then the value of Event_State shall be NORMAL, or
(b) if the Reliability property is present and Reliability is NO_FAULT_DETECTED then Event_State shall be NORMAL, or
(c) If the Reliability property is present and Reliability is not NO_FAULT_DETECTED then Event_State shall be FAULT.

12.18.9 Reliability

The reliability property, of type BACnetReliability, provides an indication of whether the Present_Value or the operation of the physical inputs in question are "reliable" as far as the BACnet Device or operator can determine and, if not, why. The Reliability property is required to be present if the Fault_Values property is present. The Reliability property for this object type may have any of the following values:

{NO_FAULT_DETECTED, NO_SENSOR, OVER_RANGE, UNDER_RANGE, OPEN_LOOP, SHORTED_LOOP, MULTI_STATE_FAULT, UNRELIABLE_OTHER}.

12.18.9.1 Conditions for Generating a TO-FAULT Event

A TO-FAULT event is generated under these conditions:

(a) the Reliability property becomes not equal to NO_FAULT_DETECTED, and
(b) the TO-FAULT flag must be enabled in the Event_Enable property.

12.18.10 Out_Of_Service

The Out_Of_Service property, of type BOOLEAN, is an indication whether (TRUE) or not (FALSE) the inputs the object represents are not in service. This means that the Present_Value property is decoupled from the input and will not track changes to the input when the value of Out_Of_Service is TRUE. In addition, the Reliability property and the corresponding state of the FAULT flag of the Status_Flags property shall be decoupled from the input when Out_Of_Service is TRUE. While the Out_Of_Service property is TRUE, the Present_Value and Reliability properties may be changed to any value as a means of simulating specific fixed conditions or for testing purposes. Other functions that depend on the state of the Present_Value or Reliability properties shall respond to changes made to these properties while Out_Of_Service is TRUE, as if those changes had occurred in the input.

12.18.11 Number_Of_States

This property, of type Unsigned, defines the number of states that the Present_Value may have. The Number_Of_States property shall always have a value greater than zero. If the value of this property is changed, the size of the State_Text array, if present, shall also be changed to the same value.

12.18.12 State_Text

This property is a BACnetARRAY of character strings representing descriptions of all possible states of the Present_Value. The number of descriptions matches the number of states defined in the Number_Of_States property. The Present_Value, interpreted as an integer, serves as an index into the array. If the size of this array is changed, the Number_Of_States property shall also be changed to the same value.

12.18.13 Time_Delay

This optional property, of type Unsigned, shall specify the minimum period of time in seconds that the Present_Value must remain equal to any one of the values in the Alarm_Values property before a TO-OFFNORMAL event is generated or remain not equal to any of the values in the Alarm_Values property before a TO-NORMAL event is generated. This property is required if intrinsic reporting is supported by this object.

12.18.14 Notification_Class

This property, of type Unsigned, shall specify the notification class to be used when handling and generating event notifications for this object. The Notification_Class property implicitly refers to a Notification Class object that has a Notification_Class property with the same value. This property is required if intrinsic reporting is supported by this object.

12.18.15 Alarm_Values

This property, of type List of Unsigned, shall specify any states the Present_Value must equal before a TO-OFFNORMAL event is generated. This property is required if intrinsic reporting is supported by this object.

12.18.15.1 Conditions for Generating a TO-OFFNORMAL Event

A TO-OFFNORMAL event is generated under these conditions:

(a) the Present_Value property must equal at least one of the values in the Alarm_Values list, and
(b) the Present_Value must remain equal to the same value for a minimum period of time, specified in the Time_Delay property, and
(c) the TO-OFFNORMAL flag must be enabled in the Event_Enable property.

12.18.15.2 Conditions for Generating a TO-NORMAL Event

Once equal, the Present_Value must become not equal to any of the states in this property and not equal to any of the states in the Fault_Values property before a TO-NORMAL event is generated under these conditions:

(a) the Present_Value must remain not equal to any of the values in the Alarm_Values list for a minimum period of time, specified in the Time_Delay property, and
(b) the Present_Value must remain not equal to any of the states in the Fault_Values property, and
(c) the TO-NORMAL flag must be enabled in the Event_Enable property.

12.18.16 Fault_Values

This optional property, of type List of Unsigned, shall specify any states the Present_Value must equal before a TO-FAULT event is generated. If Present_Value becomes equal to any of the states in the Fault_Values list, and no physical fault has been detected for any inputs that the Present_Value represents, then the Reliability property shall have the value MULTI_STATE_FAULT. The Fault_Values property is required if intrinsic reporting is supported by this object.

12.18.17 Event_Enable

This property, of type BACnetEventTransitionBits, shall convey three flags that separately enable and disable reporting of TO-OFFNORMAL, TO-FAULT, and TO-NORMAL events. This property is required if intrinsic reporting is supported by this object.

12.18.18 Acked_Transitions

This property, of type BACnetEventTransitionBits, shall convey three flags that separately indicate the receipt of acknowledgments for TO-OFFNORMAL, TO-FAULT, and TO-NORMAL events. These flags shall be cleared upon the occurrence of the corresponding event and set under any of these conditions:

(a) upon receipt of the corresponding acknowledgment;
(b) upon the occurrence of the event if the corresponding flag is not set in the Event_Enable property (meaning event notifications will not be generated for this condition and thus no acknowledgment is expected);
(c) upon the occurrence of the event if the corresponding flag is set in the Event_Enable property and the corresponding flag in the Ack_Required property of the Notification Class object implicitly referenced by the Notification_Class property of this object is not set (meaning no acknowledgment is expected).

This property is required if intrinsic reporting is supported by this object.

12.18.19 Notify_Type

This property, of type BACnetNotifyType, shall convey whether the notifications generated by the object should be Events or Alarms. This property is required if intrinsic reporting is supported by this object.

12.18.20 Event_Time_Stamps

This optional property, of type BACnetARRAY[3] of BACnetTimeStamp, shall convey the times of the last event notifications for TO-OFFNORMAL, TO-FAULT, and TO-NORMAL events, respectively. Time stamps of type Time or Date shall have 'FF' in each octet and Sequence number time stamps shall have the value 0 if no event notification of that type has been generated since the object was created. This property is required if intrinsic reporting is supported by this object.

12.18.21 Profile_Name

This optional property, of type CharacterString, is the name of an object profile to which this object conforms. To ensure uniqueness, a profile name must begin with a vendor identifier code (see Clause 23) in base-10 integer format, followed by a dash. All subsequent characters are administered by the organization registered with that vendor identifier code. The vendor identifier code that prefixes the profile name shall indicate the organization that publishes and maintains the profile document named by the remainder of the profile name. This vendor identifier need not have any relationship to the vendor identifier of the device within which the object resides.

A profile defines a set of additional properties, behavior, and/or requirements for this object beyond those specified here. This standard defines only the format of the names of profiles. The definition of the profiles themselves is outside the scope of this standard.

12.19 Multi-state Output Object Type

The Multi-state Output object type defines a standardized object whose properties represent the desired state of one or more physical outputs or processes within the BACnet Device in which the object resides. The actual functions associated with a specific state are a local matter and not specified by the protocol. For example, a particular state may represent the active/inactive condition of several physical outputs or perhaps the value of an analog output. The Present_Value property is an unsigned integer number representing the state. The State_Text property associates a description with each state.

The Multi-state Output object type and its properties are summarized in Table 12-22 and described in detail in this subclause.

Table 12-22. Properties of the Multi-state Output Object Type

Property Identifier	Property Datatype	Conformance Code
Object_Identifier	BACnetObjectIdentifier	R
Object_Name	CharacterString	R
Object_Type	BACnetObjectType	R
Present_Value	Unsigned	W
Description	CharacterString	O
Device_Type	CharacterString	O
Status_Flags	BACnetStatusFlags	R
Event_State	BACnetEventState	R
Reliability	BACnetReliability	O
Out_Of_Service	BOOLEAN	R
Number_Of_States	Unsigned	R
State_Text	BACnetARRAY[N]of CharacterString	O
Priority_Array	BACnetPriorityArray	R
Relinquish_Default	Unsigned	R
Time_Delay	Unsigned	O[1]
Notification_Class	Unsigned	O[1]
Feedback_Value	Unsigned	O[1]
Event_Enable	BACnetEventTransitionBits	O[1]
Acked_Transitions	BACnetEventTransitionBits	O[1]
Notify_Type	BACnetNotifyType	O[1]
Event_Time_Stamps	BACnetARRAY[3] of BACnetTimeStamp	O[1]
Profile_Name	CharacterString	O

[1] These properties are required if the object supports intrinsic reporting.

12.19.1 Object_Identifier

This property, of type BACnetObjectIdentifier, is a numeric code that is used to identify the object. It shall be unique within the BACnet Device that maintains it.

12.19.2 Object_Name

This property, of type CharacterString, shall represent a name for the object that is unique within the BACnet Device that maintains it. The minimum length of the string shall be one character. The set of characters used in the Object_Name shall be restricted to printable characters.

12.19.3 Object_Type

This property, of type BACnetObjectType, indicates membership in a particular object type class. The value of this property shall be MULTISTATE_OUTPUT.

12.19.4 Present_Value (Commandable)

This property, of type Unsigned, reflects the logical state of an output. The logical state of the output shall be one of 'n' states, where 'n' is the number of states defined in the Number_Of_States property. How the Present_Value is used is a local matter. The Present_Value property shall always have a value greater than zero.

12.19.5 Description

This property, of type CharacterString, is a string of printable characters whose content is not restricted.

12.19.6 Device_Type

This property, of type CharacterString, is a text description of the physical device connected to the multi-state output. It will typically be used to describe the type of device attached to the multi-state output.

12.19.7 Status_Flags

This property, of type BACnetStatusFlags, represents four Boolean flags that indicate the general "health" of the multi-state output. Three of the flags are associated with the values of other properties of this object. A more detailed status could be determined by reading the properties that are linked to these flags. The relationship between individual flags is not defined by the protocol. The four flags are

{IN_ALARM, FAULT, OVERRIDDEN, OUT_OF_SERVICE}

where:

IN_ALARM	Logical FALSE (0) if the Event_State property has a value of NORMAL, otherwise logical TRUE (1).
FAULT	Logical TRUE (1) if the Reliability property is present and does not have a value of NO_FAULT_DETECTED, otherwise logical FALSE (0).
OVERRIDDEN	Logical TRUE (1) if the point has been overridden by some mechanism local to the BACnet Device. In this context "overridden" is taken to mean that the physical output is no longer tracking changes to the Present_Value property and the Reliability property is no longer a reflection of the physical output. Otherwise, the value is logical FALSE (0).
OUT_OF_SERVICE	Logical TRUE (1) if the Out_Of_Service property has a value of TRUE, otherwise logical FALSE (0).

12.19.8 Event_State

The Event_State property, of type BACnetEventState, is included in order to provide a way to determine if this object has an active event state associated with it. If the object supports intrinsic reporting, then the Event_State property shall indicate the event state of the object. If the object does not support intrinsic reporting, then the value of this property shall be NORMAL.

12.19.9 Reliability

The reliability property, of type BACnetReliability, provides an indication of whether the Present_Value or the operation of the physical outputs in question are "reliable" as far as the BACnet Device or operator can determine and, if not, why. The Reliability property for this object type may have any of the following values:

{NO_FAULT_DETECTED,OPEN_LOOP,SHORTED_LOOP,NO_OUTPUT,UNRELIABLE_OTHER}.

12.19.10 Out_Of_Service

The Out_Of_Service property, of type BOOLEAN, is an indication whether (TRUE) or not (FALSE) the output or process the object represents is not in service. This means that changes to the Present_Value property are decoupled from the output when the value of Out_Of_Service is TRUE. In addition, the Reliability property and the corresponding state of the FAULT flag of the Status_Flags property shall be decoupled when Out_Of_Service is TRUE. While the Out_Of_Service property is TRUE, the Present_Value and Reliability properties may still be changed to any value as a means of simulating specific fixed conditions or for testing purposes. Other functions that depend on the state of the Present_Value or Reliability properties shall respond to changes made to these properties while Out_Of_Service is TRUE, as if those changes had occurred to the output. The Present_Value property shall still be controlled by the BACnet command prioritization mechanism if Out_Of_Service is TRUE. See Clause 19.

12.19.11 Number_Of_States

This property, of type Unsigned, defines the number of states the Present_Value may have. The Number_Of_States property shall always have a value greater than zero**.** If the value of this property is changed, the size of the State_Text array, if present, shall also be changed to the same value.

12.19.12 State_Text

This property is a BACnetARRAY of character strings representing descriptions of all possible states of the Present_Value. The number of descriptions matches the number of states defined in the Number_Of_States property. The Present_Value, interpreted as an integer, serves as an index into the array. If the size of this array is changed, the Number_Of_States property shall also be changed to the same value.

12.19.13 Priority_Array

This property is a read-only array that contains prioritized commands that are in effect for this object. See Clause 19 for a description of the prioritization mechanism.

12.19.14 Relinquish_Default

This property is the default value to be used for the Present_Value property when all command priority values in the Priority_Array have a NULL value. See Clause 19.

12.19.15 Time_Delay

This property, of type Unsigned, shall specify the minimum period of time in seconds that the Present_Value must be different from the Feedback_Value property before a TO-OFFNORMAL event is generated or remain equal to the Feedback_Value property before a TO-NORMAL event is generated. This property is required if intrinsic reporting is supported by this object.

12.19.16 Notification_Class

This property, of type Unsigned, shall specify the notification class to be used when handling and generating event notifications for this object. The Notification_Class property implicitly refers to a Notification Class object that has a Notification_Class property with the same value. This property is required if intrinsic reporting is supported by this object.

12.19.17 Feedback_Value

This property, of type Unsigned, shall indicate the status of a feedback value from which the Present_Value must differ before a TO-OFFNORMAL event is generated. This property is required if intrinsic reporting is supported by this object. The manner by which the Feedback_Value is determined shall be a local matter.

12.19.17.1 Conditions for Generating a TO-OFFNORMAL Event

A TO-OFFNORMAL event is generated under these conditions:

(a) the Present_Value must be different from the Feedback_Value for a minimum period of time, specified by the Time_Delay property, and
(b) the TO-OFFNORMAL flag must be enabled in the Event_Enable property.

12.19.17.2 Conditions for Generating a TO-NORMAL Event

Once different, the Present_Value must become equal to this property before a TO-NORMAL event is generated under these conditions:

(a) the Present_Value must remain equal to the Feedback_Value for a minimum period of time, specified by the Time_Delay property, and
(b) the TO-NORMAL flag must be enabled in the Event_Enable property.

12.19.18 Event_Enable

This property, of type BACnetEventTransitionBits, shall convey three flags that separately enable and disable reporting of TO-OFFNORMAL, TO-FAULT, and TO-NORMAL events. This property is required if intrinsic reporting is supported by this object.

12.19.19 Acked_Transitions

This property, of type BACnetEventTransitionBits, shall convey three flags that separately indicate the receipt of acknowledgments for TO-OFFNORMAL, TO-FAULT, and TO-NORMAL events. These flags shall be cleared upon the occurrence of the corresponding event and set under any of these conditions:

(a) upon receipt of the corresponding acknowledgment;
(b) upon the occurrence of the event if the corresponding flag is not set in the Event_Enable property (meaning event notifications will not be generated for this condition and thus no acknowledgment is expected);
(c) upon the occurrence of the event if the corresponding flag is set in the Event_Enable property and the corresponding flag in the Ack_Required property of the Notification Class object implicitly referenced by the Notification_Class property of this object is not set (meaning no acknowledgment is expected).

This property is required if intrinsic reporting is supported by this object.

12.19.20 Notify_Type

This property, of type BACnetNotifyType, shall convey whether the notifications generated by the object should be Events or Alarms. This property is required if intrinsic reporting is supported by this object.

12.19.21 Event_Time_Stamps

This optional property, of type BACnetARRAY[3] of BACnetTimeStamp, shall convey the times of the last event notifications for TO-OFFNORMAL, TO-FAULT, and TO-NORMAL events respectively. Time stamps of type Time or Date shall have 'FF' in each octet and Sequence number time stamps shall have the value 0 if no event notification of that type has been generated since the object was created. This property is required if intrinsic reporting is supported by this object.

12.19.22 Profile_Name

This optional property, of type CharacterString, is the name of an object profile to which this object conforms. To ensure uniqueness, a profile name must begin with a vendor identifier code (see Clause 23) in base-10 integer format, followed by a dash. All subsequent characters are administered by the organization registered with that vendor identifier code. The vendor identifier code that prefixes the profile name shall indicate the organization that publishes and maintains the profile document named by the remainder of the profile name. This vendor identifier need not have any relationship to the vendor identifier of the device within which the object resides.

A profile defines a set of additional properties, behavior, and/or requirements for this object beyond those specified here. This standard defines only the format of the names of profiles. The definition of the profiles themselves is outside the scope of this standard.

12.20 Multi-state Value Object Type

The Multi-state Value object type defines a standardized object whose properties represent the externally visible characteristics of a multi-state value. A "multi-state value" is a control system parameter residing in the memory of the BACnet Device. The actual functions associated with a specific state are a local matter and not specified by the protocol. For example, a particular state may represent the active/inactive condition of several physical inputs and outputs or perhaps the value of an analog input or output. The Present_Value property is an unsigned integer number representing the state. The State_Text property associates a description with each state.

The Multi-state Value object type and its properties are summarized in Table 12-23 and described in detail in this subclause.

NOTE: Do not confuse the Present_Value state with the Event_State property, which reflects the offnormal state of the Multi-state Value.

Table 12-23. Properties of the Multi-state Value Object Type

Property Identifier	Property Datatype	Conformance Code
Object_Identifier	BACnetObjectIdentifier	R
Object_Name	CharacterString	R
Object_Type	BACnetObjectType	R
Present_Value	Unsigned	R[1]
Description	CharacterString	O
Status_Flags	BACnetStatusFlags	R
Event_State	BACnetEventState	R
Reliability	BACnetReliability	O[2]
Out_Of_Service	BOOLEAN	R
Number_Of_States	Unsigned	R
State_Text	BACnetARRAY[N] of CharacterString	O
Priority_Array	BACnetPriorityArray	O[3]
Relinquish_Default	Unsigned	O[3]
Time_Delay	Unsigned	O[4]
Notification_Class	Unsigned	O[4]
Alarm_Values	List of Unsigned	O[4]
Fault_Values	List of Unsigned	O[4]
Event_Enable	BACnetEventTransitionBits	O[4]
Acked_Transitions	BACnetEventTransitionBits	O[4]
Notify_Type	BACnetNotifyType	O[4]
Event_Time_Stamps	BACnetARRAY[3] of BACnetTimeStamp	O[4]
Profile_Name	CharacterString	O

[1] If Present_Value is commandable, then it is required to also be writable. This property is required to be writable when Out_Of_Service is TRUE.
[2] This property shall be required if Fault_Values is present.
[3] If Present_Value is commandable, then both of these properties shall be present.
[4] These properties are required if the object supports intrinsic reporting.

12.20.1 Object_Identifier

This property, of type BACnetObjectIdentifier, is a numeric code that is used to identify the object. It shall be unique within the BACnet Device that maintains it.

12.20.2 Object_Name

This property, of type CharacterString, shall represent a name for the object that is unique within the BACnet Device that maintains it. The minimum length of the string shall be one character. The set of characters used in the Object_Name shall be restricted to printable characters.

12.20.3 Object_Type

This property, of type BACnetObjectType, indicates membership in a particular object type class. The value of this property shall be MULTISTATE_VALUE.

12.20.4 Present_Value

This property, of type Unsigned, reflects the logical state of the multi-state value. The logical state of the multi-state value shall be one of 'n' states, where 'n' is the number of states defined in the Number_Of_States property. How the Present_Value is used is a local matter. The Present_Value property shall always have a value greater than zero. Present_Value shall be optionally commandable. If Present_Value is commandable for a given object instance, then the Priority_Array and Relinquish_Default properties shall also be present for that instance. The Present_Value property shall be writable when Out_Of_Service is TRUE.

12.20.5 Description

This property, of type CharacterString, is a string of printable characters whose content is not restricted.

12.20.6 Status_Flags

This property, of type BACnetStatusFlags, represents four Boolean flags that indicate the general "health" of the multi-state value. Three of the flags are associated with the values of other properties of this object. A more detailed status could be determined by reading the properties that are linked to these flags. The relationship between individual flags is not defined by the protocol. The four flags are

{IN_ALARM, FAULT, OVERRIDDEN, OUT_OF_SERVICE}

where:

IN_ALARM	Logical FALSE (0) if the Event_State property has a value of NORMAL, otherwise logical TRUE (1).
FAULT	Logical TRUE (1) if the Reliability property is present and does not have a value of NO_FAULT_DETECTED, otherwise logical FALSE (0).
OVERRIDDEN	Logical TRUE (1) if the point has been overridden by some mechanism local to the BACnet Device. In this context "overridden" is taken to mean that the physical output is no longer tracking changes to the Present_Value property and the Reliability property is no longer a reflection of the physical output. Otherwise, the value is logical FALSE (0).
OUT_OF_SERVICE	Logical TRUE (1) if the Out_Of_Service property has a value of TRUE, otherwise logical FALSE (0).

12.20.7 Event_State

The Event_State property, of type BACnetEventState, is included in order to provide a way to determine if this object has an active event state associated with it. If the object supports intrinsic reporting, then the Event_State property shall indicate the event state of the object. If the object does not support intrinsic reporting then:

(a) if the Reliability property is not present, then the value of Event_State shall be NORMAL, or
(b) if the Reliability property is present and Reliability is NO_FAULT_DETECTED then Event_State shall be NORMAL, or
(c) if the Reliability property is present and Reliability is not NO_FAULT_DETECTED then Event_State shall be FAULT.

12.20.8 Reliability

The Reliability property, of type BACnetReliability, provides an indication of whether the Present_Value is "reliable" as far as the BACnet Device or operator can determine and, if not, why. The Reliability property is required to be present if the Fault_Values property is present. The Reliability property for this object type may have any of the following values:

{NO_FAULT_DETECTED, MULTI_STATE_FAULT, UNRELIABLE_OTHER}.

12.20.8.1 Conditions for Generating a TO-FAULT Event

A TO-FAULT event is generated under these conditions:

(a) the Reliability property becomes not equal to NO_FAULT_DETECTED, and
(b) the TO-FAULT flag must be enabled in the Event_Enable property.

12.20.9 Out_Of_Service

The Out_Of_Service property, of type BOOLEAN, is an indication whether (TRUE) or not (FALSE) the Present_Value property of the Multi-state Value object is prevented from being modified by software local to the BACnet device in which the object resides. When Out_Of_Service is TRUE the Present_Value property may still be written to freely. In addition, the Reliability property and the corresponding state of the FAULT flag of the Status_Flags property shall be decoupled when Out_Of_Service is TRUE. While the Out_Of_Service property is TRUE, the Present_Value and Reliability properties may be changed to any value as a means of simulating specific fixed conditions or for testing purposes. Other functions that depend on the state of the Present_Value or Reliability properties shall respond to changes made to these properties while Out_Of_Service is TRUE. If the Priority_Array and Relinquish_Default properties are present, then writing to the Present_Value property shall be controlled by the BACnet command prioritization mechanism. See Clause 19.

12.20.10 Number_Of_States

This property, of type Unsigned, defines the number of states the Present_Value may have. The Number_Of_States property shall always have a value greater than zero. If the value of this property is changed, the size of the State_Text array, if present, shall also be changed to the same value.

12.20.11 State_Text

This property is a BACnetARRAY of character strings representing descriptions of all possible states of the Present_Value. The number of descriptions matches the number of states defined in the Number_Of_States property. The Present_Value, interpreted as an integer, serves as an index into the array. If the size of this array is changed, the Number_Of_States property shall also be changed to the same value.

12.20.12 Priority_Array

This property is a read-only array that contains prioritized commands that are in effect for this object. See Clause 19 for a description of the prioritization mechanism. If either the Priority_Array property or the Relinquish_Default property are present, then both of them shall be present. If Present_Value is commandable, then Priority_Array and Relinquish_Default shall both be present.

12.20.13 Relinquish_Default

This property is the default value to be used for the Present_Value property when all command priority values in the Priority_Array property have a NULL value. See Clause 19. If either the Relinquish_Default property or the Priority_Array property are present, then both of them shall be present. If Present_Value is commandable, then Priority_Array and Relinquish_Default shall both be present.

12.20.14 Time_Delay

This property, of type Unsigned, shall specify the minimum period of time in seconds that the Present_Value must remain:

(a) equal to any one of the values in the Alarm_Values property before a TO-OFFNORMAL event is generated, or
(b) not equal to any of the values in the Alarm_Values property before a TO-NORMAL event is generated.

This property is required if intrinsic reporting is supported by this object.

12.20.15 Notification_Class

This property, of type Unsigned, shall specify the notification class to be used when handling and generating event notifications for this object. The Notification_Class property implicitly refers to a Notification Class object that has a Notification_Class property with the same value. This property is required if intrinsic reporting is supported by this object.

12.20.16 Alarm_Values

This property, of type List of Unsigned, shall specify any states the Present_Value must equal before a TO-OFFNORMAL event is generated. This property is required if intrinsic reporting is supported by this object.

12.20.16.1 Conditions for Generating a TO-OFFNORMAL Event

A TO-OFFNORMAL event is generated under these conditions:

(a) the Present_Value must equal at least one of the values in the Alarm_Values list, and
(b) the Present_Value must remain equal to the same value for a minimum period of time, specified by the Time_Delay property, and
(c) the TO-OFFNORMAL flag must be enabled in the Event_Enable property.

12.20.16.2 Conditions for Generating a TO-NORMAL Event

Once equal, the Present_Value must become not equal to any of the states in this property and not equal to any of the states in the Fault_Values property before a TO-NORMAL event is generated under these conditions:

(a) the Present_Value must remain not equal to any of the states in the Alarm_Values property for a minimum period of time, specified by the Time_Delay property, and
(b) the Present_Value must remain not equal to any of the states in the Fault_Values property, and
(c) the TO-NORMAL flag must be enabled in the Event_Enable property.

12.20.17 Fault_Values

This property, of type List of Unsigned, shall specify any states the Present_Value must equal before a TO-FAULT event is generated. If Present_Value becomes equal to any of the states in the Fault_Values list, and no physical fault has been detected for any inputs or outputs that the Present_Value represents, then the Reliability property shall have the value MULTI_STATE_FAULT. The Fault_Values property is required if intrinsic reporting is supported by this object.

12.20.18 Event_Enable

This property, of type BACnetEventTransitionBits, shall convey three flags that separately enable and disable reporting of TO-OFFNORMAL, TO-FAULT, and TO-NORMAL events. This property is required if intrinsic reporting is supported by this object.

12.20.19 Acked_Transitions

This property, of type BACnetEventTransitionBits, shall convey three flags that separately indicate the receipt of acknowledgments for TO-OFFNORMAL, TO-FAULT, and TO-NORMAL events. These flags shall be cleared upon the occurrence of the corresponding event and set under any of these conditions:

(a) upon receipt of the corresponding acknowledgment;
(b) upon the occurrence of the event if the corresponding flag is not set in the Event_Enable property (meaning event notifications will not be generated for this condition and thus no acknowledgment is expected);
(c) upon the occurrence of the event if the corresponding flag is set in the Event_Enable property and the corresponding flag in the Ack_Required property of the Notification Class object implicitly referenced by the Notification_Class property of this object is not set (meaning no acknowledgment is expected).

This property is required if intrinsic reporting is supported by this object.

12.20.20 Notify_Type

This property, of type BACnetNotifyType, shall convey whether the notifications generated by the object should be Events or Alarms. This property is required if intrinsic reporting is supported by this object.

12.20.21 Event_Time_Stamps

This optional property, of type BACnetARRAY[3] of BACnetTimeStamp, shall convey the times of the last event notifications for TO-OFFNORMAL, TO-FAULT, and TO-NORMAL events, respectively. Time stamps of type Time or Date shall have 'FF' in each octet and Sequence number time stamps shall have the value 0 if no event notification of that type has been generated since the object was created. This property is required if intrinsic reporting is supported by this object.

12.20.22 Profile_Name

This optional property, of type CharacterString, is the name of an object profile to which this object conforms. To ensure uniqueness, a profile name must begin with a vendor identifier code (see Clause 23) in base-10 integer format, followed by a dash. All subsequent characters are administered by the organization registered with that vendor identifier code. The vendor identifier code that prefixes the profile name shall indicate the organization that publishes and maintains the profile document named by the remainder of the profile name. This vendor identifier need not have any relationship to the vendor identifier of the device within which the object resides.

A profile defines a set of additional properties, behavior, and/or requirements for this object beyond those specified here. This standard defines only the format of the names of profiles. The definition of the profiles themselves is outside the scope of this standard.

12.21 Notification Class Object Type

The Notification Class object type defines a standardized object that represents and contains information required for the distribution of event notifications within BACnet systems. Notification Classes are useful for event-initiating objects that have identical needs in terms of how their notifications should be handled, what the destination(s) for their notifications should be, and how they should be acknowledged.

A notification class defines how event notifications shall be prioritized in their handling according to TO-OFFNORMAL, TO-FAULT, and TO-NORMAL events; whether these categories of events require acknowledgment (nearly always by a human operator); and what destination devices or processes should receive notifications.

The purpose of prioritization is to provide a means to ensure that alarms or event notifications with critical time considerations are not unnecessarily delayed. The possible range of priorities is 0 - 255. A lower number indicates a higher priority. The priority and the Network Priority (Clause 6.2.2) are associated as defined in Table 13-5. Priorities may be assigned to TO-OFFNORMAL, TO-FAULT, and TO-NORMAL events individually within a notification class.

The purpose of acknowledgment is to provide assurance that a notification has been acted upon by some other agent, rather than simply having been received correctly by another device. In most cases, acknowledgments come from human operators. TO-OFFNORMAL, TO-FAULT, and TO-NORMAL events may, or may not, require individual acknowledgment within a notification class.

It is often necessary for event notifications to be sent to multiple destinations or to different destinations based on the time of day or day of week. Notification Classes may specify a list of destinations, each of which is qualified by time, day of week, and type of handling. A destination specifies a set of days of the week (Monday through Sunday) during which the destination is considered viable by the Notification Class object. In addition, each destination has a FromTime and ToTime, which specify a window, on those days of the week, during which the destination is viable. If an event that uses a Notification Class object occurs and the day is one of the days of the week that is valid for a given destination and the time is within the window specified in the destination, then the destination shall be sent a notification. Destinations may be further qualified, as applicable, by any combination of the three event transitions TO-OFFNORMAL, TO-FAULT, or TO-NORMAL.

The destination also defines the recipient device to receive the notification and a process within the device. Processes are identified by numeric handles that are only meaningful to the destination device. The administration of these handles is a local matter. The recipient device may be specified by either its unique Device Object_Identifier or its BACnetAddress. In the latter case, a specific node address, a multicast address, or a broadcast address may be used. The destination further specifies whether the notification shall be sent using a confirmed or unconfirmed event notification.

The Notification Class object and its properties are summarized in Table 12-24 and described in detail in this subclause.

Table 12-24. Properties of the Notification Class Object Type

Property Identifier	Property Datatype	Conformance Code
Object_Identifier	BACnetObjectIdentifier	R
Object_Name	CharacterString	R
Object_Type	BACnetObjectType	R
Description	CharacterString	O
Notification_Class	Unsigned	R
Priority	BACnetARRAY[3] of Unsigned	R
Ack_Required	BACnetEventTransitionBits	R
Recipient_List	List of BACnetDestination	R
Profile_Name	CharacterString	O

12.21.1 Object_Identifier

This property, of type BACnetObjectIdentifier, is a numeric code that is used to identify the object. It shall be unique within the BACnet Device that maintains it**.**

12.21.2 Object_Name

This property, of type CharacterString, shall represent a name for the object that is unique within the BACnet Device that maintains it. The minimum length of the string shall be one character. The set of characters used in the Object_Name shall be restricted to printable characters.

12.21.3 Object_Type

This property, of type BACnetObjectType, indicates membership in a particular object type class. The value of this property shall be NOTIFICATION_CLASS.

12.21.4 Description

This property, of type CharacterString, is a string of printable characters whose content is not restricted.

12.21.5 Notification_Class

This property, of type Unsigned, shall indicate the numeric value of this notification class and shall be equal to the instance number of the Notification Class object. Event-initiating objects shall use this number to refer to this Notification Class object indirectly.

12.21.6 Priority

This property, of type BACnetARRAY[3] of Unsigned, shall convey the priority to be used for event notifications for TO-OFFNORMAL, TO-FAULT, and TO-NORMAL events, respectively. Priorities shall range from 0 - 255 inclusive. A lower number indicates a higher priority. The priority and the Network Priority (see 6.2.2) are associated as defined in Table 13-5.

12.21.7 Ack_Required

This property, of type BACnetEventTransitionBits, shall convey three separate flags that represent whether acknowledgment shall be required in notifications generated for TO-OFFNORMAL, TO-FAULT, and TO-NORMAL event transitions, respectively.

12.21.8 Recipient_List

This property, of type List of BACnetDestination, shall convey a list of one or more recipient destinations to which notifications shall be sent when event-initiating objects using this class detect the occurrence of an event. The destinations themselves define a structure of parameters that is summarized in Table 12-25.

Table 12-25. Components of a BACnetDestination

Parameter	Type	Description
Valid Days	BACnetDaysOfWeek	The set of days of the week on which this destination may be used between From Time and To Time
From Time, To Time	Time	The window of time (inclusive) during which the destination is viable on the days of the week Valid Days
Recipient	BACnetRecipient	The destination device(s) to receive notifications
Process Identifier	Unsigned32	The handle of a process within the recipient device that is to receive the event notification
Issue Confirmed Notifications	Boolean	(TRUE) if confirmed notifications are to be sent and (FALSE) if unconfirmed notifications are to be sent
Transitions	BACnet Event Transition Bits	A set of three flags that indicate those transitions {TO-OFFNORMAL, TO-FAULT, TO-NORMAL} for which this recipient is suitable

12.21.9 Profile_Name

This optional property, of type CharacterString, is the name of an object profile to which this object conforms. To ensure uniqueness, a profile name must begin with a vendor identifier code (see Clause 23) in base-10 integer format, followed by a dash. All subsequent characters are administered by the organization registered with that vendor identifier code. The vendor identifier code that prefixes the profile name shall indicate the organization that publishes and maintains the profile document named by the remainder of the profile name. This vendor identifier need not have any relationship to the vendor identifier of the device within which the object resides.

A profile defines a set of additional properties, behavior, and/or requirements for this object beyond those specified here. This standard defines only the format of the names of profiles. The definition of the profiles themselves is outside the scope of this standard.

12.22 Program Object Type

The Program object type defines a standardized object whose properties represent the externally visible characteristics of an application program. In this context, an application program is an abstract representation of a process within a BACnet Device, which is executing a particular body of instructions that act upon a particular collection of data structures. The logic that is embodied in these instructions and the form and content of these data structures are local matters.

The Program object provides a network-visible view of selected parameters of an application program in the form of properties of the Program object. Some of these properties are specified in the standard and exhibit a consistent behavior across different BACnet Devices. The operating state of the process that executes the application program may be viewed and controlled through these standardized properties, which are required for all Program objects. In addition to these standardized properties, a Program object may also provide vendor-specific properties. These vendor-specific properties may serve as inputs to the program, outputs from the program, or both. However, these vendor-specific properties may not be present at all. If any vendor-specific properties are present, the standard does not define what they are or how they work, as this is specific to the particular application program and vendor.

The Program object type and its standardized properties are summarized in Table 12-26 and described in detail in this subclause.

Table 12-26. Properties of the Program Object Type

Property Identifier	Property Datatype	Conformance Code
Object_Identifier	BACnetObjectIdentifier	R
Object_Name	CharacterString	R
Object_Type	BACnetObjectType	R
Program_State	BACnetProgramState	R
Program_Change	BACnetProgramRequest	W
Reason_For_Halt	BACnetProgramError	O[1]
Description_Of_Halt	CharacterString	O[1]
Program_Location	CharacterString	O
Description	CharacterString	O
Instance_Of	CharacterString	O
Status_Flags	BACnetStatusFlags	R
Reliability	BACnetReliability	O
Out_Of_Service	BOOLEAN	R
Profile_Name	CharacterString	O

[1] If one of the optional properties Reason_For_Halt or Description_Of_Halt is present, then both of these properties shall be present.

12.22.1 Object_Identifier

This property, of type BACnetObjectIdentifier, is a numeric code that is used to identify the object. It shall be unique within the BACnet Device that maintains it.

12.22.2 Object_Name

This property, of type CharacterString, shall represent a name for the object that is unique within the BACnet Device that maintains it. The minimum length of the string shall be one character. The set of characters used in the Object_Name shall be restricted to printable characters.

12.22.3 Object_Type

This property, of type BACnetObjectType, indicates membership in a particular object-type class. The value of this property shall be PROGRAM.

12.22.4 Program_State

This property, of type BACnetProgramState, reflects the current logical state of the process executing the application program this object represents. This property is Read-Only. The values that may be taken on by this property are:

IDLE	process is not executing
LOADING	application program being loaded
RUNNING	process is currently executing
WAITING	process is waiting for some external event
HALTED	process is halted because of some error condition
UNLOADING	process has been requested to terminate

12.22.5 Program_Change

This property, of type BACnetProgramRequest, is used to request changes to the operating state of the process this object represents. The Program_Change property provides one means for changing the operating state of this process. The process may change its own state as a consequence of execution as well. The values that may be taken on by this property are:

READY	ready for change request (the normal state)
LOAD	request that the application program be loaded, if not already loaded
RUN	request that the process begin executing, if not already running
HALT	request that the process halt execution
RESTART	request that the process restart at its initialization point
UNLOAD	request that the process halt execution and unload

Normally the value of the Program_Change property will be READY, meaning that the program is ready to accept a new request to change its operating state. If the Program_Change property is not READY, then it may not be written to and any attempt to write to the property shall return a Result(-). If it has one of the other enumerated values, then a previous request to change state has not yet been honored, so new requests cannot be accepted. When the request to change state is finally honored, then the Program_Change property value shall become READY and the new state shall be reflected in the Program_State property. Depending on the current Program_State, certain requested values for Program_Change may be invalid and would also return a Result(-) if an attempt were made to write them. Figure 12-3 shows the valid state transitions and the resulting new Program_State.

It is important to note that program loading could be terminated either due to an error or a request to HALT that occurs during loading. In either case, it is possible to have Program_State=HALTED and yet not have a complete or operable program in place. In this case, a request to RESTART is taken to mean LOAD instead. If a complete program is loaded but HALTED for any reason, then RESTART simply reenters program execution at its initialization entry point.

There may be BACnet Devices that support Program objects but do not require "loading" of the application programs, as these applications may be built in. In these cases, loading is taken to mean "prepare for execution," the specifics of which are a local matter.

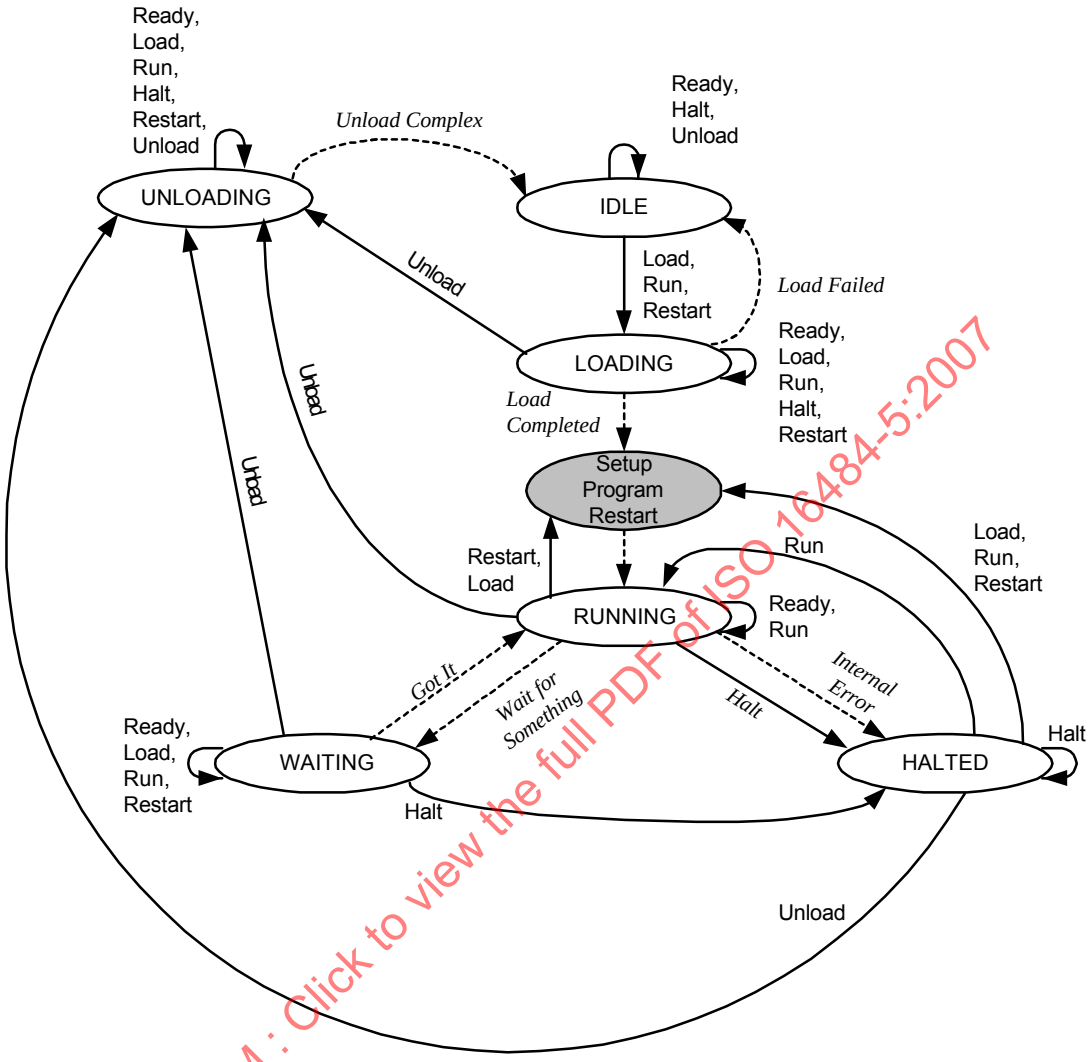


Figure 12-3. State Transitions for the program object.

12.22.6 Reason_For_Halt

If the process executing the application program this object represents encounters any type of error that causes process execution to be halted, then this property shall reflect the reason why the process was halted. The Reason_For_Halt property shall be an enumerated type called BACnetProgramError. The values that may be taken on by this property are:

NORMAL	process is not halted due to any error condition
LOAD_FAILED	the application program could not complete loading
INTERNAL	process is halted by some internal mechanism
PROGRAM	process is halted by Program_Change request
OTHER	process is halted for some other reason

If one of the optional properties Reason_For_Halt or Description_Of_Halt is present, then both of these properties shall be present.

12.22.7 Description_Of_Halt

This property is a character string that may be used to describe the reason why a program has been halted. This property provides essentially the same information as the Reason_For_Halt property, except in a human-readable form. The content of this string is a local matter. If one of the optional properties Reason_For_Halt or Description_Of_Halt is present, then both of these properties shall be present.

12.22.8 Program_Location

This property is a character string that may be used by the application program to indicate its location within the program code, for example, a line number or program label or section name. The content of this string is a local matter.

12.22.9 Description

This property is a string of printable characters that may be used to describe the application being carried out by this process or other locally desired descriptive information.

12.22.10 Instance_Of

This property is a character string that is the local name of the application program being executed by this process. The content of this string is a local matter.

12.22.11 Status_Flags

This property, of type BACnetStatusFlags, represents four Boolean flags that indicate the general "health" of the program. Three of the flags are associated with the values of other properties of this object. A more detailed status could be determined by reading the properties that are linked to these flags. The relationship between individual flags is not defined by the protocol. The four flags are

{IN_ALARM, FAULT, OVERRIDDEN, OUT_OF_SERVICE}

where:

IN_ALARM	The value of this flag shall be logical FALSE (0).
FAULT	Logical TRUE (1) if the Reliability property is present and does not have a value of NO_FAULT_DETECTED, otherwise logical FALSE (0).
OVERRIDDEN	Logical TRUE (1) if the program has been overridden by some mechanism local to the BACnet Device. In this context "overridden" is taken to mean that neither the Program_Change, Program_State nor any other program-specific property may be changed through BACnet services. Otherwise, the value is logical FALSE (0).
OUT_OF_SERVICE	Logical TRUE (1) if the Out_Of_Service property has a value of TRUE, otherwise logical FALSE (0).

12.22.12 Reliability

The Reliability property, of type BACnetReliability, provides an indication of whether the application-specific properties of the program object or the process executing the application program are "reliable" as far as the BACnet Device can determine and, if not, why. The Reliability property for this object type may have any of the following values:

{NO_FAULT_DETECTED, PROCESS_ERROR, UNRELIABLE_OTHER}.

12.22.13 Out_Of_Service

The Out_Of_Service property, of type BOOLEAN, is an indication whether (TRUE) or not (FALSE) the process this object represents is not in service. In this case, "in service" means that the application program is properly loaded and initialized, although the process may or may not be actually executing. If the Program_State property has the value IDLE, then Out_Of_Service shall be TRUE.

12.22.14 Profile_Name

This optional property, of type CharacterString, is the name of an object profile to which this object conforms. To ensure uniqueness, a profile name must begin with a vendor identifier code (see Clause 23) in base-10 integer format, followed by a dash. All subsequent characters are administered by the organization registered with that vendor identifier code. The vendor identifier code that prefixes the profile name shall indicate the organization that publishes and maintains the profile document named by the remainder of the profile name. This vendor identifier need not have any relationship to the vendor identifier of the device within which the object resides.

A profile defines a set of additional properties, behavior, and/or requirements for this object beyond those specified here. This standard defines only the format of the names of profiles. The definition of the profiles themselves is outside the scope of this standard.

12.23 Pulse Converter Object Type

The Pulse Converter object type defines a standardized object that represents a process whereby ongoing measurements made of some quantity, such as electric power or water or natural gas usage, and represented by pulses or counts, might be monitored over some time interval for applications such as peak load management, where it is necessary to make periodic measurements but where a precise accounting of every input pulse or count is not required.

The Pulse Converter object might represent a physical input. As an alternative, it might acquire the data from the Present_Value of an Accumulator object, representing an input in the same device as the Pulse Converter object. This linkage is illustrated by the dotted line in Figure 12-4. Every time the Present_Value property of the Accumulator object is incremented, the Count property of the Pulse Converter object is also incremented.

The Present_Value property of the Pulse Converter object can be adjusted at any time by writing to the Adjust_Value property, which causes the Count property to be adjusted, and the Present_Value recomputed from Count. In the illustration in Figure 12-4, the Count property of the Pulse Converter was adjusted down to 0 when the Total_Count of the Accumulator object had the value 0070.

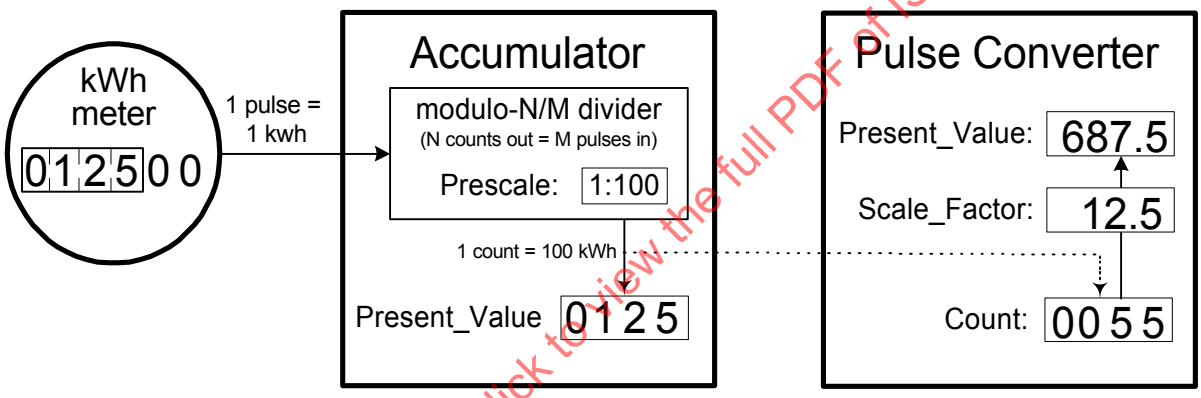


Figure 12-4. Relationship between the Pulse Converter and Accumulator objects.

The object and its properties are summarized in Table 12-27 and described in detail in this subclause.

Table 12-27. Properties of the Pulse Converter Object

Property Identifier	Property Datatype	Conformance Code
Object_Identifier	BACnetObjectIdentifier	R
Object_Name	CharacterString	R
Object_Type	BACnetObjectType	R
Description	CharacterString	O
Present_Value	REAL	R[1]
Input_Reference	BACnetObjectPropertyReference	O
Status_Flags	BACnetStatusFlags	R
Event_State	BACnetEventState	R
Reliability	BACnetReliability	O
Out_Of_Service	BOOLEAN	R
Units	BACnetEngineeringUnits	R
Scale_Factor	REAL	R
Adjust_Value	REAL	W
Count	Unsigned	R
Update_Time	BACnetDateTime	R
Count_Change_Time	BACnetDateTime	R[2]
Count_Before_Change	Unsigned	R[2]
COV_Increment	REAL	O[3]
COV_Period	Unsigned	O[3]
Notification_Class	Unsigned	O[4]
Time_Delay	Unsigned	O[4]
High_Limit	REAL	O[4]
Low_Limit	REAL	O[4]
Deadband	REAL	O[4]
Limit_Enable	BACnetLimitEnable	O[4]
Event_Enable	BACnetEventTransitionBits	O[4]
Acked_Transitions	BACnetEventTransitionBits	O[4]
Notify_Type	BACnetNotifyType	O[4]
Event_Time_Stamps	BACnetARRAY[3] of BACnetTimeStamp	O[4]
Profile_Name	CharacterString	O

[1] This property is required to be writable when Out_Of_Service is TRUE.
[2] These properties are required if Count_Before_Change is writable.
[3] These properties are required if the object supports COV reporting.
[4] These properties are required if the object supports intrinsic reporting.

12.23.1 Object_Identifier

This property, of type BACnetObjectIdentifier, is a numeric code that is used to identify the object. It shall be unique within the BACnet Device that maintains it.

12.23.2 Object_Name

This property, of type CharacterString, shall represent a name for the object that is unique within the BACnet Device that maintains it. The minimum length of the string shall be one character. The set of characters used in the Object_Name shall be restricted to printable characters.

12.23.3 Object_Type

This property, of type BACnetObjectType, indicates membership in a particular object type class. The value of this property shall be PULSE_CONVERTER.

12.23.4 Description

This property, of type CharacterString, is a string of printable characters whose content is not restricted.

12.23.5 Present_Value

This property, of type REAL, indicates the accumulated value of the input being measured. It is computed by multiplying the current value of the Count property by the value of the Scale_Factor property. The value of the Present_Value property may be adjusted by writing to the Adjust_Value property. The Present_Value property shall be writable when Out_Of_Service is TRUE.

12.23.6 Input_Reference

This optional property, of type BACnetObjectPropertyReference, indicates the object and property (typically an Accumulator object's Present_Value property) representing the actual physical input that is to be measured and presented by the Pulse Converter object. The referenced property should have a datatype of INTEGER or Unsigned.

If this property is not present, the Pulse Converter object directly represents the physical input.

12.23.7 Status_Flags

This property, of type BACnetStatusFlags, represents four Boolean flags that indicate the general "health" of a Pulse Converter. Three of the flags are associated with the values of other properties of this object. A more detailed status could be determined by reading the properties that are linked to these flags. The relationship between individual flags is not defined by the protocol. The four flags are

{IN_ALARM, FAULT, OVERRIDDEN, OUT_OF_SERVICE}

where:

IN_ALARM	Logical FALSE (0) if the Event_State property has a value of NORMAL, otherwise logical TRUE (1).
FAULT	Logical TRUE (1) if the Reliability property is present and does not have a value of NO_FAULT_DETECTED, otherwise logical FALSE (0).
OVERRIDDEN	Logical TRUE (1) if the program has been overridden by some mechanism local to the BACnet Device. In this context "overridden" is taken to mean that the Present_Value, Count and Reliability properties are no longer tracking changes to the input. Otherwise, the value is logical FALSE (0).
OUT_OF_SERVICE	Logical TRUE (1) if the Out_Of_Service property has a value of TRUE, otherwise logical FALSE (0).

12.23.8 Event_State

The Event_State property, of type BACnetEventState, is included in order to provide a way to determine if this object has an active event state associated with it. If the object supports intrinsic reporting, then the Event_State property shall indicate the event state of the object. If the object does not support intrinsic reporting and if the Reliability property is not present, then the value of this property shall be NORMAL. If the Reliability property is present and does not have a value of NO_FAULT_DETECTED, then the value of the Event_State property shall be FAULT. Changes in the Event_State property to the value FAULT are considered to be "fault" events.

12.23.9 Reliability

The Reliability property, of type BACnetReliability, provides an indication of whether the Present_Value and/or Count properties or the operation of the physical input in question is "reliable" as far as the BACnet Device or operator can determine and, if not, why. The Reliability property for this object type may have any of the following values:

{NO_FAULT_DETECTED, NO_SENSOR, OVER_RANGE, UNDER_RANGE, OPEN_LOOP, SHORTED_LOOP, UNRELIABLE_OTHER, CONFIGURATION_ERROR}

If Input_Reference is configured to reference a property that is not of datatype Unsigned or INTEGER, or is otherwise not supported as an input source for this object, the Reliability property shall indicate CONFIGURATION_ERROR.

12.23.10 Out_Of_Service

The Out_Of_Service property, of type BOOLEAN, is an indication whether (TRUE) or not (FALSE) the input that the object directly represents, if any, is not in service. ("Directly represents" means that the Input_Reference property is not present in this object.) The Present_Value property is decoupled from the Count property and will not track changes to the input when the value of Out_Of_Service is TRUE. In addition, the Reliability property and the corresponding state of the FAULT flag of the Status_Flags property shall be decoupled from the input when Out_Of_Service is TRUE. While the Out_Of_Service property is TRUE, the Present_Value and Reliability properties may be changed to any value as a means of simulating specific fixed conditions or for testing purposes. Other functions that depend on the state of the Present_Value or Reliability properties shall respond to changes made to these properties while Out_Of_Service is TRUE as if those changes had occurred in the input.

If the Input_Reference property is present, the state of the Out_Of_Service property of the object referenced by Input_Reference shall not be indicated by the Out_Of_Service property of the Pulse Converter object.

12.23.11 Units

This property, of type BACnetEngineeringUnits, indicates the measurement units of the Present_Value property. See the BACnetEngineeringUnits ASN.1 production in Clause 21 for a list of engineering units defined by this standard.

12.23.12 Scale_Factor

This property, of type REAL, provides the conversion factor for computing Present_Value. It represents the change in Present_Value resulting from changing the value of Count by one.

12.23.13 Adjust_Value

This property, of type REAL, is written to adjust the Present_Value property (and thus the Count property also) by the amount written to Adjust_Value.

If this property is writable the following series of operations shall be performed atomically when this property is written:

(1) The value written to Adjust_Value shall be stored in the Adjust_Value property.
(2) The value of Count shall be copied to the Count_Before_Change property.
(3) The value of Count shall be decremented by the value calculated by performing the integer division (Adjust_Value/Scale_Factor) and discarding the remainder.
(4) The current date and time shall be stored in the Count_Change_Time property.

A write to this property results in a change in the value of Present_Value. Whether the new value is computed as part of the atomic series of operations or when Present_Value is read is a local matter.

If Adjust_Value has never been written, it shall have a value of zero.

12.23.14 Count

This read-only property, of type Unsigned, indicates the count of the input pulses as acquired from the physical input or the property referenced by the Input_Reference property.

If the property referenced by Input_Reference property is present, has datatype Unsigned or INTEGER, and is supported as an input source for this object, the value of the Count property is derived from the referenced property. An increment by one

count in the referenced property is reflected by an increment of one count in the Count property. The means by which this is done shall be a local matter. Because the value of the Pulse Converter object Count property may be changed by a write to the Adjust_Value property, the value of the Count property can be different from the value of the referenced property.

12.23.15 Update_Time

This read-only property, of type BACnetDateTime, reflects the date and time of the most recent change to the Count property as a result of input pulse accumulation and is updated atomically with the Count property. If no such change has yet occurred, this property shall have wildcard values for all date and time fields.

12.23.16 Count_Change_Time

This read-only property, of type BACnetDateTime, represents the date and time of the most recent occurrence of a write to the Adjust_Value property. If no such write has yet occurred, this property shall have wildcard values for all date and time fields.

12.23.17 Count_Before_Change

This property, of type Unsigned, indicates the value of the Count property just prior to the most recent write to the Adjust_Value properties. If no such write has yet occurred, this property shall have the value zero.

12.23.18 COV_Increment

This property, of type REAL, shall specify the minimum change in Present_Value that will cause a COV notification to be issued to subscriber COV-clients. This property is required if COV reporting is supported by this object.

12.23.19 COV_Period

The COV_Period property, of type Unsigned, shall indicate the amount of time in seconds between the periodic COV notifications performed by this object. This property is required if COV reporting is supported by this object.

12.23.20 Notification_Class

This property, of type Unsigned, shall specify the notification class to be used when handling and generating event notifications for this object. The Notification_Class property implicitly refers to a Notification Class object that has a Notification_Class property with the same value. This property is required if intrinsic reporting is supported by this object.

12.23.21 Time_Delay

This property, of type Unsigned, shall specify the minimum period of time in seconds that the Present_Value must remain outside the band defined by the High_Limit and Low_Limit properties before a TO-OFFNORMAL event is generated or remain within the same band, including the Deadband property, before a TO-NORMAL event is generated. This property is required if intrinsic reporting is supported by this object.

12.23.22 High_Limit

This property, of type REAL, shall specify a limit that the Present_Value must exceed before an event is generated. This property is required if intrinsic reporting is supported by this object.

12.23.22.1 Conditions for Generating a TO-OFFNORMAL Event

A TO-OFFNORMAL event is generated under these conditions:

(a) the Present_Value must exceed the High_Limit for a minimum period of time, specified in the Time_Delay property, and
(b) the HighLimitEnable flag must be set in the Limit_Enable property, and
(c) the TO-OFFNORMAL flag must be set in the Event_Enable property.

12.23.22.2 Conditions for Generating a TO-NORMAL Event

Once exceeded, the Present_Value must fall below the High_Limit minus the Deadband before a TO-NORMAL event is generated under these conditions:

(a) the Present_Value must fall below the High_Limit minus the Deadband for a minimum period of time, specified in the Time_Delay property, and
(b) the HighLimitEnable flag must be set in the Limit_Enable property, and

(c) the TO-NORMAL flag must be set in the Event_Enable property.

12.23.23 Low_Limit

This property, of type REAL, shall specify a limit below which the Present_Value must fall before an event is generated. This property is required if intrinsic reporting is supported by this object.

12.23.23.1 Conditions for Generating a TO-OFFNORMAL Event

A TO-OFFNORMAL event is generated under these conditions:

(a) the Present_Value must fall below the Low_Limit for a minimum period of time, specified in the Time_Delay property, and
(b) the LowLimitEnable flag must be set in the Limit_Enable property, and
(c) the TO-OFFNORMAL flag must be set in the Event_Enable property.

12.23.23.2 Conditions for Generating a TO-NORMAL Event

Once the Present_Value has fallen below the Low_Limit, the Present_Value must exceed the Low_Limit plus the Deadband before a TO-NORMAL event is generated under these conditions:

(a) the Present_Value must exceed the Low_Limit plus the Deadband for a minimum period of time, specified in the Time_Delay property, and
(c) the LowLimitEnable flag must be set in the Limit_Enable property, and
(c) the TO-NORMAL flag must be set in the Event_Enable property.

12.23.24 Deadband

This property, of type REAL, shall specify a range between the High_Limit and Low_Limit properties, which the Present_Value must remain within for a TO-NORMAL event to be generated under these conditions:

(a) the Present_Value must fall below the High_Limit minus Deadband, and
(b) the Present_Value must exceed the Low_Limit plus the Deadband, and
(c) the Present_Value must remain within this range for a minimum period of time, specified in the Time_Delay property, and
(d) either the HighLimitEnable or LowLimitEnable flag must be set in the Limit_Enable property, and
(e) the TO-NORMAL flag must be set in the Event_Enable property

This property is required if intrinsic reporting is supported by this object.

12.23.25 Limit_Enable

This property, of type BACnetLimitEnable, shall convey two flags that separately enable and disable reporting of high limit and low limit offnormal events and their return to normal. This property is required if intrinsic reporting is supported by this object.

12.23.26 Event_Enable

This property, of type BACnetEventTransitionBits, shall convey three flags that separately enable and disable reporting of TO-OFFNORMAL, TO-FAULT, and TO-NORMAL events. In the context of Pulse Converter objects, transitions to the High_Limit or Low_Limit Event_States are considered to be "offnormal" events. This property is required if intrinsic reporting is supported by this object.

12.23.27 Acked_Transitions

This property, of type BACnetEventTransitionBits, shall convey three flags that separately indicate the receipt of acknowledgements for TO-OFFNORMAL, TO-FAULT, and TO-NORMAL events. In the context of Pulse Converter objects, transitions to High_Limit and Low_Limit Event_State are considered to be "offnormal" events. These flags shall be cleared upon the occurrence of the corresponding event and set under any of these conditions:

(a) upon receipt of the corresponding acknowledgement;
(b) upon the occurrence of the event if the corresponding flag is not set in the Event_Enable property (meaning event notifications will not be generated for this condition and thus no acknowledgement is expected);

(c) upon the occurrence of the event if the corresponding flag is set in the Event_Enable property and the corresponding flag in the Ack_Required property of the Notification Class object implicitly referenced by the Notification_Class property of this object is not set (meaning no acknowledgement is expected).

This property is required if intrinsic reporting is supported by this object.

12.23.28 Notify_Type

This property, of type BACnetNotifyType, shall convey whether the notifications generated by the object should be Events or Alarms. This property is required if intrinsic reporting is supported by this object.

12.23.29 Event_Time_Stamps

This optional property, of type BACnetARRAY[3] of BACnetTimeStamp, shall convey the times of the last event notifications for TO-OFFNORMAL, TO-FAULT, and TO-NORMAL events, respectively. Time stamps of type Time or Date shall have X'FF' in each octet and Sequence number time stamps shall have the value 0 if no event notification of that type has been generated since the object was created. This property is required if intrinsic reporting is supported by this object.

12.23.30 Profile_Name

This property, of type CharacterString, is the name of an object profile to which this object conforms. To ensure uniqueness, a profile name must begin with a vendor identifier code (see Clause 23) in base-10 integer format, followed by a dash. All subsequent characters are administered by the organization registered with that vendor identifier code. The vendor identifier code that prefixes the profile name shall indicate the organization that publishes and maintains the profile document named by the remainder of the profile name. This vendor identifier need not have any relationship to the vendor identifier of the device within which the object resides.

A profile defines a set of additional properties, behavior, and/or requirements for this object beyond those specified here. This standard defines only the format of the names of profiles. The definition of the profiles themselves is outside the scope of this standard.

12.24 Schedule Object Type

The Schedule object type defines a standardized object used to describe a periodic schedule that may recur during a range of dates, with optional exceptions at arbitrary times on arbitrary dates. The Schedule object also serves as a binding between these scheduled times and the writing of specified "values" to specific properties of specific objects at those times. The Schedule object type and its properties are summarized in Table 12-28 and described in detail in this subclause.

Schedules are divided into days, of which there are two types: normal days within a week and exception days. Both types of days can specify scheduling events for either the full day or portions of a day, and a priority mechanism defines which scheduled event is in control at any given time.

The current state of the Schedule object is represented by the value of its Present_Value property, which is normally calculated using the time/value pairs from the Weekly_Schedule and Exception_Schedule properties, with a default value for use when no schedules are in effect. Details of this calculation are provided in the description of the Present_Value property.

Versions of the Schedule object prior to Protocol_Revision 4 only support schedules that define an entire day, from midnight to midnight. For compatibility with these versions, this whole day behavior can be achieved by using a specific schedule format. Weekly_Schedule and Exception_Schedule values that begin at 00:00, and do not use any NULL values, will define schedules for the entire day. Property values in this format will produce the same results in all versions of the Schedule object.

Table 12-28. Properties of the Schedule Object Type

Property Identifier	Property Datatype	Conformance Code
Object_Identifier	BACnetObjectIdentifier	R
Object_Name	CharacterString	R
Object_Type	BACnetObjectType	R
Present_Value	Any	R
Description	CharacterString	O
Effective_Period	BACnetDateRange	R
Weekly_Schedule	BACnetARRAY[7]of BACnetDailySchedule	O[1]
Exception_Schedule	BACnetARRAY[N]of BACnetSpecialEvent	O[1]
Schedule_Default	Any	R
List_Of_Object_Property_References	List of BACnetDeviceObjectPropertyReference	R
Priority_For_Writing	Unsigned(1..16)	R
Status_Flags	BACnetStatusFlags	R
Reliability	BACnetReliability	R
Out_Of_Service	BOOLEAN	R
Profile_Name	CharacterString	O

[1] At least one of these properties is required.

12.24.1 Object_Identifier

This property, of type BACnetObjectIdentifier, is a numeric code that is used to identify the object. It shall be unique within the BACnet Device that maintains it.

12.24.2 Object_Name

This property, of type CharacterString, shall represent a name for the object that is unique within the BACnet Device that maintains it. The minimum length of the string shall be one character. The set of characters used in the Object_Name shall be restricted to printable characters.

12.24.3 Object_Type

This property, of type BACnetObjectType, indicates membership in a particular object-type class. The value of this property shall be SCHEDULE.

12.24.4 Present_Value

This property indicates the current value of the schedule, which may be any primitive datatype. As a result, most analog, binary, and enumerated values may be scheduled. This property shall be writable when Out_Of_Service is TRUE (see 12.24.14).

Any change in the value of this property shall be written to all members of the List_Of_Object_Property_References property. An error writing to any member of the list shall not stop the Schedule object from writing to the remaining members.

The normal calculation of the value of the Present_Value property is illustrated as follows (the actual algorithm used is a local matter but must yield the same results as this one):

1. Find the highest relative priority (as defined by 12.24.8) Exception_Schedule array element that is in effect for the current day and whose current value (see method below) is not NULL, and assign that value to the Present_Value property.
2. If the Present_Value was not assigned in the previous step, then evaluate the current value of the Weekly_Schedule array element for the current day and if that value is not NULL, assign it to the Present_Value property.
3. If the Present_Value was not assigned in the previous steps, then assign the value of the Schedule_Default property to the Present_Value property.

The method for evaluating the current value of a schedule (either exception or weekly) is to find the latest element in the list of BACnetTimeValues that occurs on or before the current time, and then use that element's value as the current value for the schedule. If no such element is found, then the current value for the schedule shall be NULL.

These calculations are such that they can be performed at any time and the correct value of Present_Value property will result. These calculations must be performed at 00:00 each day, whenever the device resets, whenever properties that can affect the results are changed, whenever the time in the device changes by an amount that may have an effect on the calculation result, and at other times, as required, to maintain the correct value of the Present_Value property through the normal passage of time.

Note that the Present_Value property will be assigned the value of the Schedule_Default property at 00:00 of any given day, unless there is an entry for 00:00 in effect for that day. If a scheduled event logically begins on one day and ends on another, an entry at 00:00 shall be placed in the schedule that is in effect for the second day, and for any subsequent days of the event's duration, to ensure the correct result whenever Present_Value is calculated

12.24.5 Description

This property is a string of printable characters whose content is not restricted.

12.24.6 Effective_Period

This property specifies the range of dates within which the Schedule object is active. Seasonal scheduling may be achieved by defining several SCHEDULE objects with non-overlapping Effective_Periods to control the same property references. Upon entering its effective period, the object shall calculate its Present_Value and write that value to all members of the List_Of_Object_Property_References property. An error writing to any member of the list shall not stop the Schedule object from writing to the remaining members.

12.24.7 Weekly_Schedule

This property is a BACnetARRAY containing exactly seven elements. Each of the elements 1-7 contains a BACnetDailySchedule. A BACnetDailySchedule consists of a list of BACnetTimeValues that are (time, value) pairs, which describe the sequence of schedule actions on one day of the week when no Exception_Schedule is in effect. The array elements 1-7 correspond to the days Monday - Sunday, respectively. The Weekly_Schedule is an optional property, but either the Weekly_Schedule or a non-empty Exception_Schedule shall be supported in every instance of a Schedule object.

If the Weekly_Schedule property is written with a schedule item containing a datatype not supported by this instance of the Schedule object (e.g., the List_Of_Object_Property_References property cannot be configured to reference a property of the unsupported datatype), the device may return a Result(-) response, specifying an 'Error Class' of PROPERTY and an 'Error Code' of DATATYPE_NOT_SUPPORTED.

12.24.8 Exception_Schedule

This property is a BACnetARRAY of BACnetSpecialEvents. Each BACnetSpecialEvent describes a sequence of schedule actions that takes precedence over the normal day's behavior on a specific day or days.

BACnetSpecialEvent ::= (Period, List of BACnetTimeValue, EventPriority)

Period ::= Choice of {BACnetCalendarEntry | CalendarReference}

EventPriority ::= Unsigned (1..16)

The Period may be a BACnetCalendarEntry or it may refer to a Calendar object. A BACnetCalendarEntry would be used if the Exception_Schedule is specific to this Schedule object, while calendars might be defined for common holidays to be referenced by multiple Schedule objects. Each BACnetCalendarEntry is either an individual date (Date), range of dates (BACnetDateRange), or month/week-of-month/day-of-week specification (BACnetWeekNDay). If the current date matches any of the calendar entry criteria, the Exception Schedule would be activated and the list of BACnetTimeValues would be enabled for use.

Individual fields of the various constructs of the BACnetCalendarEntry may also have a "wildcard" value used for determining if the current date falls within the Period of the Exception Schedule. In a date range, for example, if the startDate is a wildcard, it means "any date up to and including the endDate." If the endDate is a wildcard, it means "any date from the startDate on." If the calendar entry were a BACnetWeekNDay with wildcard for month and week-of-month fields but with a specific day-of-week, it would mean the Exception Schedule would apply on that day-of-week all year long.

Each BACnetSpecialEvent contains an EventPriority that determines its importance relative to other BACnetSpecialEvents within the same Exception_Schedule. Since SpecialEvents within the same Exception_Schedule may have overlapping periods, it is necessary to have a mechanism to determine the relative priorities for the SpecialEvents that apply on any given day. If more than one SpecialEvent applies to a given day, the relative priority of the SpecialEvents shall be determined by their EventPriority values. If multiple overlapping SpecialEvents have the same EventPriority value, then the SpecialEvent with the lowest index number in the array shall have higher relative priority. The highest EventPriority is 1 and the lowest is 16. The EventPriority is not related to the Priority_For_Writing property of the Schedule object.

If a BACnet Device supports writing to the Exception_Schedule property, all possible choices in the BACnetSpecialEvents shall be supported. If the size of this array is increased by writing to array index zero, each new array element shall contain an empty List of BACnetTimeValue.

If the Exception_Schedule property is written with a schedule item containing a datatype not supported by this instance of the Schedule object (e.g., the List_Of_Object_Property_References property cannot be configured to reference a property of the unsupported datatype), the device may return a Result(-) response, specifying an 'Error Class' of PROPERTY and an 'Error Code' of DATATYPE_NOT_SUPPORTED.

12.24.9 Schedule_Default

This property holds a default value to be used for the Present_Value property when no other scheduled value is in effect (see 12.24.4). This may be any primitive datatype.

If the Schedule_Default property is written with a value containing a datatype not supported by this instance of the Schedule object (e.g., the List_Of_Object_Property_References property cannot be configured to reference a property of the unsupported datatype), the device may return a Result(-) response, specifying an 'Error Class' of PROPERTY and an 'Error Code' of DATATYPE_NOT_SUPPORTED.

12.24.10 List_Of_Object_Property_Reference

This property specifies the Device Identifiers, Object Identifiers and Property Identifiers of the properties to be written with specific values at specific times on specific days.

If this property is writable, it may be restricted to only support references to objects inside of the device containing the Schedule object. If the property is restricted to referencing objects within the containing device, an attempt to write a

reference to an object outside the containing device into this property shall cause a Result(-) to be returned with an error class of PROPERTY and an error code of OPTIONAL_FUNCTIONALITY_NOT_SUPPORTED.

If this property is set to reference an object outside the device containing the Schedule object, the method used for writing to the referenced property value for the purpose of controlling the property is a local matter. The only restriction on the method of writing to the referenced property is that the scheduling device be capable of using WriteProperty for this purpose so as to be interoperable with all BACnet devices.

12.24.11 Priority_For_Writing

This property defines the priority at which the referenced properties are commanded. It corresponds to the 'Priority' parameter of the WriteProperty service. It is an unsigned integer in the range 1-16, with 1 being considered the highest priority and 16 the lowest. See Clause 19.

12.24.12 Status_Flags

This property, of type BACnetStatusFlags, represents four Boolean flags that indicate the general "health" of the schedule object. Two of the flags are associated with the values of other properties of this object. A more detailed status could be determined by reading the properties that are linked to these flags. The relationship between individual flags is not defined by the protocol. The four flags are

{IN_ALARM, FAULT, OVERRIDDEN, OUT_OF_SERVICE}

where:

IN_ALARM	The value of this flag shall be logical FALSE (0).
FAULT	Logical TRUE (1) if the Reliability property does not have a value of NO_FAULT_DETECTED, otherwise logical FALSE (0).
OVERRIDDEN	Logical TRUE (1) if the schedule object has been overridden by some mechanism local to the BACnet Device. In this context "overridden" is taken to mean that the Present_Value property is not changeable through BACnet services. Otherwise, the value is logical FALSE (0).
OUT_OF_SERVICE	Logical TRUE (1) if the Out_Of_Service property has a value of TRUE, otherwise logical FALSE (0).

12.24.13 Reliability

The Reliability property, of type BACnetReliability, provides an indication that the properties of the schedule object are in a consistent state. All non-NULL values used in the Weekly_Schedule, the Exception_Schedule, and the Schedule_Default properties shall be of the same datatype, and all members of the List_Of_Object_Property_References shall be writable with that datatype. If these conditions are not met, then this property shall have the value CONFIGURATION_ERROR. The Reliability property for this object type may have any of the following values:

{NO_FAULT_DETECTED, CONFIGURATION_ERROR, UNRELIABLE_OTHER}.

If the List_Of_Object_Property_References contains a member that references a property in a remote device, the detection of a configuration error may be delayed until an attempt is made to write a scheduled value.

12.24.14 Out_Of_Service

The Out_Of_Service property, of type BOOLEAN, is an indication whether (TRUE) or not (FALSE) the internal calculations of the schedule object are used to determine the value of the Present_Value property. This means that the Present_Value property is decoupled from the internal calculations and will not track changes to other properties when Out_Of_Service is TRUE. Other functions that depend on the state of the Present_Value, such as writing to the members of the List_Of_Object_Property_References, shall respond to changes made to that property while Out_Of_Service is TRUE, as if those changes had occurred by internal calculations.

12.24.15 Profile_Name

This optional property, of type CharacterString, is the name of an object profile to which this object conforms. To ensure uniqueness, a profile name must begin with a vendor identifier code (see Clause 23) in base-10 integer format, followed by a dash. All subsequent characters are administered by the organization registered with that vendor identifier code. The vendor identifier code that prefixes the profile name shall indicate the organization that publishes and maintains the profile document named by the remainder of the profile name. This vendor identifier need not have any relationship to the vendor identifier of the device within which the object resides.

A profile defines a set of additional properties, behavior, and/or requirements for this object beyond those specified here. This standard defines only the format of the names of profiles. The definition of the profiles themselves is outside the scope of this standard.

12.25 Trend Log Object Type

A Trend Log object monitors a property of a referenced object and, when predefined conditions are met, saves ("logs") the value of the property and a timestamp in an internal buffer for subsequent retrieval. The data may be logged periodically or upon a change of value. Errors that prevent the acquisition of the data, as well as changes in the status or operation of the logging process itself, are also recorded. Each timestamped buffer entry is called a trend log "record."

The referenced object may reside in the same device as the Trend Log object or in an external device. The referenced property's value may be recorded upon COV subscription or periodic poll. Where status flags are available (such as when the COVNotification or ReadPropertyMultiple services are used), they are also acquired and saved with the data.

Each Trend Log object maintains an internal, optionally fixed-size buffer. This buffer fills or grows as log records are added. If the buffer becomes full, the least recent record is overwritten when a new record is added, or collection may be set to stop. Trend Log records are transferred as BACnetLogRecords using the ReadRange service. The buffer may be cleared by writing a zero to the Record_Count property. Each record in the buffer has an implied SequenceNumber which is equal to the value the Total_Record_Count property has immediately after the record is added. If the Total_Record_Count is incremented past 2^{32}-1, then it shall reset to 1.

Several datatypes are defined for storage in the log records. The ability to store ANY datatypes is optional. Data stored in the log buffer may be optionally restricted in size to 32 bits, as in the case of bit strings, to facilitate implementation in devices with strict storage requirements.

Logging may be enabled and disabled through the Log_Enable property and at dates and times specified by the Start_Time and Stop_Time properties. Trend Log enabling and disabling is recorded in the log buffer.

Event reporting (notification) may be provided to facilitate automatic fetching of log records by processes on other devices such as fileservers. Support is provided for algorithmic reporting; optionally, intrinsic reporting may be provided.

In intrinsic reporting, when the number of records specified by the Notification_Threshold property have been collected since the previous notification (or startup), a new notification is sent to all subscribed devices. BUFFER_READY algorithmic reporting is described in Clause 13.3.7.

In response to a notification, subscribers may fetch all of the new records. If a subscriber needs to fetch all of the new records, it should use the 'By Sequence Number' form of the ReadRange service request.

A missed notification may be detected by a subscriber if the Current_Notify_Record it received in its previous notification is different than the Previous_Notify_Record parameter of the current notification. If the ReadRange-ACK response to the ReadRange request issued under these conditions has its 'first-item' flag set to TRUE, Trend Log records have probably been missed by this subscriber.

The acquisition of log records by remote devices has no effect upon the state of the Trend Log object itself. This allows completely independent, but properly sequential, access to its log records by all remote devices. Any remote device can independently update its records at any time.

Table 12-29. Properties of the Trend Log Object Type

Property Identifier	Property Datatype	Conformance Code
Object_Identifier	BACnetObjectIdentifier	R
Object_Name	CharacterString	R
Object_Type	BACnetObjectType	R
Description	CharacterString	O
Log_Enable	BOOLEAN	W
Start_Time	BACnetDateTime	O[1,2]
Stop_Time	BACnetDateTime	O[1,2]
Log_DeviceObjectProperty	BACnetDeviceObjectPropertyReference	O[1]
Log_Interval	Unsigned	O[1,2]
COV_Resubscription_Interval	Unsigned	O
Client_COV_Increment	BACnetClientCOV	O
Stop_When_Full	BOOLEAN	R
Buffer_Size	Unsigned32	R
Log_Buffer	List of BACnetLogRecord	R
Record_Count	Unsigned32	W
Total_Record_Count	Unsigned32	R
Notification_Threshold	Unsigned32	O[3]
Records_Since_Notification	Unsigned32	O[3]
Last_Notify_Record	Unsigned32	O[3]
Event_State	BACnetEventState	R
Notification_Class	Unsigned	O[3]
Event_Enable	BACnetEventTransitionBits	O[3]
Acked_Transitions	BACnetEventTransitionBits	O[3]
Notify_Type	BACnetNotifyType	O[3]
Event_Time_Stamps	BACnetARRAY[3] of BACnetTimeStamp	O[3]
Profile_Name	CharacterString	O

[1] These properties are required to be present if the monitored property is a BACnet property.
[2] If present, these properties are required to be writable.
[3] These properties are required to be present if the object supports intrinsic reporting.

12.25.1 Object_Identifier

This property, of type BACnetObjectIdentifier, is a numeric code that is used to identify the object. It shall be unique within the BACnet Device that maintains it.

12.25.2 Object_Name

This property, of type CharacterString, shall represent a name for the Object that is unique within the BACnet Device that maintains it. The minimum length of the string shall be one character. The set of characters used in the Object_Name shall be restricted to printable characters.

12.25.3 Object_Type

This property, of type BACnetObjectType, indicates membership in a particular object type class. The value of this property shall be TREND_LOG.

12.25.4 Description

This property, of type CharacterString, is a string of printable characters whose content is not restricted.

12.25.5 Log_Enable

This property, of type BOOLEAN, indicates and controls whether (TRUE) or not (FALSE) logging is enabled. A value of FALSE overrides the time interval defined by Start_Time and Stop_Time. If logging is otherwise enabled by the Start_Time and Stop_Time properties, changes to the value of the Log_Enable property shall be recorded in the log. When the device begins operation the value TRUE shall be recorded in the log.

12.25.6 Start_Time

This property, of type BACnetDateTime, specifies the date and time at or after which logging shall be enabled by this property. If any of the fields of the BACnetDateTime contain "wildcard" values, then the conditions for logging to be enabled by Start_Time shall be ignored. If Start_Time specifies a date and time after Stop_Time, then logging shall be disabled. This property must be writable if present.

12.25.7 Stop_Time

This property, of type BACnetDateTime, specifies the date and time at or after which logging shall be disabled by this property. If any of the fields of the BACnetDateTime contain "wildcard" values, then the conditions for logging to be enabled by Stop_Time shall be ignored. If Stop_Time specifies a date and time earlier than Start_Time, then logging shall be disabled. This property must be writable if present.

12.25.8 Log_DeviceObjectProperty

This property, of type BACnetDeviceObjectPropertyReference, specifies the Device Identifier, Object Identifier and Property Identifier of the property to be trend logged.

If this property is writable, it may be restricted to reference only objects inside the device containing the Trend Log object. If the property is restricted to referencing objects within the containing device, an attempt to write a reference to an object outside the containing device into this property shall cause a Result(-) to be returned.

12.25.9 Log_Interval

This property, of type Unsigned, specifies the periodic interval in hundredths of seconds for which the referenced property is to be logged. If this property has the value zero then the Trend Log shall issue COV subscriptions for the referenced property. The value of this property must be non-zero if COV_Resubscription_Interval is not present. This property must be writable if present.

12.25.10 COV_Resubscription_Interval

If the Trend Log is acquiring data from a remote device by COV subscription, this property, of type Unsigned, specifies the number of seconds between COV resubscriptions, provided that COV subscription is in effect. SubscribeCOV requests shall specify twice this lifetime for the subscription and shall specify the issuance of confirmed notifications. If COV subscriptions are in effect, the first COV subscription is issued when the Trend Log object begins operation or when Log_Enable becomes TRUE. If present, the value of this property must be non-zero. If this property is not present, then COV subscription shall not be attempted.

12.25.11 Client_COV_Increment

If the Trend Log is acquiring COV data, this property, of type BACnetClientCOV, specifies the increment to be used in determining that a change of value has occurred. If the referenced object and property supports COV reporting according to 13.1, this property may have the value NULL; in this case change of value is determined by the criteria of 13.1.

12.25.12 Stop_When_Full

This property, of type BOOLEAN, specifies whether (TRUE) or not (FALSE) logging should cease when the buffer is full. When logging ceases, Log_Enable shall be set FALSE.

12.25.13 Buffer_Size

This property, of type Unsigned32, shall specify the maximum number of records the buffer may hold. If writable, it may not be written when Log_Enable is TRUE. The disposition of existing records when Buffer_Size is written is a local matter.

12.25.14 Log_Buffer

This property is a list of up to Buffer_Size timestamped records of datatype BACnetLogRecord, each of which conveys a recorded data value, an error related to data-collection, or status changes in the Trend Log object. Each record has data fields as follows:

Timestamp The local date and time when the record was collected.

LogDatum The data value read from the monitored object and property, an error encountered in an attempt to read a value, or a change in status or operation of the Trend Log object itself.

StatusFlags The Status_Flags property of the monitored object, if present and available atomically associated with the LogDatum data value. If the Status_Flags property is not present or not available atomically associated with the data value, this item shall not be included in the log record.

The choices available for the LogDatum are listed below:

log-status This choice represents a change in the status or operation of the Trend Log object. Whenever one of the events represented by the flags listed below occurs, except as noted, a record shall be appended to the buffer.

- log-disabled This flag is set whenever the Trend Log object is disabled, such as when Log_Enable is set to FALSE. Whenever the Trend Log object begins operation, this flag shall be presumed to have changed from TRUE to FALSE and a log entry shall be made.
- buffer-purged This flag shall be set to TRUE whenever the buffer is deleted by a write of the value zero to the Record_Count property. After this value is recorded in the buffer, the subsequent immediate change to FALSE shall not be recorded.

boolean-value
real-value
enum-value
unsigned-value
signed-value
bitstring-value
null-value
These choices represent data values read from the monitored object and property.

failure This choice represents an error encountered in an attempt to read a data value from the monitored object. If the error is conveyed by an error response from a remote device the Error Class and Error Code in the response shall be recorded.

time-change This choice represents a change in the clock setting in the device, it records the number of seconds by which the clock changed. If the number is not known, such as when the clock is initialized for the first time, the value recorded shall be zero.

any-value This choice represents data values read from the monitored object and property.

Also associated with each record is an implied record number, the value of which is equal to Total_Record_Count at the point where the record has been added into the Log Buffer and Total_Record_Count has been adjusted accordingly. All clients must be able to correctly handle the case where the Trend Log is reset such that its Total_Record_Count is returned to zero and also the case where Total_Record_Count has wrapped back to 1.

The buffer is not network accessible except through the use of the ReadRange service, in order to avoid problems with record sequencing when segmentation is required.

12.25.15 Record_Count

This property, of type Unsigned32, shall represent the number of records currently resident in the log buffer. A write of the value zero to this property shall cause all records in the log buffer to be deleted and Records_Since_Notification to be reset to zero. Upon completion, this event shall be reported in the log as the initial entry.

12.25.16 Total_Record_Count

This property, of type Unsigned32, shall represent the total number of records collected by the Trend Log object since creation. When the value of Total_Record_Count reaches its maximum possible value of $2^{32} - 1$, the next value it takes shall be one. Once this value has wrapped to one, its semantic value (the total number of records collected) has been lost but its use in generating notifications remains.

12.25.17 Notification_Threshold

This property, of type Unsigned32, shall specify the value of Records_Since_Notification at which notification occurs. This property is required if intrinsic reporting is supported by this object.

12.25.18 Records_Since_Notification

This property, of type Unsigned32, represents the number of records collected since the previous notification, or since the beginning of logging if no previous notification has occurred. This property is required if intrinsic reporting is supported by this object.

12.25.19 Last_Notify_Record

This property, of type Unsigned32, represents the SequenceNumber associated with the most recently collected record whose collection triggered a notification. If no notification has occurred since logging began the value of this property shall be zero. This property is required if intrinsic reporting is supported by this object.

12.25.20 Event_State

The Event_State property, of type BACnetEventState, is included in order to provide a way to determine if this object has an active event state associated with it. If the object supports intrinsic reporting, then the Event_State property shall indicate the event state of the object. If the object does not support intrinsic reporting, then the value of this property shall be NORMAL. The allowed states are NORMAL, and FAULT.

12.25.21 Notification_Class

This property, of type Unsigned, shall specify the notification class to be used when handling and generating event notifications for this object. The Notification_Class property implicitly refers to a Notification Class object that has a Notification_Class property with the same value. This property is required if intrinsic reporting is supported by this object.

12.25.22 Event_Enable

This property, of type BACnetEventTransitionBits, shall convey three flags that separately enable and disable reporting of TO-FAULT and TO-NORMAL events. In the context of Trend Log objects, the value of the Records_Since_Notification property becoming equal to or greater than the value of the Notification_Threshold property shall cause a TO-NORMAL transition. The failure of an attempted COV subscription shall cause a TO-FAULT state transition. The TO-NORMAL transition must be enabled when intrinsic reporting is to be used; this shall be set by default. This property is required if intrinsic reporting is supported by this object.

12.25.23 Acked_Transitions

This property, of type BACnetEventTransitionBits, shall convey three flags that separately indicate the receipt of acknowledgments for TO-OFFNORMAL, TO-FAULT and TO-NORMAL events. These flags shall be cleared upon the occurrence of the corresponding event and set under any of these conditions:

(a) upon receipt of the corresponding acknowledgment;
(b) upon the occurrence of the event if the corresponding flag is not set in the Event_Enable property (meaning event notifications will not be generated for this condition and thus no acknowledgment is expected);
(c) upon the occurrence of the event if the corresponding flag is set in the Event_Enable property and the corresponding flag in the Ack_Required property of the Notification Class object implicitly referenced by the Notification_Class property of this object is not set (meaning no acknowledgment is expected).

This property is required if intrinsic reporting is supported by this object.

12.25.24 Notify_Type

This property, of type BACnetNotifyType, shall convey whether the notifications generated by the object should be Events or Alarms. This property is required if intrinsic reporting is supported by this object.

12.25.25 Event_Time_Stamps

This optional property, of type BACnetARRAY [3] of BACnetTimeStamp, shall convey the times of the last event notifications for TO-OFFNORMAL, TO-FAULT, and TO-NORMAL events, respectively. Time stamps of type Time or Date shall have 'FF' in each octet and Sequence number time stamps shall have the value 0 if no event notification of that type has been generated since the object was created. This property is required if intrinsic reporting is supported by this object.

12.25.26 Profile_Name

This optional property, of type CharacterString, is the name of an object profile to which this object conforms. To ensure uniqueness, a profile name must begin with a vendor identifier code (see Clause 23) in base-10 integer format, followed by a dash. All subsequent characters are administered by the organization registered with that vendor identifier code. The vendor identifier code that prefixes the profile name shall indicate the organization that publishes and maintains the profile document named by the remainder of the profile name. This vendor identifier need not have any relationship to the vendor identifier of the device within which the object resides.

A profile defines a set of additional properties, behavior, and/or requirements for this object beyond those specified here. This standard defines only the format of the names of profiles. The definition of the profiles themselves is outside the scope of this standard.

13 ALARM AND EVENT SERVICES

This clause describes the conceptual approach and application services used in BACnet to manage communication related to events. Object types relating to event management are defined in Clause 12. In general, "events" are changes of value of certain properties of certain objects, or internal status changes, that meet predetermined criteria. There are three mechanisms provided in BACnet for managing events: change of value reporting, intrinsic reporting, and algorithmic change reporting.

Change of value (COV) reporting allows a COV-client to subscribe with a COV-server, on a permanent or temporary basis, to receive reports of some changes of value of some referenced property based on fixed criteria. Certain BACnet standard objects may optionally support COV reporting. If a standard object provides COV reporting, then changes of value of specific properties of the object, in some cases based on programmable increments, trigger COV notifications to be sent to one or more subscriber clients. Typically, COV notifications are sent to supervisory programs in COV-client devices or to operators or logging devices. Proprietary objects may support COV reporting at the implementor's option.

Intrinsic reporting allows a BACnet device to provide one or more event sources, intrinsic to the device, that generate event notifications that may be directed to one or more destinations. Certain BACnet standard objects may optionally support intrinsic reporting by supporting optional properties that define the type of event to be generated and options for handling and routing of the notifications. Internal status changes and alarms may also use intrinsic reporting to generate diagnostic notifications. Proprietary objects may support intrinsic reporting at the implementor's option.

Algorithmic change reporting allows a notification-client to subscribe with a notification-server to receive reports of changes of value of any property of any object on the basis of predetermined, and network-visible, criteria. Any of the standardized algorithms described in Clause 13.3 may be used to establish criteria for change reporting. Once established, occurrences of change may be reported to one or more destinations based on further criteria. Algorithmic change reporting differs from intrinsic reporting in that Event Enrollment objects are used to determine the event condition(s).

Intrinsic and algorithmic change reporting rely on the concept of event classification for selective reporting of individual occurrences of events. Particular intrinsic or algorithmic change events may specify a notification class number that is directly related to a Notification Class object within the initiating device. The Notification Class object is used to specify the handling and routing of events to one or more destinations. The Notification Class object defines the priorities to be used in event-notification messages, whether acknowledgment by an application process or human operator is required, and at what time periods during the week given destinations are to be used.

BACnet devices that support event notification are free to define any number of unique conditions that trigger alarm or event notifications. Alarms and events are broadly classified into one of three possible groups: fault, offnormal, and normal. A "fault" condition is a malfunction, nearly always representing a failure within the automation system itself. An "offnormal" condition is a condition within the system that is not normally expected or is outside the bounds of ideal operation. A "normal" condition is anything else.

Event-initiating objects may identify their "state" from moment to moment as one of any number of possibly unique event states. Notifications are triggered by the "transition" of conditions for an object, usually from one unique event state to another. A transition to a particular event state may be used to identify specific unique handling for the notification(s) generated by an object, for example, controlling the destination for a notification or whether the particular transition event should require acknowledgment. In these contexts, all transitions that result in a state that is not normal, and not a fault, are considered to be TO-OFFNORMAL transitions. Transitions to any fault state are considered to be TO-FAULT transitions. All other transitions are, by definition, TO-NORMAL transitions.

Events may be selectively identified as belonging to the category of "alarms" or "events." Event-initiating objects indicate this distinction through the Notify_Type property. In BACnet, the singular distinction between alarms and all other events is whether the event will be reported by the GetAlarmSummary service or not. Alarms will be reported by the GetAlarmSummary service, while all other events will not be reported by GetAlarmSummary. In every other respect, BACnet makes no distinction between an alarm and an event.

None of the reporting mechanisms is preferred, as each addresses particular needs generally found in building automation and control systems. A given BACnet device may use any or all of these mechanisms to provide alarm and event management

functions. However, each mechanism dictates a standardized complement of services and/or objects that are used to realize their functions.

13.1 Change of Value Reporting

Change of value (COV) reporting allows a COV-client to subscribe with a COV-server, on a permanent or temporary basis, to receive reports of some changes of value of some referenced property based on fixed criteria. If an object provides COV reporting, then changes of value of any subscribed-to properties of the object, in some cases based on programmable increments, trigger COV notifications to be sent to subscribing clients. Typically, COV notifications are sent to supervisory programs in COV-client devices or to operators or logging devices. Any object, proprietary or standard, may support COV reporting at the implementor's option.

COV subscriptions are established using the SubscribeCOV service or the SubscribeCOVProperty service. The subscription establishes a connection between the change of value detection and reporting mechanism within the COV-server device and a "process" within the COV-client device. Notifications of changes are issued by the COV-server when changes occur after the subscription has been established. The ConfirmedCOVNotification and UnconfirmedCOVNotification services are used by the COV-server to convey change notifications. The choice of confirmed or unconfirmed service is specified in the subscription.

When a BACnet standard object, of a type listed in Table 13-1, supports COV reporting it shall support COV reporting for the property as listed in Table 13-1. At the implementor's discretion, COV reporting may also be supported for any other property of the object. For properties listed in Table 13-1 that have a REAL datatype, the COV increment used to determine when to generate notifications will be the COV_Increment property of the object unless a COV_Increment parameter is supplied in the SubscribeCOVProperty service. For other properties that have a REAL datatype, the COV increment to use when not supplied with the SubscribeCOVProperty service shall be a local matter. This is to allow multiple subscribers that do not require a specific increment to use a common increment to allow for the reduction of the processing burden on the COV-server. The criteria for COV reporting for properties other than those listed in Table 13-1 is based on the datatype of the property subscribed to and is described in Table 13-1a.

If an object supports the COV_Period property and COV_Period is non-zero, it shall issue COV notifications to all subscribed recipients at the regular interval specified by COV_Period, in addition to the notifications initiated by the change of value of the monitored property. The value of the monitored property conveyed by the periodic COV notification shall be the basis for determining whether a subsequent COV notification is required by the change in value of the monitored property. If COV_Period is zero, the periodic notifications shall not be issued.

It is the responsibility of the COV-server to maintain the list of active subscriptions for each object that supports COV notification. This list of subscriptions shall be capable of holding at least a single subscription for each object that supports COV notification, although multiple subscriptions may be supported at the implementor's option. The list of subscriptions is network-visible through the device object's Active_COV_Subscriptions property. Subscriptions may be created with finite lifetimes, meaning that the subscription may lapse and be automatically canceled after a period of time. Optionally, the lifetime may be specified as infinite, meaning that no automatic cancellation occurs. However, the COV-server is not required to guarantee preservation of subscriptions across power failures or "restarts." Periodic resubscription is allowed and expected and shall simply succeed as if the subscription were new, extending the lifetime of the subscription.

The different standard objects that support standardized COV reporting use different criteria for determining that a "change of value" has occurred, which are summarized in Table 13-1. Proprietary object types, or other standard object types not listed in Table 13-1, that support COV reporting of the Present_Value property, should follow these criteria whenever possible. Any objects that may optionally provide COV support and the change of value algorithms they shall employ are summarized in Tables 13-1 and 13-1a.

Table 13-1. Standardized Objects That May Support COV Reporting

Object Type	Criteria	Properties Reported
Analog Input, Analog Output, Analog Value	If Present_Value changes by COV_Increment or Status_Flags changes at all	Present_Value, Status_Flags
Binary Input, Binary Output, Binary Value, Life Safety Point, Life Safety Zone, Multi-state Input, Multi-state Output, Multi-state Value	If Present_Value changes at all or Status_Flags changes at all	Present_Value, Status_Flags
Loop	If Present_Value changes by COV_Increment or Status_Flags changes at all	Present_Value, Status_Flags, Setpoint, Controlled_Variable_Value
Pulse Converter	If Present_Value changes by COV_Increment or Status_Flags changes at all or If COV_Period expires	Present_Value, Status_Flags, Update_Time

Table 13-1a. Criteria Used for COV Reporting of Properties Other Than Those Listed in Table 13-1.

Datatype	Criteria	Properties Reported
REAL	If the property changes by the increment (from the service if provided; otherwise, as determined by the device) or Status_Flags changes at all (if the object has a Status_Flags property)	The subscribed-to property, Status_Flags (if the object has a Status_Flags property)
All other datatypes	If the property changes at all or Status_Flags changes at all (if the object has a Status_Flags property)	The subscribed-to property, Status_Flags (if the object has a Status_Flags property)

13.2 Intrinsic Reporting

Intrinsic reporting allows a BACnet device to provide one or more alarm or event sources, intrinsic to the device, which generate alarm or event notifications that may be directed to one or more destinations. Certain BACnet standard objects may optionally support intrinsic reporting by providing optional properties for defining the type of alarm or event to be generated and options for handling and routing of the notifications. Internal status changes and alarms may also use intrinsic reporting to generate diagnostic notifications. Proprietary objects may support intrinsic reporting at the implementor's option. If a standard object provides intrinsic reporting, then changes of value of specific properties of the object, in some cases based on programmable criteria, or changes of status internal to the object trigger event notifications to be sent to one or more destinations based on notification class. Typically, event notifications are sent to operators or logging devices represented by "processes" within a notification-client device. The ConfirmedEventNotification and UnconfirmedEventNotification services are used by the notification-server to convey notifications.

The connection between the event detection and reporting mechanism within the device, which initiates the event notification, and the "processes" within one or more notification-client devices is made indirectly through a Notification Class object in the initiating device. The Notification Class object may be configured statically or dynamically in those devices that support the dynamic modification or creation of Notification Class objects. Multiple alarm or event-initiating objects may reference the same Notification Class object in a device. The notification class provides information to control the handling of the notification in the following ways:

(a) by providing the priority to be used in the event notification, based on whether the event transition is TO-OFFNORMAL, TO-FAULT, or TO-NORMAL;
(b) by specifying whether the event requires acknowledgment, based on whether the event transition is TO-OFFNORMAL, TO-FAULT, or TO-NORMAL;
(c) by providing a specific destination for the event notification that is a (process, recipient) tuple, based on time of day, day of week, and type of event transition (TO-OFFNORMAL, TO-FAULT, or TO-NORMAL);
(d) by providing an indication of whether confirmed or unconfirmed notification is to be used for a given destination.

Different object types use different standardized criteria for determining that an event has occurred. Proprietary object types that support intrinsic notification shall follow these criteria for determining when an event has occurred. Proprietary intrinsic reporting shall use the services described in 13.8 and 13.9. The standardized objects that may optionally provide intrinsic event notification support and the event types they shall employ are summarized in Table 13-2. The object properties are described in Clause 12.

The standard object types shown in Table 13-2 use standard event types for reporting the values of parameters relevant to the event. These values are returned in the 'Event Values' parameter of the ConfirmedEventNotification and UnconfirmedEventNotification services. The event type determines which set of values to return for each of the standard event types. These return values are summarized in Table 13-3. Proprietary object types that do not use one of the standard event types may return a list of property values instead.

In the case of Binary Input and Binary Value objects, the Alarm_Value property specifies which of the two Present_Value states shall be interpreted as OFFNORMAL. The opposing state shall be interpreted as NORMAL. Transitions to either or both states may generate notifications if the corresponding flags are set in Event_Enable.

In the case of Multi-state Input, Multi-state Value, Life Safety Point, and Life Safety Zone objects the Alarm_Values property lists each of the possible Present_Value states that shall be interpreted as OFFNORMAL. The Fault_Values property lists each of the possible Present_Value states that shall be interpreted as FAULT. All other Present_Value states shall be interpreted as NORMAL. Transitions to any of the states may generate notifications if the corresponding flags are set in Event_Enable.

In the case of Life Safety Zone and Life Safety Point, the Life_Safety_Alarm_Values property lists each of the possible Present_Value states that shall be interpreted as LIFE_SAFETY_ALARM. The Alarm_Values property lists each of the possible Present_Value states that shall be interpreted as OFFNORMAL. The Fault_Values property lists each of the possible Present_Value states that shall be interpreted as FAULT. All other Present_Value states shall be interpreted as NORMAL. Transitions to any of the states may generate notifications if the corresponding flags are set in Event_Enable.

Table 13-2. Standard Objects That May Support Intrinsic Reporting

Object Type	Criteria	Event Type
Binary Input, Binary Value, Multi-state Input, Multi-state Value	If Present_Value changes to a new state for longer than Time_Delay AND the new transition is enabled in Event_Enable	CHANGE_OF_STATE
Binary Output, Multi-state Output	If Present_Value differs from Feedback_Value for longer than Time_Delay AND the new transition is enabled in Event_Enable	COMMAND_FAILURE
Loop	If the absolute difference between Setpoint and Controlled_Variable_Value exceeds Error_Limit for longer than Time_Delay AND the new transition is enabled in Event_Enable	FLOATING_LIMIT
Analog Input, Analog Output, Analog Value, Pulse Converter	If Present_Value exceeds range between High_Limit and Low_Limit for longer than Time_Delay AND the new transition is enabled in Event_Enable and Limit_Enable, OR Present_Value returns within the High_Limit - Deadband to Low_Limit + Deadband range for longer than Time_Delay AND the new transition is enabled in Event_Enable and Limit_Enable	OUT_OF_RANGE
Trend Log	If Event_State is NORMAL state and Records_Since_Notification is equal to Notification_Threshold	BUFFER_READY
Life Safety Point, Life Safety Zone	If Present_Value changes to become equal to one of the values in the Life_Safety_Alarm_Values list AND remains equal to a value within the Life_Safety_Alarm_Values list for longer than Time_Delay AND the new transition is enabled in Event-Enable OR If Present_Value changes to become equal to one of the values in the Alarm_Values list AND remains equal to a value within the Alarm_Values list for longer than Time_Delay AND the new transition is enabled in Event-Enable OR Mode changes	CHANGE_OF_LIFE_SAFETY
Accumulator	If Pulse_Rate exceeds range from Low_Limit through High_Limit for longer than Time_Delay AND the new transition is enabled in Event_Enable and Limit_Enable, OR Pulse_Rate returns to range from Low_Limit through High_Limit for longer than Time_Delay AND the new transition is enabled in Event_Enable and Limit_Enable	UNSIGNED_RANGE

Table 13-3. Standard Object Property Values Returned in Notifications

Object	Event Type	Notification Parameters	Referenced Object's Properties
Binary Input, Binary Value, Multi-state Input, Multi-state Value	CHANGE_OF_STATE	New_State Status_Flags	Present_Value Status_Flags
Binary Output, Multi-state Output	COMMAND_FAILURE	Command_Value Status_Flags Feedback_Value	Present_Value Status_Flags Feedback_Value
Loop	FLOATING_LIMIT	Referenced_Value Status_Flags Setpoint_Value Error_Limit	Controlled_Variable_Value Status_Flags Setpoint Error_Limit
Analog Input, Analog Output, Analog Value, Pulse Converter	OUT_OF_RANGE	Exceeding_Value Status_Flags Deadband Exceeded_Limit	Present_Value Status_Flags Deadband Low_Limit or High_Limit
Trend Log	BUFFER_READY	Buffer_Property Previous_Notification Current_Notification	BACnetDeviceObjectPropertyReference[1] Last_Notify_Record Total_Record_Count
Life Safety Point, Life Safety Zone	CHANGE_OF_LIFE_SAFETY	New_State New_Mode Status_Flags Operation_Expected	Present_Value Mode Status_Flags Operation_Expected
Accumulator	UNSIGNED_RANGE	Exceeding_Value Status_Flags Exceeded_Limit	Pulse_Rate Status_Flags Low_Limit or High_Limit

[1] This parameter conveys a reference to the Log_Buffer property of the Trend Log object.

Table 13-4. Notification Parameters for Standard Event Types

Event Type	Notification Parameters	Description
CHANGE_OF_BITSTRING	Referenced_Bitstring Status_Flags	The new value of the referenced bitstring property. The Status_Flags of the referenced object.
CHANGE_OF_STATE	New_State Status_Flags	The new value of the referenced property. The Status_Flags of the referenced object.
CHANGE_OF_VALUE	New_Value Status_Flags	The new value of the referenced property. The Status_Flags of the referenced object.
COMMAND_FAILURE	Command_Value Status_Flags Feedback_Value	The value of the property that was commanded. The Status_Flags of the referenced object. The value that differs from the Command_Value.
FLOATING_LIMIT	Referenced_Value Status_Flags Setpoint_Value Error_Limit	The new value of the referenced property. The Status_Flags of the referenced object. The value of the setpoint reference. The difference limit that was exceeded.
OUT_OF_RANGE	Exceeding_Value Status_Flags Deadband Exceeded_Limit	The value that exceeded a limit. The Status_Flags of the referenced object. The deadband used for limit checking. The limit that was exceeded.
BUFFER_READY	Buffer_Property Previous_Notification Current_Notification	Reference to the buffer property. Current_Notification parameter of the previous notification sent or 0 if no previouis notification has been sent. SequenceNumber of the record that triggered this notification.
CHANGE_OF_LIFE_SAFETY	New_State New_Mode Status_Flags Operation_Expected	The new value of the referenced property The new mode of the referenced object The Status_Flags of the referenced object The next operation requested by the referenced object.
UNSIGNED_RANGE	Exceeding_Value Status_Flags Exceeded_Limit	The value that exceeded a limit The Status_Flags of the referenced object The limit that was exceeded

13.3 Algorithmic Change Reporting

Algorithmic change reporting enables a BACnet device to provide one or more alarm or event sources, defined by Event Enrollment objects, to generate alarm or event notifications that may be directed to one or more destinations. Any of the standardized algorithms may be used to establish criteria for change reporting. Once established, occurrences of change may be reported to one or more destinations based on further criteria. Changes of value of specific properties of an object may be programmed to trigger event notifications to be sent to one or more destinations based on notification class. Typically, event notifications are sent to application programs represented by processes within a notification-client device. The ConfirmedEventNotification and UnconfirmedEventNotification services are used by the notification-server to convey notifications.

The object(s) whose properties are referred to is known as the Reference Object(s). The criteria used to ascertain that an event has occurred are determined by the Event Type.

The following event type algorithms are specified in this standard because of their widespread occurrence in building automation and control systems. They are:

(a) CHANGE_OF_BITSTRING
(b) CHANGE_OF_STATE
(c) CHANGE_OF_VALUE
(d) COMMAND_FAILURE
(e) FLOATING_LIMIT
(f) OUT_OF_RANGE
(g) BUFFER_READY
(h) CHANGE_OF_LIFE_SAFETY
(i) UNSIGNED_RANGE

Events are based on algorithms applied to the properties of specific objects called "reference objects." Such a property is called a "referenced property." These properties are referenced by the Object_Property_Reference of an Event Enrollment object. The parameters required to compute the current state of the event are contained in the Event_Parameters property of the same Event Enrollment object. The relationship between the Event_Type, Event_State, and the Event_Parameters are summarized in Table 12-14. Each parameter in the Event_Parameters property is described in detail in 12.11.7. The algorithms that are used to determine the Event_State of each event type are specified in this subclause. Figure 13-11 shows an example of the relationships between Event Enrollment objects, referenced objects, and Notification Class objects.

Event enrollment objects shall use standard event types for reporting the values of parameters relevant to the event. These values are returned in the 'Event Values' parameter of the ConfirmedEventNotification and UnconfirmedEventNotification services. The event type determines which set of values to return for each of the standard event types. These return values are summarized in Table 13-4. Event Enrollment objects must report proprietary event algorithms using the extended-event notification type.

If the referenced object and property is any of those appearing in Table 13-3, then the value of the Status_Flags property of the referenced object shall be conveyed by the 'status-flags' parameter of the ConfirmedEventNotification or UnconfirmedEventNotification service request issued by the Event Enrollment object. A change in the FAULT flag (independent of any Time_Delay notification parameter) of the referenced object's Status_Flags property shall be treated as if the referenced object's Event_State property had made the associated transition to or from the FAULT state and a notification issued if notification for the resultant transition is enabled.

When an Event Enrollment object is created, its Event_State property shall be initialized to NORMAL.

If any of the parameters required to process the event algorithm are missing or inconsistent with the Event_Type, the Event Enrollment object shall become disabled and all the bit flags in the Event_Enable property shall be cleared.

13.3.1 CHANGE_OF_BITSTRING Algorithm

A CHANGE_OF_BITSTRING occurs when the value of the referenced property becomes equal to one of the values contained in the List_Of_Bitstring_Values after applying the Bitmask, and that value remains equal for Time_Delay seconds. For the purpose of event notification, CHANGE_OF_BITSTRING generates a TO-OFFNORMAL transition.
A CHANGE_OF_BITSTRING clears when the value of the referenced property is no longer equal to one of the values contained in the List_Of_Bitstring_Values after applying the Bitmask, and that value remains not equal for Time_Delay seconds. The clearing of a CHANGE_OF_BITSTRING generates a TO-NORMAL transition. See Figure 13-1.

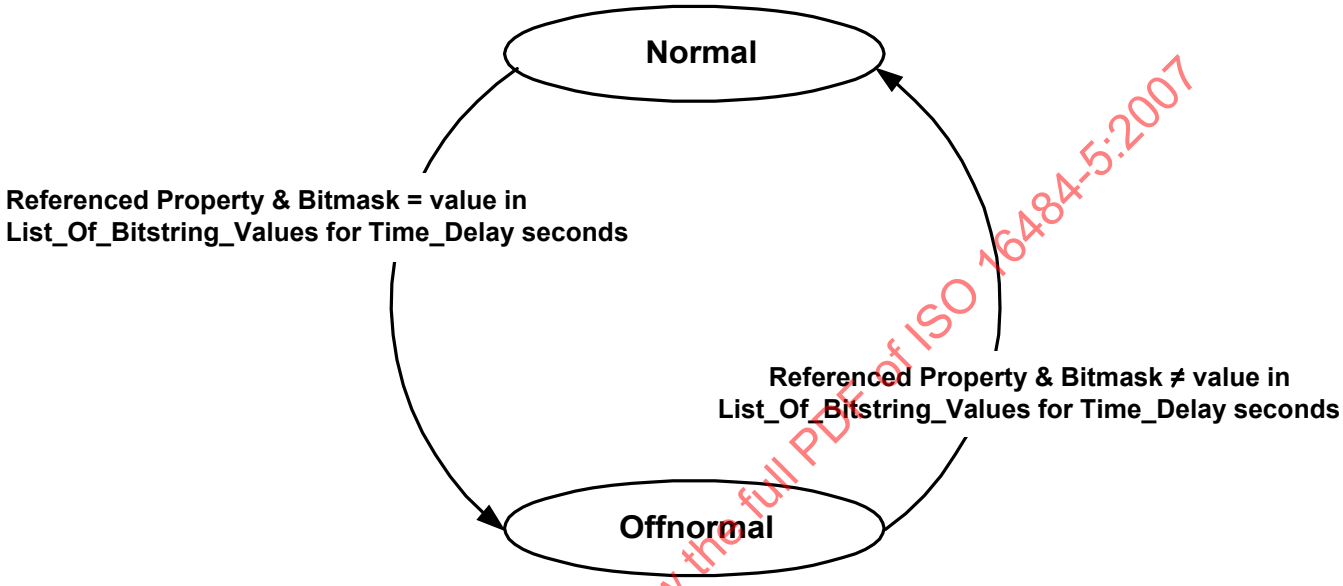


Figure 13-1. CHANGE_OF_BITSTRING algorithm.

13.3.2 CHANGE_OF_STATE Algorithm

A CHANGE_OF_STATE occurs when the value of the referenced property becomes equal to one of the values contained in the List_Of_Values, and this value remains equal for Time_Delay seconds. This type of event may only be applied to properties that have discrete or enumerated values, including Boolean. For the purposes of event notification, CHANGE_OF_STATE events generate a TO-OFFNORMAL transition.

A CHANGE_OF_STATE event clears when the value of the referenced property is no longer equal to one of the values contained in the List_Of_Values, and that value remains not equal for Time_Delay seconds. The clearing of a CHANGE_OF_STATE generates a TO-NORMAL transition. See Figure 13-2.

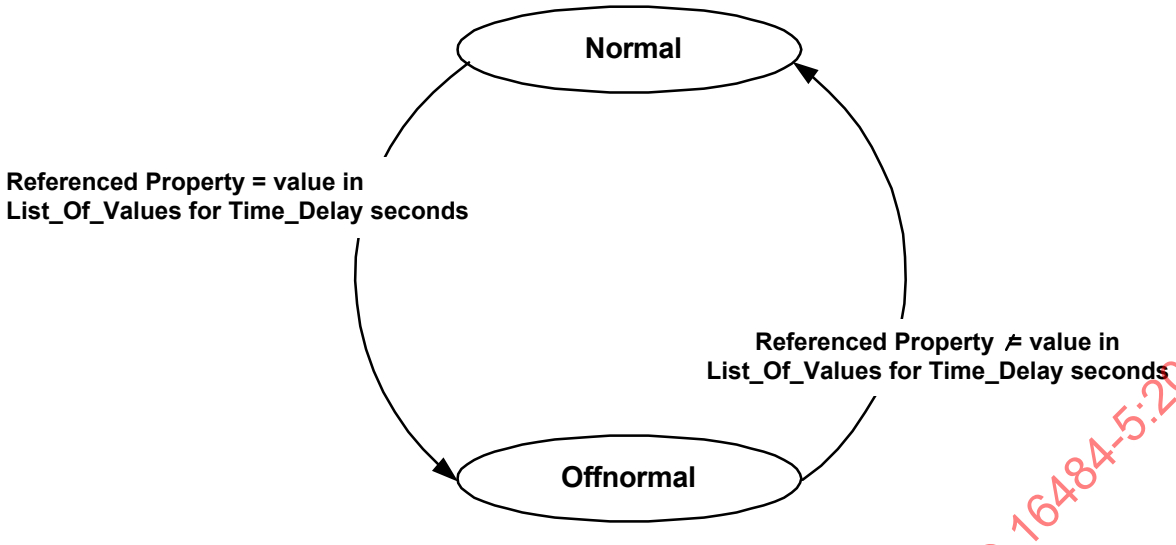


Figure 13-2. CHANGE_OF_STATE algorithm.

13.3.3 CHANGE_OF_VALUE Algorithm

A CHANGE_OF_VALUE occurs when the absolute value of a referenced property changes by an amount equal to or greater than the Referenced_Property_Increment, and this condition remains for Time_Delay seconds. The initialization of the referenced property value used in the algorithm shall be a local matter, but the value of the referenced property at the time a CHANGE_OF_VALUE occurs shall be used in carrying out the algorithm until the next CHANGE_OF_VALUE. For the purposes of event notification, CHANGE_OF_VALUE generates TO-NORMAL transitions. See Figure 13-3.

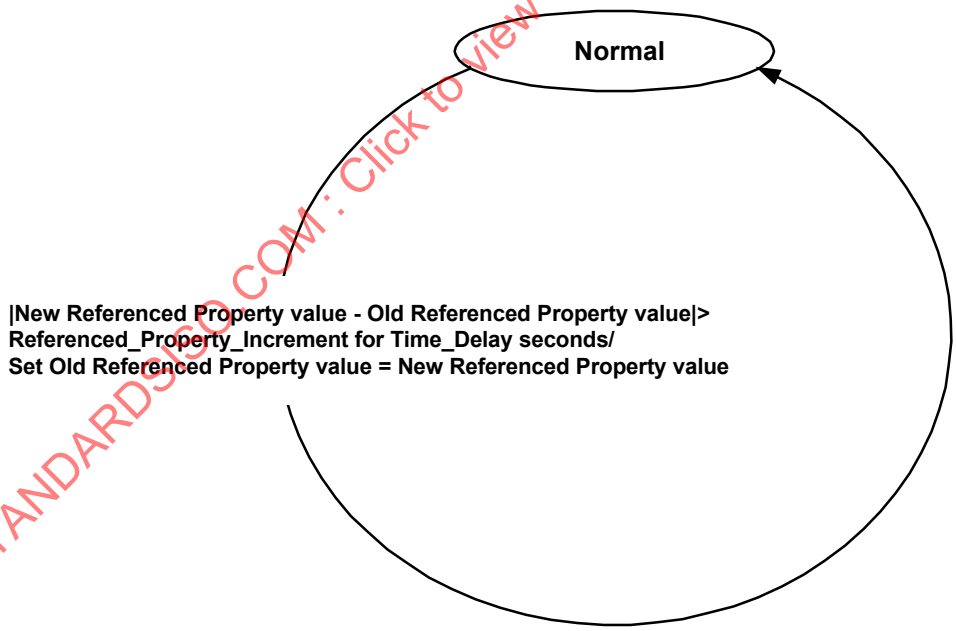


Figure 13-3. CHANGE_OF_VALUE algorithm.

If the referenced property is a bitstring datatype, then the CHANGE_OF_VALUE occurs when any of the bits defined in the Bitmask parameter change state and remain changed for Time_Delay seconds. See Figure 13-4.

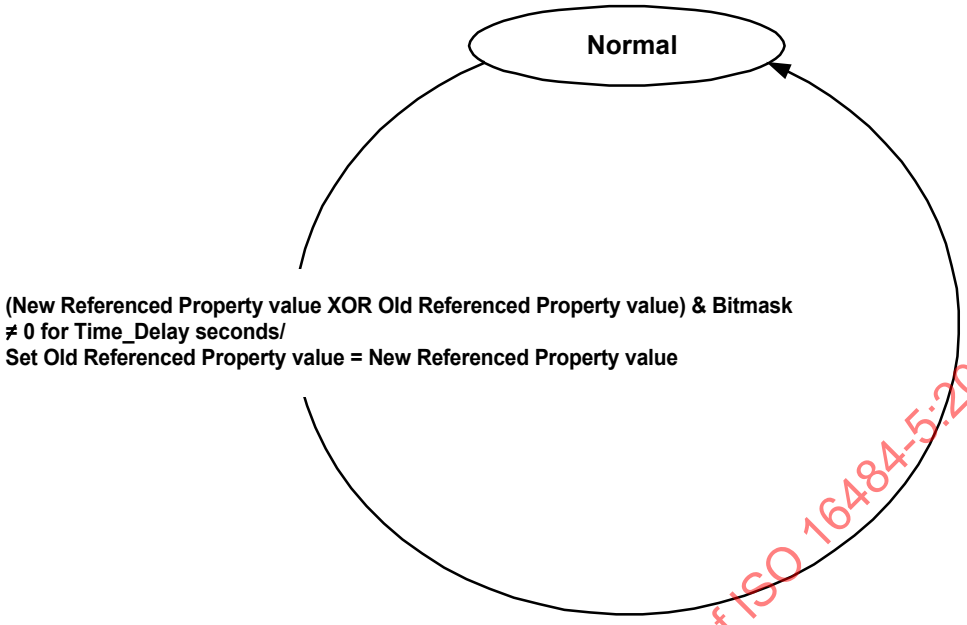


Figure 13-4. CHANGE_OF_VALUE algorithm (Bitstring).

13.3.4 COMMAND_FAILURE Algorithm

A COMMAND_FAILURE occurs if the values of the referenced property and the Feedback_Property_Reference disagree for a time period greater than the Time_Delay parameter. It may be used, for example, to verify that a process change has occurred after writing to a property. This type of event shall only be applied to properties that take on discrete values. For the purpose of event notification, COMMAND_FAILURE generates a TO-OFFNORMAL transition.

A COMMAND_FAILURE clears if, at any time subsequent to its occurrence, the value of the referenced property and the Feedback_Property_Reference become equal for a time period greater than the Time_Delay parameter. The clearing of a COMMAND_FAILURE generates a TO-NORMAL transition. See Figure 13-5.

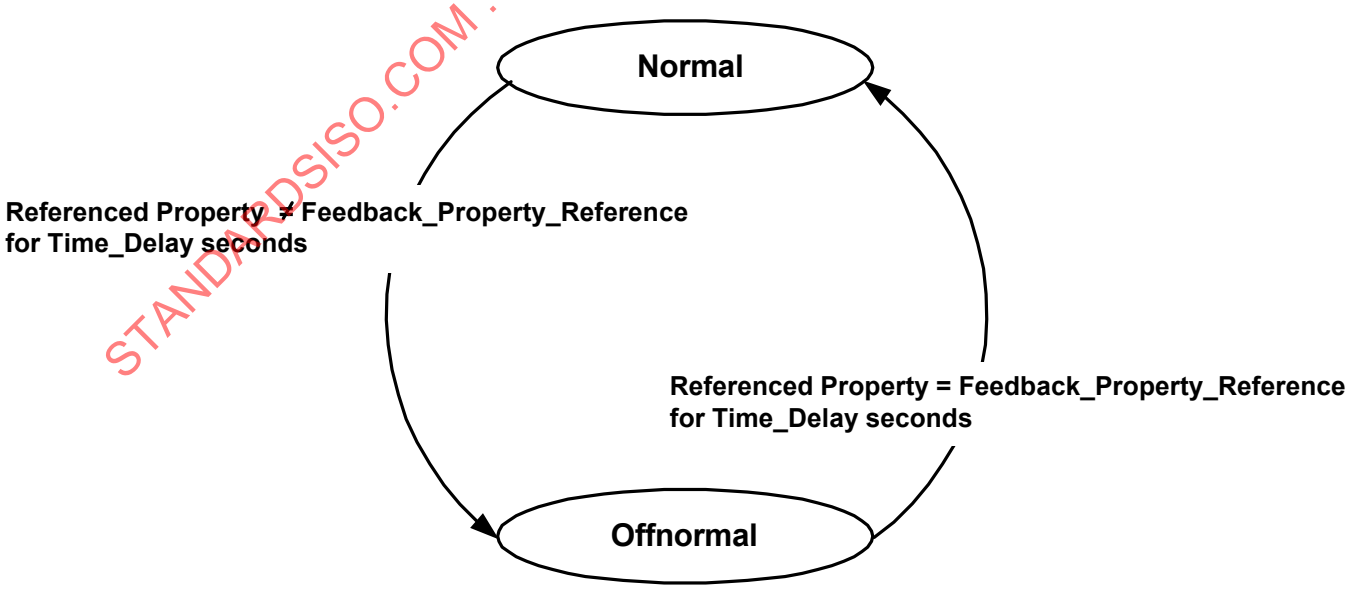


Figure 13-5. COMMAND_FAILURE algorithm.

13.3.5 FLOATING_LIMIT Algorithm

A FLOATING_LIMIT occurs if the referenced property leaves a range of values determined by the current value of the Setpoint_Reference, High_Diff_Limit, Low_Diff_Limit, and Deadband.

Starting in the NORMAL state, if the value of the referenced property becomes greater than Setpoint_Reference + High_Diff_Limit for a period of time greater than the Time_Delay parameter, the Event Enrollment object sets the Event_State property to the HIGH_LIMIT state and generates a TO-OFFNORMAL transition. The Event Enrollment object generates a TO-NORMAL transition and sets the Event_State property to NORMAL when the referenced property returns to a value less than Setpoint_Reference + High_Diff_Limit - Deadband for a period of time greater than the Time_Delay parameter. In each case, the event notification shall show an 'Event Type' of FLOATING_LIMIT.

Starting in the NORMAL state, if the value of the referenced property becomes less than Setpoint_Reference - Low_Diff_Limit for a period greater than the Time_Delay parameter, the Event Enrollment object sets the Event_State property to the LOW_LIMIT state and generates a TO-OFFNORMAL transition. The Event Enrollment object generates a TO-NORMAL transition and sets the Event_State property to NORMAL when the referenced property returns to a value greater that Setpoint_Reference - Low_Diff_Limit + Deadband for a period of time greater than the Time_Delay parameter. In each case, the event notification shall show an EventType of FLOATING_LIMIT. See Figure 13-6.

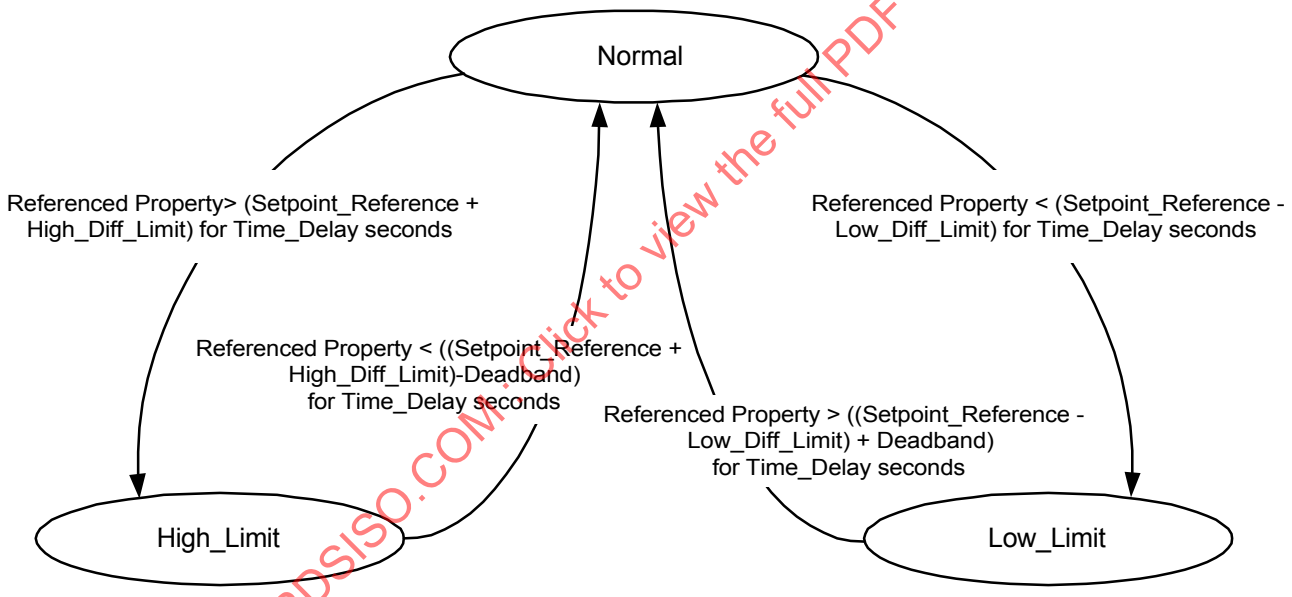


Figure 13-6. FLOATING_LIMIT algorithm.

Figure 13-7 shows the relationship of the various parameters used in the FLOATING_LIMIT algorithm.

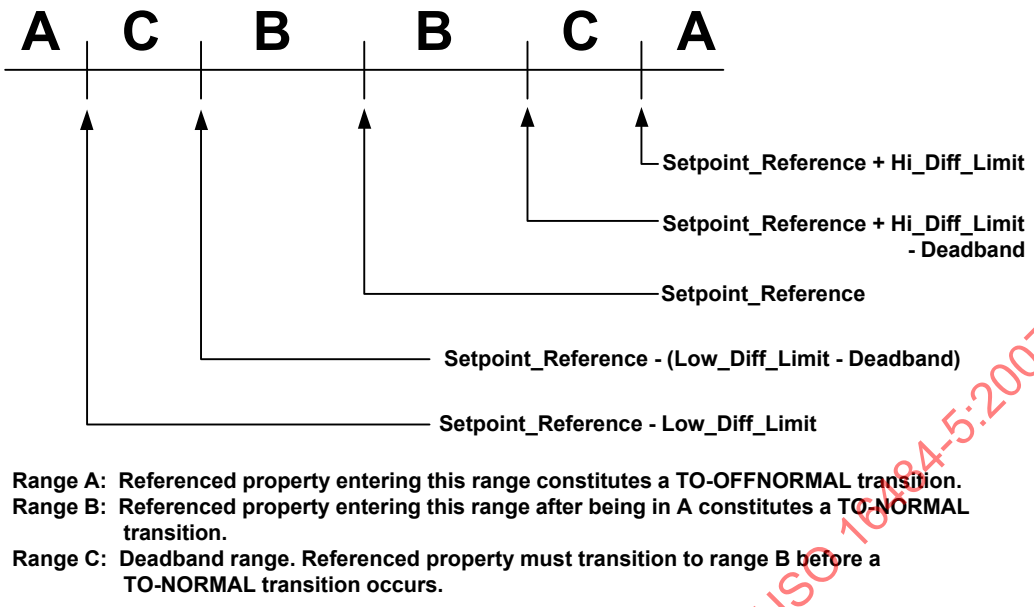


Figure 13-7. FLOATING_LIMIT algorithm.

13.3.6 OUT_OF_RANGE Algorithm

An OUT_OF_RANGE occurs if the referenced property leaves a range of values defined by the High_Limit and Low_Limit parameters and remains there for Time_Delay seconds. If the transition is to a value above the High_Limit or below the Low_Limit, the Event Enrollment object generates a TO-OFFNORMAL transition. The event notification shall show an 'Event Type' of OUT_OF_RANGE.

An OUT_OF_RANGE clears when the referenced property attains a value greater than the (Low_Limit + Deadband) or a value less than the (High_Limit - Deadband) and remains there for Time_Delay seconds. Note that the limit values may be Boolean TRUE or FALSE as well as analog values. The Event Enrollment object generates a TO-NORMAL transition. The event notification shall show an 'Event Type' of OUT_OF_RANGE. See Figure 13-8.

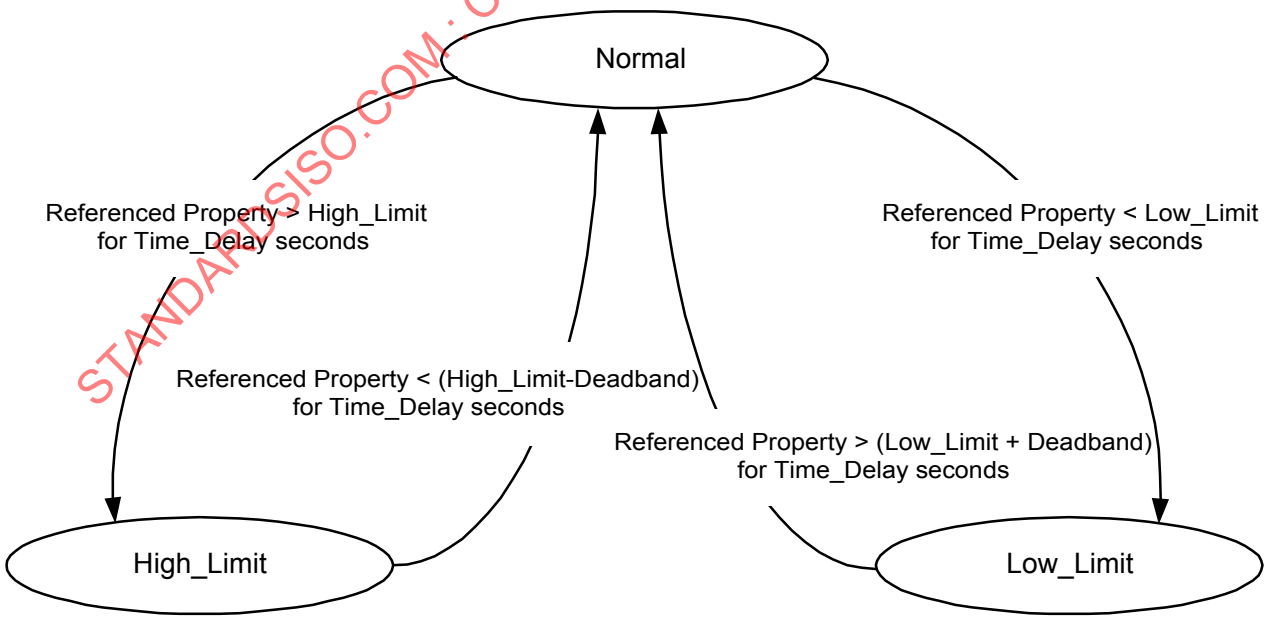


Figure 13-8. OUT_OF_RANGE algorithm.

13.3.7 BUFFER_READY Algorithm

A BUFFER_READY occurs when the number of records specified by Notification_Threshold have been entered into the log since the start of operation or the previous notification, whichever is most recent. The number of records collected is determined by the formula Total_Record_Count – Previous_Notification_Count, if Total_Record_Count is greater than or equal to Previous_Notification_Count, otherwise it is determined by the formula Total_Record_Count – Previous_Notification_Count + 2^{32}. Upon completion of the notification, Previous_Record_Count is set to the value of Total_Record_Count that caused the notifications to occur.

Previous_Notification_Count is an internal variable, of type Unsigned32, which maintains the value of Total_Record_Count at which the most recent notification took place. Upon initialization it shall be set to the value of the Total_Record_Count property in the object referenced by Object_Property_Reference.

For the purposes of event notification, the BUFFER_READY event generates TO-NORMAL transitions.

13.3.8 CHANGE_OF_LIFE_SAFETY Algorithm

A CHANGE_OF_LIFE_SAFETY occurs when the value of the referenced property becomes equal to any of the values in the List_Of_Life_Safety_Alarm_Values, and remains within the set of values in this list for Time_Delay seconds. The resulting event state is LIFE_SAFETY_ALARM.

A CHANGE_OF_LIFE_SAFETY also occurs when the value of the referenced property becomes equal to any of the values in the List_Of_Alarm_Values, and remains within the set of values in this list for Time_Delay seconds. The resulting event state is OFFNORMAL.

A CHANGE_OF_LIFE_SAFETY also occurs for any change of the property referred to by the Mode_Property_Reference.

For the purpose of event notification, CHANGE_OF_LIFE_SAFETY events generate a TO-OFFNORMAL transition.

The CHANGE_OF_LIFE_SAFETY algorithm is depicted in Figure 13-9

13.3.9 UNSIGNED_RANGE Algorithm

An UNSIGNED_RANGE occurs if the referenced property leaves the range of values from Low_Limit through High_Limit parameters and remains there for Time_Delay seconds. If the transition is to a value above High_Limit or below Low_Limit, the Event Enrollment object generates a TO-OFFNORMAL transition. The event notification shall show an 'Event Type' of UNSIGNED_RANGE.

An UNSIGNED_RANGE clears when the referenced property attains a value from Low_Limit through High_Limit and remains there for Time_Delay seconds. The Event Enrollment object generates a TO-NORMAL transition. The event notification shall show an 'Event Type' of UNSIGNED_RANGE. See Figure 13-10.

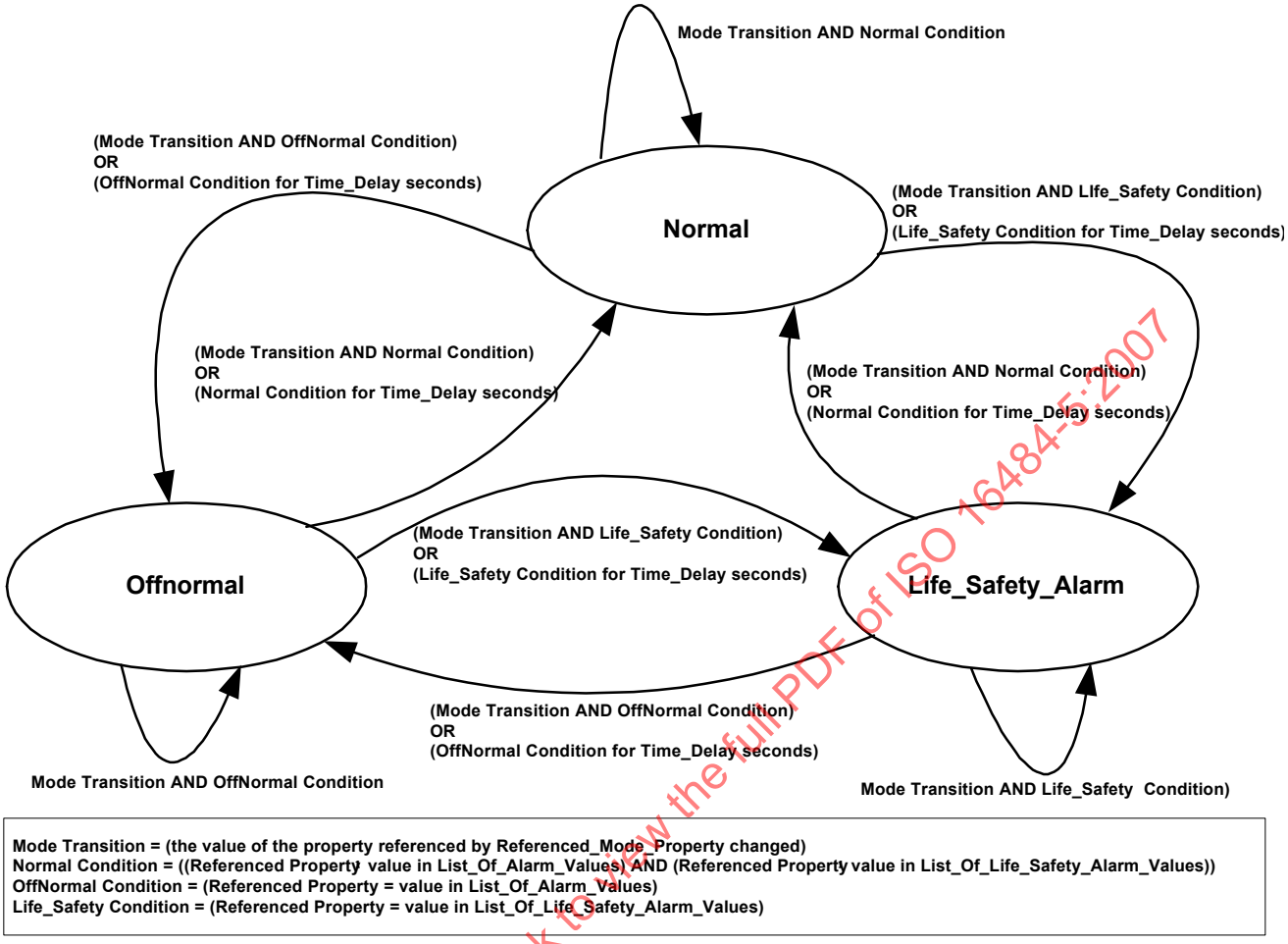


Figure 13-9. CHANGE_OF_LIFE_SAFETY algorithm.

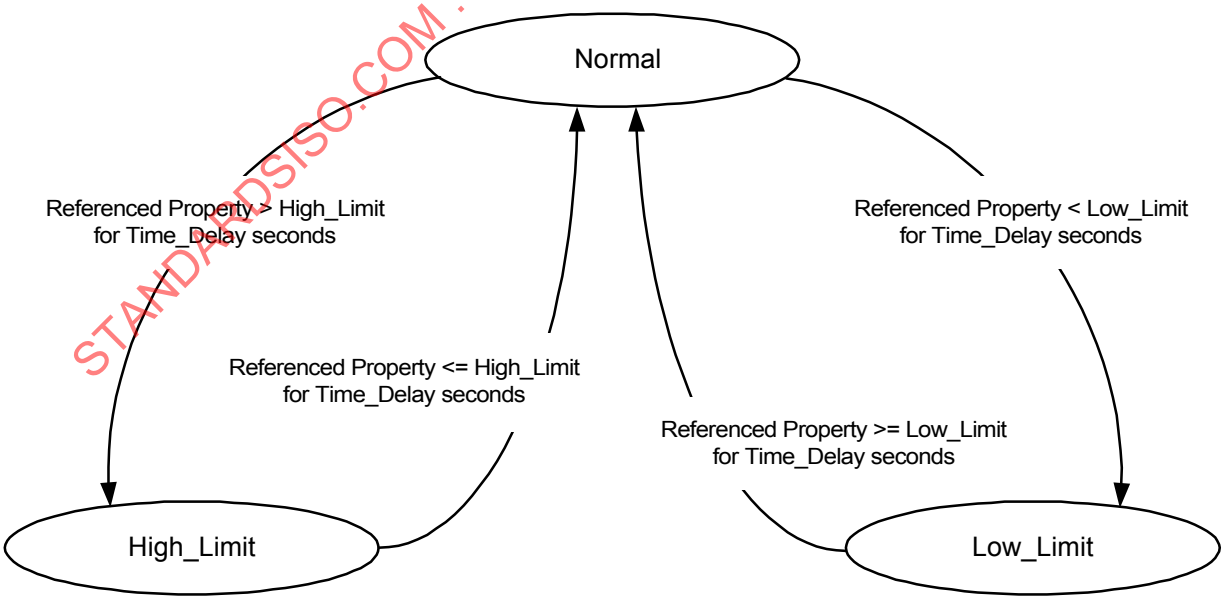


Figure 13-10. UNSIGNED_RANGE algorithm.

13.4 Alarm and Event Occurrence and Notification

The connection between the occurrence of an event and the transmission of a notification message to one or more recipients is established in one of several ways, depending on the type of reporting desired. COV events are connected to one or more subscribers through the use of the SubscribeCOV service. Intrinsic events are connected to one or more recipients indirectly through association with a Notification Class object. Algorithmic change events are connected to one or more recipients by the creation of an Event Enrollment object.

The criteria for event occurrence detection are specified through properties of those objects that may generate event notifications and using parameters of the SubscribeCOV service. These properties are described under the particular objects that may generate events, as summarized in Table 13-2, and in the Event Enrollment object. All objects that generate events have a property that indicates the state of the object with respect to alarm and event handling, and each such object may be explicitly enabled or disabled for reporting of specific events.

Intrinsic object-generated events, and events generated by Event Enrollment objects, may be controlled by a Notification Class object that defines their handling options. Event Enrollment objects, may alternatively specify single recipients to receive notifications without special handling.

COV and event notifications may be specified to use either confirmed or unconfirmed services for notification messages. By providing two kinds of notification mechanisms, BACnet allows the application designer to decide the relative importance of each event and whether or not notification of its occurrence is essential or merely desirable. In the former case, notification can be carried out with a confirmed service and repeated for as many recipients as required. In the latter case, an unconfirmed service using a broadcast or multicast address may be used.

Event Enrollment objects and Notification Class objects specify the destination devices for notification messages using BACnetRecipients. The recipients may be individual devices, groups of devices with a common multicast address, or all devices reachable by a broadcast address. If a broadcast is used, the scope may be limited to all devices on a single network or it may be extended to encompass all devices on a BACnet internetwork. See Clause 6.

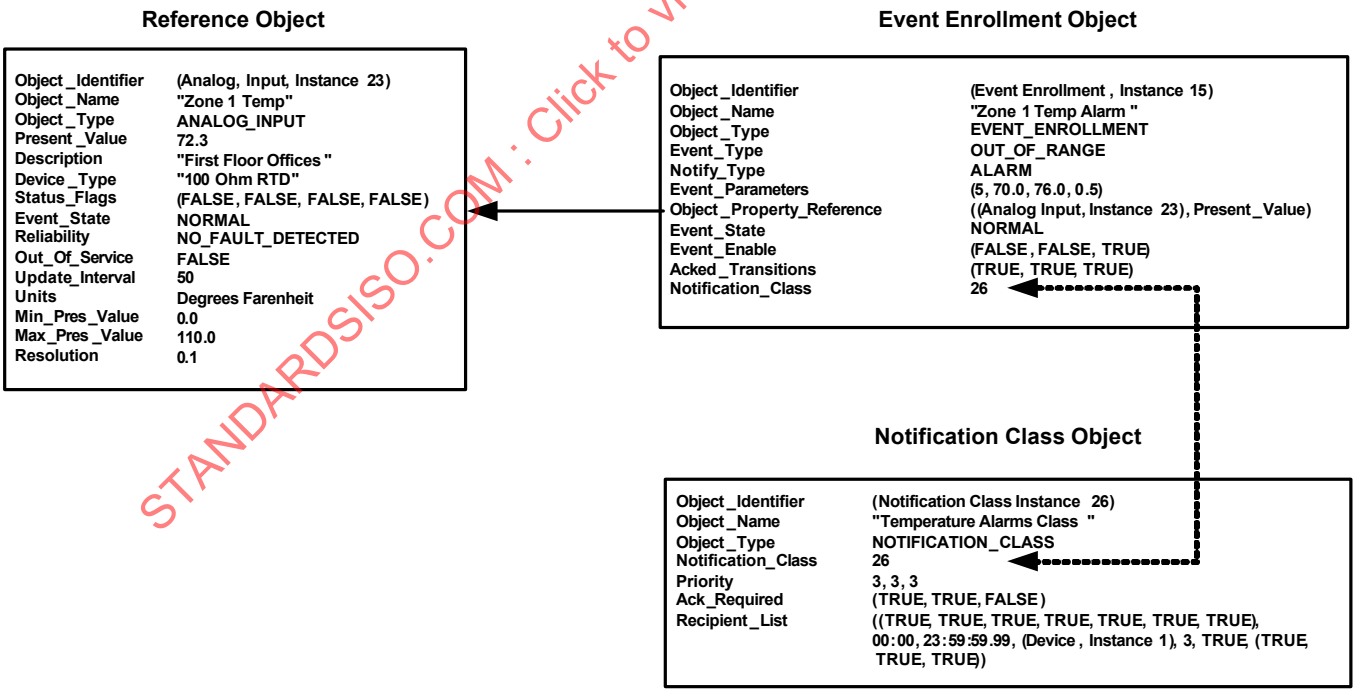


Figure 13-11. Example of an Event Enrollment.

The event notification services contain a 'Time Stamp' parameter that indicates the chronological order of events. This 'Time Stamp' may be the actual time as determined by the local device clock or, if the device has no clock, a sequence number. Sequence numbers are required to increase monotonically up to their maximum value, at which point the number "wraps around" to zero. A device may have a single sequence number for all event-initiating objects, or it may have a separate sequence number for each object.

Eleven services are defined specifically for event management:

(a) AcknowledgeAlarm
(b) ConfirmedCOVNotification
(c) UnconfirmedCOVNotification
(d) ConfirmedEventNotification
(e) UnconfirmedEventNotification
(f) GetAlarmSummary
(g) GetEnrollmentSummary
(h) GetEventInformation
(i) LifeSafetyOperation
(j) SubscribeCOV
(k) SubscribeCOVProperty

13.4.1 Alarm and Event Priority Classification

Alarms and events traversing the BACnet network need prioritization to assure that important information reaches its destination and is acted upon quickly. To assure alarm prioritization at the network level, the Network Priority as defined in 6.2.2 shall be set as a function of the alarm and event priority as defined in Table 13-5. Annex M provides additional clarity and examples of specific messages and priorities.

Table 13-5. Alarm and Event Priority - Network Priority Association

Alarm and Event Priority	Network Priority
00 – 63	Life Safety message
64 - 127	Critical Equipment message
128 - 191	Urgent message
192 - 255	Normal message

(Blank page)

13.5 AcknowledgeAlarm Service

In some systems a device may need to know that an operator has seen the alarm notification. The AcknowledgeAlarm service is used by a notification-client to acknowledge that a human operator has seen and responded to an event notification with 'AckRequired' = TRUE. Ensuring that the acknowledgment actually comes from a person with appropriate authority is a local matter. This service may be used in conjunction with either the ConfirmedEventNotification service or the UnconfirmedEventNotification service.

13.5.1 Structure

The structure of the AcknowledgeAlarm service primitives is shown in Table 13-6. The terminology and symbology used in this table are explained in 5.6.

Table 13-6. Structure of AcknowledgeAlarm Service Primitives

Parameter Name	Req	Ind	Rsp	Cnf
Argument	M	M(=)		
Acknowledging Process Identifier	M	M(=)		
Event Object Identifier	M	M(=)		
Event State Acknowledged	M	M(=)		
Time Stamp	M	M(=)		
Acknowledgment Source	M	M(=)		
Time Of Acknowledgment	M	M(=)		
Result(+)			S	S(=)
Result(-)			S	S(=)
Error Type			M	M(=)

13.5.1.1 Argument

This parameter shall convey the parameters for the AcknowledgeAlarm confirmed service request.

13.5.1.2 Acknowledging Process Identifier

This parameter, of type Unsigned32, shall specify the 'Process Identifier' parameter from the event notification to which this acknowledgment is a response. This allows the initiating object to ensure that the desired process has received the notification.

13.5.1.3 Event Object Identifier

This parameter, of type BACnetObjectIdentifier, shall specify the 'Event Object Identifier' parameter of the event notification to which this acknowledgment is a response. This is the same object that initiated the event notification that is being acknowledged.

13.5.1.4 Event State Acknowledged

This parameter, of type BACnetEventState, shall be equal to the value of the 'To State' from the event notification that is being acknowledged. This parameter is included so that the remote device that initiated the event notification can ensure that the state being acknowledged is recorded in the Acked_Transitions property of the initiating object.

13.5.1.5 Time Stamp

This parameter, of type BACnetTimeStamp, shall convey the same 'Time Stamp' that was received in the event notification that is being acknowledged by this service. The 'Time Stamp' is used by the recipient of this service request to identify the event notification that is being acknowledged in the case when more than one has been issued with the same 'To State'.

13.5.1.6 Acknowledgment Source

This parameter, of type CharacterString, shall specify the identity of the operator or process that is acknowledging the event notification.

13.5.1.7 Time Of Acknowledgment

This parameter, of type BACnetTimeStamp, shall specify the time that the operator or process acknowledged the event notification.

13.5.1.8 Result(+)

The 'Result(+)' parameter shall indicate that the service request succeeded and the alarm is marked as acknowledged.

13.5.1.9 Result(-)

The 'Result(-)' parameter shall indicate that the service request failed. The reason for failure is specified by the 'Error Type' parameter.

13.5.1.10 Error Type

This parameter consists of two components: (1) 'Error Class' and (2) 'Error Code'. See Clause 18.

13.5.2 Service Procedure

After verifying the validity of the request, the responding BACnet-user shall attempt to locate the specified object. If the object exists and if the 'Time Stamp' parameter matches the most recent time for the event being acknowledged, then the bit in the Acked_Transitions property of the object that corresponds to the value of the 'Event State Acknowledged' parameter is acknowledged by changing the bit value to one, and a 'Result(+)' primitive shall be issued. Otherwise, a 'Result(-)' primitive shall be issued. If the acknowledgment was successful, causing a 'Result(+)' to be issued, then an event notification, with a 'Notify Type' parameter equal to ACK_NOTIFICATION, shall also be issued. The acknowledgment notification shall use the same type of service (confirmed or unconfirmed) directed to the same recipients to which the original confirmed or unconfirmed event notification was sent.

13.6 ConfirmedCOVNotification Service

The ConfirmedCOVNotification service is used to notify subscribers about changes that may have occurred to the properties of a particular object. Subscriptions for COV notifications are made using the SubscribeCOV service or the SubscribeCOVProperty service.

13.6.1 Structure

The structure of the ConfirmedCOVNotification service primitives is shown in Table 13-7. The terminology and symbology used in this table are explained in 5.6.

Table 13-7. Structure of ConfirmedCOVNotification Service Primitives

Parameter Name	Req	Ind	Rsp	Cnf
Argument	M	M(=)		
Subscriber Process Identifier	M	M(=)		
Initiating Device Identifier	M	M(=)		
Monitored Object Identifier	M	M(=)		
Time Remaining	M	M(=)		
List of Values	M	M(=)		
Result(+)			S	S(=)
Result(-)			S	S(=)
Error Type			M	M(=)

13.6.1.1 Argument

This parameter shall convey the parameters for the ConfirmedCOVNotification service request.

13.6.1.2 Subscriber Process Identifier

This parameter, of type Unsigned32, shall convey a numeric "handle" meaningful to the subscriber. This handle shall be used to identify the process within the COV client that should receive the notification.

13.6.1.3 Initiating Device Identifier

This parameter, of type BACnetObjectIdentifier, shall convey the Device Object_Identifier of the device that initiated the ConfirmedCOVNotification service request.

13.6.1.4 Monitored Object Identifier

This parameter, of type BACnetObjectIdentifier, shall convey the Object_Identifier of the object that has changed.

13.6.1.5 Time Remaining

This parameter, of type Unsigned, shall convey the remaining lifetime of the subscription in seconds. A value of zero shall indicate an indefinite lifetime without automatic cancellation.

13.6.1.6 List of Values

This parameter shall convey a list of one or more property values whose contents depends on the type of object being monitored. Table 13-1 summarizes the BACnet standard objects and those property values that shall be returned in the 'List of Values' parameter when those objects are enabled for COV reporting. The property values are returned in the order shown in Table 13-1.

13.6.1.7 Result(+)

The 'Result(+)' parameter shall indicate that the requested service has succeeded.

13.6.1.8 Result(-)

The 'Result(-)' parameter shall indicate that the service request has failed. The reason for failure shall be specified by the 'Error Type' parameter.

13.6.1.8.1 Error Type

This parameter shall consist of two component parameters: (1) the 'Error Class' and (2) the 'Error Code'. See Clause 18.

13.6.2 Service Procedure

After verifying the validity of the request, the responding BACnet-user shall take whatever local actions have been assigned to the indicated COV and issue a 'Result(+)' service primitive. If the service request cannot be executed, a 'Result(-)' service primitive shall be issued indicating the error encountered.

13.7 UnconfirmedCOVNotification Service

The UnconfirmedCOVNotification Service is used to notify subscribers about changes that may have occurred to the properties of a particular object, or to distribute object properties of wide interest (such as outside air conditions) to many devices simultaneously without a subscription. Subscriptions for COV notifications are made using the SubscribeCOV service. For unsubscribed notifications, the algorithm for determining when to issue this service is a local matter and may be based on a change of value, periodic updating, or some other criteria.

13.7.1 Structure

The structure of the UnconfirmedCOVNotification service primitive is shown in Table 13-8. The terminology and symbology used in this table are explained in 5.6.

Table 13-8. Structure of UnconfirmedCOVNotification Service Primitive

Parameter Name	Req	Ind
Argument	M	M(=)
Subscriber Process Identifier	M	M(=)
Initiating Device Identifier	M	M(=)
Monitored Object Identifier	M	M(=)
Time Remaining	M	M(=)
List of Values	M	M(=)

13.7.1.1 Argument

This parameter shall convey the parameters for the UnconfirmedCOVNotification service request.

13.7.1.2 Subscriber Process Identifier

This parameter, of type Unsigned32, shall convey a numeric "handle" meaningful to the subscriber. This handle shall be used to identify the process within the COVclient that should receive the notification. The value of zero is reserved for unsubscribed COV.

13.7.1.3 Initiating Device Identifier

This parameter, of type BACnetObjectIdentifier, shall convey the Device Object_Identifier of the device that initiated the UnconfirmedCOVNotification service request.

13.7.1.4 Monitored Object Identifier

This parameter, of type BACnetObjectIdentifier, shall convey the Object_Identifier of the object that has changed.

13.7.1.5 Time Remaining

This parameter, of type Unsigned, shall convey the remaining lifetime of the subscription in seconds. A value of zero shall indicate an indefinite lifetime, without automatic cancellation, or an unsubscribed notification.

13.7.1.6 List of Values

This parameter shall convey a list of one or more property values whose contents depend on the type of object being monitored. Table 13-1 summarizes the BACnet standard objects and those property values that shall be returned in the 'List of Values' parameter when those objects are enabled for COV reporting.

13.7.2 Service Procedure

Since this is an unconfirmed service, no response primitives are expected. Actions taken in response to this notification are a local matter.

13.8 ConfirmedEventNotification Service

The ConfirmedEventNotification service is used by a notification-server to notify a remote device that an event has occurred and that the notification-server needs a confirmation that the notification has been received. This confirmation means only that the device received the message. It does not imply that a human operator has been notified. A separate AcknowledgeAlarm service is used to indicate that an operator has acknowledged the receipt of the notification if the notification specifies that acknowledgment is required. If multiple recipients must be notified, a separate invocation of this service shall be used to notify each intended recipient. If a confirmation that a notification was received is not needed, then the UnconfirmedEventNotification may be used.

13.8.1 Structure

The structure of the ConfirmedEventNotification service primitives is shown in Table 13-9. The terminology and symbology used in this table are explained in 5.6.

Table 13-9. Structure of ConfirmedEventNotification Service Primitives

Parameter Name	Req	Ind	Rsp	Cnf
Argument	M	M(=)		
Process Identifier	M	M(=)		
Initiating Device Identifier	M	M(=)		
Event Object Identifier	M	M(=)		
Time Stamp	M	M(=)		
Notification Class	M	M(=)		
Priority	M	M(=)		
Event Type	M	M(=)		
Message Text	U	U(=)		
Notify Type	M	M(=)		
AckRequired	C	C(=)		
From State	C	C(=)		
To State	M	M(=)		
Event Values	C	C(=)		
Result(+)			S	S(=)
Result(-)			S	S(=)
Error Type			M	M

13.8.1.1 Argument

This parameter shall convey the parameters for the ConfirmedEventNotification service request.

13.8.1.2 Process Identifier

This parameter, of type Unsigned32, shall convey the identification of the process in the receiving device for which the notification is intended.

13.8.1.3 Initiating Device Identifier

This parameter, of type BACnetObjectIdentifier, shall convey the Device Object_Identifier of the device that initiated the ConfirmedEventNotification service request.

13.8.1.4 Event Object Identifier

This parameter, of type BACnetObjectIdentifier, shall specify the Object_Identifier of the object that is initiating the notification. This parameter is used by the AcknowledgeAlarm service to identify the object whose notification is being acknowledged.

13.8.1.5 Time Stamp

This parameter, of type BACnetTimeStamp, shall convey the current time as determined by the clock in the device issuing the service request. If this device has no clock, then this parameter shall convey a sequence number, of type Unsigned, which indicates the relative ordering of this event notification to all other event notifications issued by this device without regard to their intended recipient. The sequence numbers shall increase monotonically (they may be implemented using modulo arithmetic). A device may have a single sequence number for all event-initiating objects or a separate sequence number for each object.

13.8.1.6 Notification Class

This parameter, of type Unsigned, designates the notification class of the event. Definition of the various notification classes is a local matter (see 13.2, 13.4, and 12.21 for discussion of Notification Class objects).

13.8.1.7 Priority

This parameter, of type Unsigned8, shall specify the priority of the event that has occurred. The priority is specified by the Priority property of the Notification Class or Event Enrollment objects associated with this event. The possible range of priorities is 0-255. A lower number indicates a higher priority. The priority and the Network Priority (see 6.2.2) are associated as defined in Table 13-5.

13.8.1.8 Event Type

This parameter, of type BACnetEventType, shall specify the type of event that has occurred. Event types that are defined in this standard may be found in Table 12-15.

13.8.1.9 Message Text

This optional parameter, of type CharacterString, shall convey a string of printable characters. This parameter may be used to convey a message to be logged or displayed, which pertains to the occurrence of the event. The content of the message is a local matter.

13.8.1.10 Notify Type

This parameter, of type BACnetNotifyType, shall convey whether this notification is an event or an alarm or a notification that someone has acknowledged a previous event notification:

{ALARM, EVENT, ACK_NOTIFICATION}.

13.8.1.11 AckRequired

This parameter, of type BOOLEAN, shall convey whether this notification requires acknowledgment (TRUE) or not (FALSE). This parameter shall only be present if the 'Notify Type' parameter is EVENT or ALARM.

13.8.1.12 From State

This parameter, of type BACnetEventState, shall indicate the Event_State of the object prior to the occurrence of the event that initiated this notification. This parameter shall only be present if the 'Notify Type' parameter is EVENT or ALARM.

13.8.1.13 To State

This parameter, of type BACnetEventState, shall indicate the Event_State of the object after the occurrence of the event that initiated this notification.

13.8.1.14 Event Values

This parameter, of type BACnetNotificationParameters, shall convey a set of values relevant to the particular event and whose content depends on the type of object that initiated the notification (see Tables 13-2, 13-3, and 13-4). This parameter shall only be present if the 'Notify Type' parameter is EVENT or ALARM. The returned set of values may be either a list of property values representing properties of the object specified by the 'Event Object Identifier' or one of the standard event type notification parameter sets defined in Table 13-4.

13.8.1.15 Result(+)

The 'Result(+)' parameter shall indicate that the requested service has succeeded.

13.8.1.16 Result(-)

The 'Result(-)' parameter shall indicate that the service request has failed. The reason for failure shall be specified by the 'Error Type' parameter.

13.8.1.16.1 Error Type

This parameter shall consist of two component parameters: (1) the 'Error Class' and (2) the 'Error Code'. See Clause 18.

13.8.2 Service Procedure

After verifying the validity of the request, the responding BACnet-user shall take whatever local actions have been assigned to the indicated event occurrence and issue a 'Result(+)' service primitive. If the service request cannot be executed, a 'Result(-)' service primitive shall be issued indicating the encountered error.

13.9 UnconfirmedEventNotification Service

The UnconfirmedEventNotification service is used by a notification-server to notify a remote device that an event has occurred. Its purpose is to notify recipients that an event has occurred, but confirmation that the notification was received is not required. Applications that require confirmation that the notification was received by the remote device should use the ConfirmedEventNotification service. The fact that this is an unconfirmed service does not mean it is inappropriate for notification of alarms. Events of type Alarm may require a human acknowledgment that is conveyed using the AcknowledgeAlarm service. Thus, using an unconfirmed service to announce the alarm has no effect on the ability to confirm that an operator has been notified. Any device that executes this service shall support programmable process identifiers to allow broadcast and multicast 'Process Identifier' parameters to be assigned on a per installation basis.

13.9.1 Structure

The structure of the UnconfirmedEventNotification service primitives is shown in Table 13-10. The terminology and symbology used in this table are explained in 5.6.

Table 13-10. Structure of UnconfirmedEventNotification Service Primitive

Parameter Name	Req	Ind
Argument	M	M(=)
Process Identifier	M	M(=)
Initiating Device Identifier	M	M(=)
Event Object Identifier	M	M(=)
Time Stamp	M	M(=)
Notification Class	M	M(=)
Priority	M	M(=)
Event Type	M	M(=)
Message Text	U	U(=)
Notify Type	M	M(=)
AckRequired	C	C(=)
From State	C	C(=)
To State	M	M(=)
Event Values	C	C(=)

13.9.1.1 Argument

The 'Argument' parameter shall convey the parameters for the UnconfirmedEventNotification service request.

13.9.1.2 Process Identifier

This parameter, of type Unsigned32, shall convey the identification of the process in the receiving device for which the notification is intended.

13.9.1.3 Initiating Device Identifier

This parameter, of type BACnetObjectIdentifier, shall convey the Device Object_Identifier of the device that initiated the UnconfirmedEventNotification service request.

13.9.1.4 Event Object Identifier

This parameter, of type BACnetObjectIdentifier, shall specify the identifier of the object that is initiating the notification. This parameter is used by the AcknowledgeAlarm service to identify the object whose notification is being acknowledged.

13.9.1.5 Time Stamp

This parameter, of type BACnetTimeStamp, shall convey the current time as determined by the clock in the device issuing the service request. If this device has no clock, then this parameter shall convey a sequence number that indicates the relative ordering of this event notification to all other event notifications issued by this device without regard to their intended recipient. The sequence numbers shall increase monotonically (they may be implemented using modulo arithmetic). A device may have a single sequence number for all event-initiating objects or a separate sequence number for each object.

13.9.1.6 Notification Class

This parameter, of type Unsigned, designates the notification class of the event. Definition of the various notification classes is a local matter (see 13.2, 13.4, and 12.21 for discussion of Notification Class objects).

13.9.1.7 Priority

This parameter, of type Unsigned8, shall specify the priority of the event that has occurred. The priority is specified by the Priority property of the Notification Class object associated with the event. The possible range of priorities is 0-255. A lower number indicates a higher priority. The priority and the Network Priority (see 6.2.2) are associated as defined in Table 13-5.

13.9.1.8 Event Type

This parameter, of type BACnetEventType, shall specify the type of event that has occurred. Event types that are defined in this standard may be found in Table 12-15.

13.9.1.9 Message Text

This optional parameter, of type CharacterString, shall convey a string of printable characters. This parameter may be used to convey a message to be logged or displayed, which pertains to the occurrence of the event. The content of the message is a local matter.

13.9.1.10 Notify Type

This parameter, of type BACnetNotifyType, shall convey whether this notification is an event or an alarm or a notification that someone has acknowledged a previous event notification:

{EVENT, ALARM, ACK_NOTIFICATION}.

13.9.1.11 AckRequired

This parameter, of type BOOLEAN, shall convey whether this notification requires acknowledgment (TRUE) or not (FALSE). This parameter shall only be present if the 'Notify Type' parameter is EVENT or ALARM.

13.9.1.12 From State

This parameter, of type BACnetEventState, shall indicate the state of the object prior to the occurrence of the event that initiated this notification. This parameter shall only be present if the 'Notify Type' parameter is EVENT or ALARM.

13.9.1.13 To State

This parameter, of type BACnetEventState, shall indicate the state of the object after the occurrence of the event that initiated this notification.

13.9.1.14 Event Values

This parameter, of type BACnetNotificationParameters, shall convey a set of values relevant to the particular event and whose content depends on the type of object that initiated the notification (see Tables 13-2, 13-3, and 13-4). This parameter shall only be present if the 'Notify Type' parameter is EVENT or ALARM. The returned set of values may be either a list of property values representing properties of the object specified by the 'Event Object Identifier' or one of the standard event-type notification parameter sets defined in Table 13-4.

13.9.2 Service Procedure

Since this is an unconfirmed service, no response primitives are expected. Actions taken in response to this notification are a local matter.

13.10 GetAlarmSummary Service

The GetAlarmSummary service is used by a client BACnet-user to obtain a summary of "active alarms." The term "active alarm" refers to BACnet standard objects that have an Event_State property whose value is not equal to NORMAL and a Notify_Type property whose value is ALARM. The GetEnrollmentSummary service provides a more sophisticated approach with various kinds of filters.

13.10.1 Structure

The structure of the GetAlarmSummary service primitives is shown in Table 13-11. The terminology and symbology used in this table are explained in 5.6.

Table 13-11. Structure of GetAlarmSummary Service Primitives

Parameter Name	Req	Ind	Rsp	Cnf
Argument	M	M(=)		
Result(+)			S	S(=)
List of Alarm Summaries			M	M(=)
Object Identifier			M	M(=)
Alarm State			M	M(=)
Acknowledged Transitions			M	M(=)
Result(-)			S	S(=)
Error Type			M	M(=)

13.10.1.1 Argument

This parameter indicates the GetAlarmSummary confirmed service request.

13.10.1.2 Result(+)

The 'Result(+)' parameter shall indicate that the requested service has succeeded. A successful result includes the following parameters.

13.10.1.2.1 List of Alarm Summaries

The 'List of Alarm Summaries' shall contain zero or more Alarm Summaries. Each Alarm Summary shall consist of three parameters: 'Object Identifier', 'Alarm State', and 'Acknowledged States'. If the list is of length zero, then this shall be interpreted to mean that there are no active alarms for this device.

13.10.1.2.1.1 Object Identifier

This parameter, of type BACnetObjectIdentifier, shall identify the event-initiating object whose Event_State property is not equal to NORMAL and that has a Notify_Type property whose value is ALARM.

13.10.1.2.1.2 Alarm State

This parameter, of type BACnetEventState, indicates the value of the Event_State property of the object.

13.10.1.2.1.3 Acknowledged Transitions

This parameter, of type BACnetEventTransitionBits, indicates the value of the Acked_Transitions property of the object.

13.10.1.3 Result(-)

The 'Result(-)' parameter shall indicate that the service request has failed. The reason for failure shall be specified by the 'Error Type' parameter.

13.10.1.3.1 Error Type

This parameter shall consist of two component parameters: (1) the 'Error Class' and (2) the 'Error Code'. See Clause 18.

13.10.2 Service Procedure

After verifying the validity of the request, the responding BACnet-user shall search all event-initiating objects that have an Event_State property not equal to NORMAL and a Notify_Type property whose value is ALARM. A positive response containing the alarm summaries for objects found in this search shall be constructed. If no objects are found that meet these criteria, then a list of length zero shall be returned.

13.11 GetEnrollmentSummary Service

The GetEnrollmentSummary service is used by a client BACnet-user to obtain a summary of event-initiating objects. Several different filters may be applied to define the search criteria. This service may be used to obtain summaries of objects with any EventType and is thus a superset of the functionality provided by the GetAlarmSummary Service.

13.11.1 Structure

The structure of the GetEnrollmentSummary service primitives is shown in Table 13-12. The terminology and symbology used in this table are explained in 5.6.

Table 13-12. Structure of GetEnrollmentSummary Service Primitives

Parameter Name	Req	Ind	Rsp	Cnf
Argument	M	M(=)		
Acknowledgment Filter	M	M(=)		
Enrollment Filter	U	U(=)		
Event State Filter	U	U(=)		
Event Type Filter	U	U(=)		
Priority Filter	U	U(=)		
Notification Class Filter	U	U(=)		
Result(+)			S	S(=)
List of Enrollment Summaries			M	M(=)
Object Identifier			M	M(=)
Event Type			M	M(=)
Event State			M	M(=)
Priority			M	M(=)
Notification Class			U	U(=)
Result(-)			S	S(=)
Error type			M	M(=)

13.11.1.1 Argument

This parameter shall convey the parameters for the GetEnrollmentSummary confirmed service request.

13.11.1.1.1 Acknowledgment Filter

This parameter, of type ENUMERATED, shall provide a means of restricting the event-initiating objects that are to be summarized. The 'Acknowledgment Filter' may take any of three values:

ALL - Shall request that the returned summary contain all event-initiating objects without regard to whether the objects have acknowledgments or not.

ACKED - Shall request that the returned summary contain only those objects for which the Acked_Transitions property has a value of one in every bit position.

NOT-ACKED - Shall request that the returned summary contain only reports for those objects for which the Acked_Transitions property has a value of zero in one or more bit positions.

13.11.1.1.2 Enrollment Filter

This parameter, of type BACnetRecipientProcess, shall provide a means of restricting the set of objects that are to be summarized. Only those objects for which the specified BACnetRecipient and Process Identifier are enrolled to receive notifications, either confirmed or unconfirmed, shall be summarized. In this case, "enrolled" shall mean that an event-initiating object references a Notification Class object containing one or more BACnetDestinations containing the indicated Process Identifier and BACnetRecipient.

If this parameter is omitted, it shall mean that event-initiating objects shall be summarized without regard to enrollment status.

13.11.1.1.3 Event State Filter

This parameter shall provide a means of restricting the set of event-initiating objects that are to be summarized. It may have any of the following values:

{OFFNORMAL, FAULT, NORMAL, ALL, ACTIVE}.

Only those event-initiating objects whose Event_State property matches the value specified in this parameter shall be included. If the value ALL is specified, then all of the event-initiating objects shall be summarized without regard to the value of the Event_State property. If the value ACTIVE is specified, then only those event-initiating objects whose Event_State property has a value other than NORMAL shall be summarized. If this parameter is omitted, a default value of ALL shall be assumed.

13.11.1.1.4 Event Type Filter

This parameter is provided as a means of restricting the summary to only those event-initiating objects that can generate event notifications with an Event_Type equal to the value of this parameter. This parameter may have any legal value of Event_Type as defined in the Event Enrollment object specification. If this parameter is omitted, all event-initiating objects shall be included in the summary without regard to which EventTypes they generate.

13.11.1.1.5 Priority Filter

This parameter consists of two parts, MinPriority and MaxPriority, each of datatype Unsigned8. It provides a means of restricting the summary to only those event-initiating objects that can generate event notifications with a Priority as specified by this parameter. The 'Priority Filter' parameter consists of two parts, MinPriority and MaxPriority. All event-initiating objects, such that MinPriority ≤ Priority ≤ MaxPriority, shall be included in the summary. If 'Priority Filter' is omitted, all event-initiating objects shall be summarized without regard to their Priority.

13.11.1.1.6 Notification Class Filter

This parameter, of type Unsigned, provides a means of restricting the summary to only those event-initiating objects that can generate event notifications with a Notification Class equal to the value of this parameter. If 'Notification Class Filter' is omitted, it shall mean that all event-initiating objects objects shall be summarized without regard to their Notification Class.

13.11.1.2 Result(+)

The 'Result(+)' parameter shall indicate that the requested service has succeeded. A successful result includes the following parameters.

13.11.1.2.1 List of Enrollment Summaries

The 'List of Enrollment Summaries' shall contain zero or more Enrollment Summaries. Each Enrollment Summary shall consist of up to five parameters: 'Object Identifier', 'Event Type', 'Event State', 'Priority', and, optionally, 'Notification Class'. If the list is of length zero, then this shall be interpreted to mean that there are no event-initiating objects that meet the search criteria specified in the request primitive.

13.11.1.2.1.1 Object Identifier

This parameter, of type BACnetObjectIdentifier, shall identify an object meeting the search criteria.

13.11.1.2.1.2 Event Type

This parameter, of type BACnetEventType, indicates the Event_Type that the object can generate.

13.11.1.2.1.3 Event State

This parameter, of type BACnetEventState, indicates the value of the Event_State property of the object.

13.11.1.2.1.4 Priority

This parameter, of type Unsigned8, indicates the priority of notifications generated by the object.

13.11.1.2.1.5 Notification Class

This optional parameter, of type Unsigned, indicates the class of notifications generated by the object and implicitly refers to a Notification Class object that has a Notification_Class property of the same value.

13.11.1.3 Result(-)

The 'Result(-)' parameter shall indicate that the service request has failed. The reason for failure shall be specified by the 'Error Type' parameter.

13.11.1.3.1 Error Type

This parameter shall consist of two component parameters: (1) the 'Error Class' and (2) the 'Error Code'. See Clause 18.

13.11.2 Service Procedure

After verifying the validity of the request, the responding BACnet-user shall search for all event-initiating objects that meet the search criteria specified in the request primitive. The search criteria are the logical conjunctions of all of the explicitly stated filters and the default values for any filters that were omitted in the request primitive. A positive response containing the enrollment summaries for objects found in this search shall be constructed. If no objects are found that meet these criteria, then a list of length zero shall be returned.

13.12 GetEventInformation Service

The GetEventInformation service is used by a client BACnet-user to obtain a summary of all "active event states". The term "active event states" refers to all event-initiating objects that

(a) have an Event_State property whose value is not equal to NORMAL, or
(b) have an Acked_Transitions property, which has at least one of the bits (TO-OFFNORMAL, TO-FAULT, TO-NORMAL) set to FALSE.

This service is intended to be implemented in all devices that generate event notifications.

13.12.1 Structure

The structure of the GetEventInformation service primitives is shown in Table 13-13. The terminology and symbology used in this table are explained in 5.6.

Table 13-13. Structure of GetEventInformation Service Primitives

Parameter Name	Req	Ind	Rsp	Cnf
Argument	M	M(=)		
Last Received Object Identifier	U	U(=)		
Result(+)				
List of Event Summaries			M	M(=)
Object Identifier			M	M(=)
Event State			M	M(=)
Acknowledged Transitions			M	M(=)
Event Time Stamps			M	M(=)
Notify Type			M	M(=)
Event Enable			M	M(=)
Event Priorities			M	M(=)
More Events			M	M(=)
Result(-)			S	S(=)
Error Type			M	M(=)

13.12.1.1 Argument

This parameter indicates the GetEventInformation confirmed service request.

13.12.1.1.1 Last Received Object Identifier

This optional parameter, of type BACnetObjectIdentifier, shall specify the last Object Identifier received in a preceding GetEventInformation-ACK if its 'More Events' parameter was TRUE. If this parameter is omitted, the returned summary shall start with the first object meeting the "active event states" criteria. A fixed object processing order is assumed, however the particular order is a local matter. If the Last Received Object Identifier has become invalid in the responding device (i.e., the object is no longer present), the service shall resume if it is possible to determine the object that would have been the successor of the object that is no longer present. Otherwise a Result(-) shall be returned with an error class of OBJECT and an error code of UNKNOWN_OBJECT.

13.12.1.2 Result(+)

The 'Result(+)' parameter shall indicate that the requested service has succeeded. A successful result includes the following parameters.

13.12.1.2.1 List of Event Summaries

The 'List of Event Summaries' shall contain zero or more Event Summaries. Each Event Summary shall consist of seven parameters: 'Object Identifier', 'Event State', 'Acknowledged Transitions', 'Event Time Stamps', 'Notify Type', 'Event Enable' and 'Event Priorities'. If the list is of length zero, then this shall be interpreted to mean that there are no event-initiating objects that have active event states in this device.

13.12.1.2.1.1 Object Identifier

This parameter, of type BACnetObjectIdentifier, shall identify the event-initiating object that has an Event_State property whose value is not equal to NORMAL or has an Acked_Transitions property that has at least one of the following bits (TO-OFFNORMAL, TO-FAULT, TO-NORMAL) set to FALSE.

13.12.1.2.1.2 Event State

This parameter, of type BACnetEventState, indicates the value of the Event_State property of the object.

13.12.1.2.1.3 Acknowledged Transitions

This parameter, of type BACnetEventTransitionBits, indicates the value of the Acked_Transitions property of the object.

13.12.1.2.1.4 Event Time Stamps

This parameter, of type BACnetARRAY[3] of BACnetTimeStamp, shall convey the timestamps of the last event notifications for TO-OFFNORMAL, TO-FAULT, and TO-NORMAL events.

13.12.1.2.1.5 Notify Type

This parameter, of type BACnetNotifyType, shall convey the value of the Notify_Type property of this object.

13.12.1.2.1.6 Event Enable

This parameter, of type BACnetEventTransitionBits, shall convey the value of the Event_Enable property of the object.

13.12.1.2.1.7 Event Priorities

This parameter, of type BACnetARRAY[3] of Unsigned, shall convey the priorities specified in the Priority property of the associated Notification Class object. In the case where an Event Enrollment Object is used without an associated Notification Class Object, the three fields of this parameter shall all contain the value of the Priority property of the Event Enrollment Object.

13.12.1.2.2 More Events

This parameter, of type BOOLEAN, shall indicate whether (TRUE) or not (FALSE) more objects exist that meet the active event state criteria of the service request, but that could not be returned in the reply.

13.12.1.3 Result(-)

The 'Result(-)' parameter shall indicate that the service request has failed. The reason for failure shall be specified by the 'Error Type' parameter.

13.12.1.3.1 Error Type

This parameter shall consist of two component parameters: (1) the 'Error Class' and (2) the 'Error Code'. See Clause 18.

13.12.2 Service Procedure

After verifying the validity of the request, the responding BACnet-user shall search for all event-initiating objects that meet the following conditions, beginning with the object following (in whatever internal ordering of objects is used by the responding device) the object specified by the 'Last Received Object Identifier' parameter, if present:

(a) have an Event_State property whose value is not equal to NORMAL, or
(b) have an Acked_Transitions property that has at least one of the following bits (TO-OFFNORMAL, TO-FAULT, TO-NORMAL) set to FALSE.

A positive response containing the event summaries for objects found in this search shall be constructed. If no objects are found that meet these criteria, then a list of length zero shall be returned. As many of the included objects as can be returned within the APDU shall be returned. If more objects exist that meet the criteria but cannot be returned in the APDU, the 'More Events' parameter shall be set to TRUE, otherwise it shall be set to FALSE.

13.13 LifeSafetyOperation Service

The LifeSafetyOperation service is intended for use in fire, life safety and security systems to provide a mechanism for conveying specific instructions from a human operator to accomplish any of the following objectives:

(a) silence audible or visual notification appliances,
(b) reset latched notification appliances, or
(c) unsilence previously silenced audible or visual notification appliances.

Ensuring that the LifeSafetyOperation request actually comes from a person with appropriate authority is a local matter.

13.13.1 Structure

The structure of the LifeSafetyOperation primitive is shown in Table 13-14. The terminology and symbology used in this table are explained in 5.6.

Table 13-14. Structure of LifeSafetyOperation Service Primitives

Parameter Name	Req	Ind	Rsp	Cnf
Argument	M	M(=)		
Requesting Process Identifier	M	M(=)		
Requesting Source	M	M(=)		
Request	M	M(=)		
Object Identifier	U	U(=)		
Result(+)			S	S(=)
Result(-)			S	S(=)
Error Type			M	M(=)

13.13.1.1 Argument

This parameter shall convey the parameters for the LifeSafetyOperation confirmed service request.

13.13.1.1.1 Requesting Process Identifier

This parameter, of type Unsigned32, specifies an identifying number of significance to the sending device that uniquely identifies the process which initiated the service request. The assignment and meaning of process identifiers shall be a local matter.

13.13.1.1.2 Requesting Source

This parameter, of type CharacterString, specifies the identity of the human operator that initiated the LifeSafetyOperation service request.

13.13.1.1.3 Request

This parameter, of type BACnetLifeSafetyOperation, shall convey the requested operation:

{SILENCE, SILENCE_AUDIBLE, SILENCE_VISUAL, RESET, RESET_ALARM, RESET_FAULT, UNSILENCE, UNSILENCE_AUDIBLE, UNSILENCE_VISUAL}

13.13.1.1.4 Object Identifier

This parameter, of type BACnetObjectIdentifier, shall convey the specific BACnet object to which the life safety request is directed. If this parameter is not present, then all applicable objects within the receiving BACnet device shall be silenced or reset accordingly based on the 'Request' provided.

13.13.1.2 Result(+)

The 'Result(+)' parameter shall indicate that the service request succeeded.

13.13.1.3 Result(-)

The 'Result(-)' parameter shall indicate that the service request has failed. The reason for the failure shall be specified by the 'Error Type' parameter.

13.13.1.3.1 Error Type

This parameter consists of two component parameters: (1) the 'Error Class' and (2) the 'Error Code'. See Clause 18.

13.13.2 Service Procedure

The responding BACnet-user shall first verify the validity of the 'Object Identifier' parameter and return a 'Result(-)' response with the appropriate error class and code if the 'Request' is invalid or if the 'Object Identifier' parameter is present and specifies an object that is either unknown or does not represent an appropriate request for this object type.

If the 'Object Identifier' parameter is not present, then the responding BACnet-user shall attempt to operate all applicable objects in the device based on the 'Request' parameter. If the 'Object Identifier' parameter is present, the responding BACnet-user shall attempt to silence or reset the object specified in the 'Object Identifier' parameter based on the 'Request' parameter. In either case, the responding BACnet-user shall issue a Result(+) primitive.

13.14 SubscribeCOV Service

The SubscribeCOV service is used by a COV-client to subscribe for the receipt of notifications of changes that may occur to the properties of a particular object. Certain BACnet standard objects may optionally support COV reporting. If a standard object provides COV reporting, then changes of value of specific properties of the object, in some cases based on programmable increments, trigger COV notifications to be sent to one or more subscriber clients. Typically, COV notifications are sent to supervisory programs in BACnet client devices or to operators or logging devices. Proprietary objects may support COV reporting at the implementor's option. The standardized objects that may optionally provide COV support and the change of value algorithms they shall employ are summarized in Table 13-1.

The subscription establishes a connection between the change of value detection and reporting mechanism within the COV-server device and a "process" within the COV-client device. Notifications of changes are issued by the COV-server device when changes occur after the subscription has been established. The ConfirmedCOVNotification and UnconfirmedCOVNotification services are used by the COV-server device to convey change notifications. The choice of confirmed or unconfirmed service is made at the time the subscription is established.

13.14.1 Structure

The structure of the SubscribeCOV service primitives is shown in Table 13-15. The terminology and symbology used in this table are explained in 5.6.

Table 13-15. Structure of SubscribeCOV Service Primitives

Parameter Name	Req	Ind	Rsp	Cnf
Argument	M	M(=)		
Subscriber Process Identifier	M	M(=)		
Monitored Object Identifier	M	M(=)		
Issue Confirmed Notifications	U	U(=)		
Lifetime	U	U(=)		
Result(+)			S	S(=)
Result(-)			S	S(=)
Error Type			M	M(=)

13.14.1.1 Argument

This parameter shall convey the parameters for the SubscribeCOV confirmed service request.

13.14.1.2 Subscriber Process Identifier

This parameter, of type Unsigned32, shall convey a numeric "handle" meaningful to the subscriber. This handle shall be used to match future re-subscriptions and cancellations from the subscriber with the COV context that exists within the COV-server device and with confirmed or unconfirmed COV notifications to identify the process within the COV-client that should receive them. The value zero is reserved for unsubscribed COV notifications as described in 13.7.

13.14.1.3 Monitored Object Identifier

This parameter, of type BACnetObjectIdentifier, shall convey the identifier of the object within the receiving device for which a subscription is desired.

13.14.1.4 Issue Confirmed Notifications

This parameter, of type BOOLEAN, shall convey whether the COV-server device shall issue ConfirmedCOVNotifications (TRUE) or UnconfirmedCOVNotifications (FALSE) when changes occur. This parameter, if present, shall indicate a subscription or re-subscription is to occur and that the lifetime shall be refreshed to its initial state. If both the 'Issue Confirmed Notifications' and 'Lifetime' parameters are absent, then this shall indicate a cancellation request. If the 'Lifetime' parameter is present then the 'Issue Confirmed Notifications' parameter shall be present.

13.14.1.5 Lifetime

This parameter, of type Unsigned, shall convey the desired lifetime of the subscription in seconds. A value of zero shall indicate an indefinite lifetime, without automatic cancellation. A non-zero value shall indicate the number of seconds that may elapse before the subscription shall be automatically cancelled. If both the 'Issue Confirmed Notifications' and 'Lifetime' parameters are absent, then this shall indicate a cancellation request. If the 'Lifetime' parameter is present then the 'Issue Confirmed Notifications' parameter shall be present.

13.14.1.6 Result(+)

The 'Result(+)' parameter shall indicate that the requested service has succeeded.

13.14.1.7 Result(-)

The 'Result(-)' parameter shall indicate that the service request has failed. The reason for failure shall be specified by the 'Error Type' parameter.

13.14.1.7.1 Error Type

This parameter shall consist of two component parameters: (1) the 'Error Class' and (2) the 'Error Code'. See Clause 18.

13.14.2 Service Procedure

If neither 'Lifetime' nor 'Issue Confirmed Notifications' are present, then the request shall be considered to be a cancellation. Any COV context that already exists for the same BACnet address contained in the PDU that carries the SubscribeCOV request and has the same 'Subscriber Process Identifier' and 'Monitored Object Identifier' shall be disabled and a 'Result(+)' returned. Cancellations that are issued for which no matching COV context can be found shall succeed as if a context had existed, returning 'Result(+)'.

If the 'Lifetime' parameter is present and has a non-zero value but the device does not support automatic cancellation of subscriptions, then a 'Result(-)' shall be returned. If the 'Lifetime' parameter is not present but the 'Issue Confirmed Notifications' parameter is present, then a value of zero (indefinite lifetime) shall be assumed for the lifetime. If the 'Issue Confirmed Notifications' parameter is present but the object to be monitored does not support COV reporting, then a 'Result(-)' shall be returned. If the object to be monitored does support COV reporting, then a check shall be made to locate an existing COV context for the same BACnet address contained in the PDU that carries the SubscribeCOV request and has the same 'Subscriber Process Identifier' and 'Monitored Object Identifier'. If an existing COV context is found, then the request shall be considered a re-subscription and shall succeed as if the subscription had been newly created.

If no COV context can be found that matches the request, then a new COV context shall be established that contains the BACnet address from the PDU that carries the SubscribeCOV request and the same 'Subscriber Process Identifier' and 'Monitored Object Identifier'. If no context can be created, then a 'Result(-)' shall be returned.

If a new context is created, or a re-subscription is received, then the COV context shall be initialized and given a lifetime as specified by the 'Lifetime' parameter, if present, or zero if the 'Lifetime' parameter is not present. The subscription shall be automatically cancelled after that many seconds have elapsed unless a re-subscription is received. A lifetime of zero shall indicate that the subscription is indefinite and no automatic cancellation shall occur. In either case, a 'Result(+)' shall be returned. A ConfirmedCOVNotification or UnconfirmedCOVNotification shall be issued as soon as possible after the successful completion of a subscription or re-subscription request, as specified by the 'Issue Confirmed Notifications' parameter.

13.15 SubscribeCOVProperty Service

The SubscribeCOVProperty service is used by a COV-client to subscribe for the receipt of notifications of changes that may occur to the properties of a particular object. Any object may optionally support COV reporting. If a standard object provides COV reporting, then changes of value of subscribed-to properties of the object, in some cases based on programmable increments, trigger COV notifications to be sent to one or more subscriber clients. Typically, COV notifications are sent to supervisory programs in BACnet client devices or to operators or logging devices.

The subscription establishes a connection between the change of value detection and reporting mechanism within the COV-server device and a "process" within the COV-client device. Notifications of changes are issued by the COV-server device when changes occur after the subscription has been established. The ConfirmedCOVNotification and UnconfirmedCOVNotification services are used by the COV-server device to convey change notifications. The choice of confirmed or unconfirmed service is made at the time the subscription is established. Any object, proprietary or standard, may support COV reporting for any property at the implementor's option.

The SubscribeCOVProperty service differs from the SubscribeCOV service in that it allows monitoring of properties other than those listed in Table 13-1.

13.15.1 Structure

The structure of the SubscribeCOVProperty service primitives is shown in Table 13-16. The terminology and symbology used in this table are explained in 5.6.

Table 13-16. Structure of SubscribeCOVProperty Service Primitives

Parameter Name	Req	Ind	Rsp	Cnf
Argument	M	M(=)		
Subscriber Process Identifier	M	M(=)		
Monitored Object Identifier	M	M(=)		
Issue Confirmed Notifications	U	U(=)		
Lifetime	U	U(=)		
Monitored Property Identifier	M	M(=)		
COV Increment	U	U(=)		
Result(+)			S	S(=)
Result(-)			S	S(=)
Error Type			M	M(=)

13.15.1.1 Argument

This parameter shall convey the parameters for the SubscribeCOVProperty confirmed service request.

13.15.1.2 Subscriber Process Identifier

This parameter, of type Unsigned32, shall convey a numeric "handle" meaningful to the subscriber. This handle shall be used to match future re-subscriptions and cancellations from the subscriber with the COV context that exists within the COV-server device and with confirmed or unconfirmed COV notifications to identify the process within the COV-client that should receive them.

13.15.1.3 Monitored Object Identifier

This parameter, of type BACnetObjectIdentifier, shall convey the identifier of the object within the receiving device that contains the property for which a subscription is desired.

13.15.1.4 Issue Confirmed Notifications

This parameter, of type BOOLEAN, shall convey whether the COV-server device shall issue ConfirmedCOVNotifications (TRUE) or UnconfirmedCOVNotifications (FALSE) when changes occur. This parameter, if present, shall indicate that a subscription or re-subscription is to occur and that the lifetime shall be refreshed to its initial state. If both the 'Issue

Confirmed Notifications' and 'Lifetime' parameters are absent, then this shall indicate a cancellation request. If the 'Lifetime' parameter is present then the 'Issue Confirmed Notifications' parameter shall be present.

13.15.1.5 Lifetime

This parameter, of type Unsigned, shall convey the desired lifetime of the subscription in seconds. A value of zero shall not be allowed. A non-zero value shall indicate the number of seconds that may elapse before the subscription shall be automatically cancelled. If both the 'Issue Confirmed Notifications' and 'Lifetime' parameters are absent, then this shall indicate a cancellation request. If the 'Issue Confirmed Notifications' parameter is present then the 'Lifetime' parameter shall be present.

13.15.1.6 Monitored Property Identifier

This parameter, of type BACnetPropertyReference, shall convey the property identifier and optional array index for which a subscription is desired. If COV reporting is supported for a property that has an array datatype, it is a local matter to determine whether to support COV subscriptions for all elements of the array or only for particular elements in the array.

13.15.1.7 COV Increment

This parameter, of type REAL, shall specify the minimum change in the monitored property that will cause a COVNotification to be issued to subscriber COV-clients. This parameter is ignored if the datatype of the monitored property is not REAL. If the monitored property is Present_Value, its datatype is REAL, this parameter is not present, and the monitored object has a COV_Increment property, then the COV increment to use is taken from the COV_Increment property of the monitored object. Otherwise, the COV increment is a local matter. The intent is to allow the subscriber to use a previously established COV increment from another subscription or to allow use of the COV_Increment property in the monitored object.

13.15.1.8 Result(+)

The 'Result(+)' parameter shall indicate that the requested service has succeeded.

13.15.1.9 Result(-)

The 'Result(-)' parameter shall indicate that the service request has failed. The reason for failure shall be specified by the 'Error Type' parameter.

13.15.1.9.1 Error Type

This parameter shall consist of two component parameters: (1) the 'Error Class' and (2) the 'Error Code'. See Clause 18.

13.15.2 Service Procedure

If neither 'Lifetime' nor 'Issue Confirmed Notifications' are present, then the request shall be considered to be a cancellation. Any COV context that already exists for the same BACnet address contained in the PDU that carries the SubscribeCOVProperty request and has the same 'Subscriber Process Identifier', 'Monitored Object Identifier' and 'Monitored Property Identifier' shall be disabled and a 'Result(+)' returned. Cancellations that are issued for which no matching COV context can be found shall succeed as if a context had existed, returning 'Result(+)'. If an existing COV context is found, it shall be removed from the Active_COV_Subscriptions property in the Device object.

If the 'Issue Confirmed Notifications' parameter is present but the property to be monitored does not support COV reporting, then a 'Result(-)' shall be returned. If the property to be monitored does support COV reporting, then a check shall be made to locate an existing COV context for the same BACnet address contained in the PDU that carries the SubscribeCOVProperty request and has the same 'Subscriber Process Identifier', 'Monitored Object Identifier' and 'Monitored Property Identifier'. If an existing COV context is found, then the request shall be considered a re-subscription and shall succeed as if the subscription had been newly created.

If no COV context can be found that matches the request, then a new COV context shall be established that contains the BACnet address from the PDU that carries the SubscribeCOVProperty request and the same 'Subscriber Process Identifier', 'Monitored Object Identifier' and 'Monitored Property Identifier'. The new context shall be included in the Active_COV_Subscriptions property of the Device object. If no context can be created, then a 'Result(-)' shall be returned.

If a new context is created, or a re-subscription is received, then the COV context shall be initialized and given a lifetime as specified by the 'Lifetime' parameter. The subscription shall be automatically cancelled after that many seconds have elapsed

unless a re-subscription is received. A 'Result(+)' shall be returned and a ConfirmedCOVNotification or UnconfirmedCOVNotification shall be issued as soon as possible after the successful completion of a subscription or re-subscription request, as specified by the 'Issue Confirmed Notifications' parameter.

14 FILE ACCESS SERVICES

This clause defines the set of services used to access and manipulate files contained in BACnet devices. The concept of files is used here as a network-visible representation for a collection of octets of arbitrary length and meaning. This is an abstract concept only and does not imply the use of disk, tape or other mass storage devices in the server devices. These services may be used to access vendor-defined files as well as specific files defined in the BACnet protocol standard.

Every file that is accessible by File Access Services shall have a corresponding File object in the BACnet device. This File object is used to identify the particular file by name. In addition, the File object provides access to "header information," such as the file's total size, creation date, and type. File Access Services may model files in two ways: as a continuous stream of octets or as a contiguous sequence of numbered records.

The File Access Services provide atomic read and write operations. In this context "atomic" means that during the execution of a read or write operation, no other AtomicReadFile or AtomicWriteFile operations are allowed for the same file. Synchronization of these services with internal operations of the BACnet device is a local matter and is not defined by this standard.

14.1 AtomicReadFile Service

The AtomicReadFile Service is used by a client BACnet-user to perform an open-read-close operation on the contents of the specified file. The file may be accessed as records or as a stream of octets.

14.1.1 Structure

The structure of the AtomicReadFile service primitives is shown in Table 14-1. The terminology and symbology used in this table are explained in 5.6.

Table 14-1. Structure of AtomicReadFile Service Primitives

Parameter Name	Req	Ind	Rsp	Cnf
Argument	M	M(=)		
File Identifier	M	M(=)		
Stream Access	S	S(=)		
File Start Position	M	M(=)		
Requested Octet Count	M	M(=)		
Record Access	S	S(=)		
File Start Record	M	M(=)		
Requested Record Count	M	M(=)		
Result(+)			S	S(=)
End Of File			M	M(=)
Stream Access			S	S(=)
File Start Position			M	M(=)
File Data			C	C(=)
Record Access			S	S(=)
File Start Record			M	M(=)
Returned Record Count			M	M(=)
File Record Data			C	C(=)
Result(-)			S	S(=)
Error Type			M	M(=)

14.1.2 Argument

This parameter shall convey the parameters for the AtomicReadFile confirmed service request.

14.1.2.1 File Identifier

This parameter is the Object_Identifier of the File object that identifies the file to be read.

14.1.2.2 Stream Access

The 'Stream Access' parameter shall indicate that stream-oriented file access is required. Stream access includes the parameters 'File Start Position' and 'Requested Octet Count'.

14.1.2.2.1 File Start Position

This parameter, of type INTEGER, represents the number of octets from the beginning of the file at which reading shall commence. A 'File Start Position' of 0 is the first octet of the file.

14.1.2.2.2 Requested Octet Count

This parameter, of type Unsigned, represents the number of octets that shall be read from the file starting at the 'File Start Position'.

14.1.2.3 Record Access

The 'Record Access' parameter shall indicate that record-oriented file access is required. Record access includes the parameters 'File Start Record' and 'Requested Record Count'.

14.1.2.3.1 File Start Record

This parameter, of type INTEGER, represents the number of records from the beginning of the file at which reading shall commence. A 'File Start Record' of 0 is the first record of the file.

14.1.2.3.2 Requested Record Count

This parameter, of type Unsigned, represents the number of records that shall be read from the file starting at the 'File Start Record'.

14.1.3 Result(+)

The 'Result(+)' parameter shall indicate that the service request succeeded. A successful result includes the following parameters.

14.1.3.1 End Of File

The 'End Of File' parameter, of type BOOLEAN, shall be equal to TRUE if this response includes the last octet of the file and FALSE otherwise. This parameter shall be used to check for the end of file since the number of octets returned could be less than the 'Requested Octet Count' or the 'Returned Record Count' could be less than the 'Requested Record Count' due to the amount of data remaining in the file. This parameter also provides a data-independent way for the client user of this service to detect an end of file.

14.1.3.2 Stream Access

The 'Stream Access' parameter shall indicate that stream-oriented file access was requested. Stream access includes the parameters 'File Start Position' and 'File Data'.

14.1.3.2.1 File Start Position

This parameter, of type INTEGER, represents the number of octets from the beginning of the file from which the start of the data was read. A 'File Start Position' of 0 is the first octet of the file.

14.1.3.2.2 File Data

This parameter consists of an OCTET STRING that contains the requested file data.

14.1.3.3 Record Access

The 'Record Access' parameter shall indicate that record-oriented file access was requested. Record access includes the parameters 'File Start Record', 'Returned Record Count', and 'File Record Data'.

14.1.3.3.1 File Start Record

This parameter, of type INTEGER, represents the number of records from the beginning of the file from which the start of the data was read. A 'File Start Record' of 0 is the first record of the file.

14.1.3.3.2 Returned Record Count

This parameter, of type Unsigned, represents the number of records that were actually read from the file, which may be less than the 'Requested Record Count'.

14.1.3.3.3 File Record Data

This parameter consists of a List of OCTET STRINGs that contain the requested file data.

14.1.4 Result(-)

The 'Result(-)' parameter shall indicate that the service request has failed in its entirety. The reason for the failure shall be specified by the 'Error Type' parameter.

14.1.4.1 Error Type

This parameter consists of two component parameters: (1) the 'Error Class' and (2) the 'Error Code'. See Clause 18.

14.1.5 Service Procedure

The responding BACnet-user shall first verify the validity of the 'File Identifier' parameter and return a 'Result(-)' response with the appropriate error class and code if the File object is unknown, if there is currently another AtomicReadFile or

AtomicWriteFile service in progress, or if the File object is currently inaccessible for another reason. If the 'File Start Position' parameter or the 'File Start Record' parameter is either less than 0 or exceeds the actual file size, then the appropriate error is returned in a 'Result(-)' response. If not, then the responding BACnet-user shall read the number of octets specified by 'Requested Octet Count' or the number of records specified by 'Requested Record Count'. If the number of remaining octets or records is less than the requested amount, then the length of the 'File Data' returned or 'Returned Record Count' shall indicate the actual number read. If the returned response contains the last octet or record of the file, then the 'End Of File' parameter shall be TRUE, otherwise FALSE.

14.2 AtomicWriteFile Service

The AtomicWriteFile Service is used by a client BACnet-user to perform an open-write-close operation of an OCTET STRING into a specified position or a List of OCTET STRINGs into a specified group of records in a file. The file may be accessed as records or as a stream of octets.

14.2.1 Structure

The structure of the AtomicWriteFile service primitives is shown in Table 14-2. The terminology and symbology used in this table are explained in 5.6.

Table 14-2. Structure of AtomicWriteFile Service Primitives

Parameter Name	Req	Ind	Rsp	Cnf
Argument	M	M(=)		
File Identifier	M	M(=)		
Stream Access	S	S(=)		
File Start Position	M	M(=)		
File Data	M	M(=)		
Record Access	S	S(=)		
File Start Record	M	M(=)		
Record Count	M	M(=)		
File Record Data	M	M(=)		
Result(+)			S	S(=)
Stream Access			S	S(=)
File Start Position			M	M(=)
Record Access			S	S(=)
File Start Record			M	M(=)
Result(-)			S	S(=)
Error Type			M	M(=)

14.2.2 Argument

This parameter shall convey the parameters for the AtomicWriteFile confirmed service request.

14.2.2.1 File Identifier

This parameter is the Object_Identifier of the File object that identifies the file to be written.

14.2.2.2 Stream Access

The 'Stream Access' parameter shall indicate that stream-oriented file access is required. Stream access includes the parameters 'File Start Position' and 'File Data'.

14.2.2.2.1 File Start Position

This parameter, of type INTEGER, represents the number of octets from the beginning of the file at which the data shall start being written. A 'File Start Position' of 0 is the first octet of the file. A 'File Start Position' of -1 shall indicate the end of the current file, i.e., an append to file operation.

14.2.2.2.2 File Data

This parameter consists of an OCTET STRING that is to be written to the file.

14.2.2.3 Record Access

The 'Record Access' parameter shall indicate that record-oriented file access is required. Record access includes the parameters 'File Start Record', 'Record Count', and 'File Record Data'.

14.2.2.3.1 File Start Record

This parameter, of type INTEGER, represents the number of records from the beginning of the file at which the data shall start being written. A 'File Start Record' of 0 is the first record of the file. A 'File Start Record' of -1 shall indicate the end of the current file, i.e. an append to file operation.

14.2.2.3.2 Record Count

This parameter, of type Unsigned, represents the number of records that shall be written to the file starting at the 'File Start Record'.

14.2.2.3.3 File Record Data

This parameter consists of a List of OCTET STRINGs that is to be written to the file.

14.2.3 Result(+)

The 'Result(+)' parameter shall indicate that the service request succeeded. A successful result shall include the following parameters.

14.2.3.1 Stream Access

The 'Stream Access' parameter shall indicate that stream-oriented file access was requested. Stream access includes the 'File Start Position' parameter. The 'File Start Position' parameter, of type INTEGER, represents the number of octets from the beginning of the file where the data were actually written. A 'File Start Position' of 0 is the first octet of the file.

14.2.3.2 Record Access

The 'Record Access' parameter shall indicate that record-oriented file access was requested. Record access includes the 'File Start Record' parameter. The 'File Start Record' parameter, of type INTEGER, represents the number of records from the beginning of the file where the data were actually written. A 'File Start Record' of 0 is the first record of the file.

14.2.4 Result(-)

The 'Result(-)' parameter shall indicate that the service request has failed in its entirety. The reason for the failure shall be specified by the 'Error Type' parameter.

14.2.4.1 Error Type

This parameter consists of two component parameters: (1) the 'Error Class' and (2) the 'Error Code'. See Clause 18.

14.2.5 Service Procedure

The responding BACnet-user shall first verify the validity of the 'File Identifier' parameter and return a 'Result(-)' response with the appropriate error class and code if the File object is unknown, if there is currently another AtomicReadFile or AtomicWriteFile service in progress, or if the File object is currently inaccessible for another reason. If the 'File Start Position' parameter or the 'File Start Record' parameter exceeds the actual file size, then the file shall be extended to the size indicated, but the contents of any intervening octets or records shall be a local matter. If either of these parameters has the special value -1, then the write operation shall be treated as an append to the current end of file. Then the responding BACnet-user shall write the number of octets specified by 'Octet Count' or the number of records specified by 'Record Count' to the file. If the write fails for any reason, then a 'Result(-)' response with the appropriate error class and code shall be returned. If the write succeeds in its entirety, then a 'Result(+)' response shall be returned. The 'File Start Position' or 'File Start Record' shall indicate the actual position or record at which data were written.

15 OBJECT ACCESS SERVICES

This clause defines nine application services that collectively provide the means to access and manipulate the properties of BACnet objects. A BACnet object is any object whose properties are accessible through this protocol regardless of its particular function within the device in which it resides. These services may be used to access the properties of vendor-defined objects as well as those of objects specified in this standard.

15.1 AddListElement Service

The AddListElement service is used by a client BACnet-user to add one or more list elements to an object property that is a list.

15.1.1 Structure

The structure of the AddListElement service primitives is shown in Table 15-1. The terminology and symbology used in this table are explained in 5.6.

Table 15-1. Structure of AddListElement Service Primitives

Parameter Name	Req	Ind	Rsp	Cnf
Argument	M	M(=)		
Object Identifier	M	M(=)		
Property Identifier	M	M(=)		
Property Array Index	C	C(=)		
List of Elements	M	M(=)		
Result(+)			S	S(=)
Result(-)			S	S(=)
Error Type			M	M(=)
First Failed Element Number			M	M(=)

15.1.1.1 Argument

This parameter shall convey the parameters for the AddListElement confirmed service request.

15.1.1.1.1 Object Identifier

This parameter, of type BACnetObjectIdentifier, shall provide the means of identifying the object whose specified list property is to be modified by this service.

15.1.1.1.2 Property Identifier

This parameter, of type BACnetPropertyIdentifier, shall provide the means of uniquely identifying the property to be modified by this service.

15.1.1.1.3 Property Array Index

If the property identified above is of datatype array, this conditional parameter of type Unsigned shall be present and shall indicate the array index of the element of the referenced property to be modified by this service. Otherwise, it shall be omitted.

15.1.1.1.4 List of Elements

This parameter specifies one or more elements that shall be added to the property specified by the 'Property Identifier' parameter. The datatype of the elements of this parameter is determined by the definition of the object type for the object specified by the 'Object Identifier' parameter.

15.1.1.2 Result(+)

The 'Result(+)' parameter shall indicate that the service request succeeded and all of the specified elements were added to the list.

15.1.1.3 Result(-)

The 'Result(-)' parameter shall indicate that the service request failed and none of the specified elements were added to the list. The reason for failure is specified by the 'Error Type' parameter.

15.1.1.3.1 Error Type

This parameter consists of two component parameters: (1) an 'Error Class' and (2) an 'Error Code'. See Clause 18.

15.1.1.3.2 First Failed Element Number

This parameter, of type Unsigned, shall convey the numerical position, starting at 1, of the offending element in the 'List of Elements' parameter received in the request. If the request is considered invalid for reasons other than the 'List of Elements' parameter, the 'First Failed Element Number' shall be equal to zero.

15.1.2 Service Procedure

After verifying the validity of the request, the responding BACnet-user shall attempt to modify the object identified in the 'Object Identifier' parameter. If the identified object exists and has the property specified in the 'Property Identifier' parameter, an attempt shall be made to add all of the elements specified in the 'List of Elements' parameter to the specified property. If this attempt is successful, a 'Result(+)' primitive shall be issued. If one or more of the elements is already present in the list, it shall be ignored, that is, shall not be added to the list. Ignoring an element that already exists shall not cause the service to fail.

If the specified object does not exist, the specified property does not exist, or the specified property is not a list, then the service shall fail and a 'Result(-)' response primitive shall be issued. If one or more elements cannot be added to the list and they are not already members, a 'Result(-)' response primitive shall be issued and no elements shall be added to the list.

The effect of this service shall be to add to the list all of the specified elements that are not already present or to add no elements to the list at all.

15.2 RemoveListElement Service

The RemoveListElement service is used by a client BACnet-user to remove one or more elements from the property of an object that is a list. If an element is itself a list, the entire element shall be removed. This service does not operate on nested lists.

15.2.1 Structure

The structure of the RemoveListElement service primitives is shown in Table 15-2. The terminology and symbology used in this table are explained in 5.6.

Table 15-2. Structure of RemoveListElement Service Primitives

Parameter Name	Req	Ind	Rsp	Cnf
Argument	M	M(=)		
Object Identifier	M	M(=)		
Property Identifier	M	M(=)		
Property Array Index	C	C(=)		
List of Elements	M	M(=)		
Result (+)			S	S(=)
Result (-)			S	S(=)
Error Type			M	M(=)
First Failed Element Number			M	M(=)

15.2.1.1 Argument

This parameter shall convey the parameters for the RemoveListElement confirmed service request.

15.2.1.1.1 Object Identifier

This parameter, of type BACnetObjectIdentifier, shall provide the means of identifying the object whose specified list property is to be modified by this service.

15.2.1.1.2 Property Identifier

This parameter, of type BACnetPropertyIdentifier, shall provide the means of uniquely identifying the property to be modified by this service.

15.2.1.1.3 Property Array Index

If the property identified above is of datatype array, this conditional parameter of type Unsigned shall be present and shall indicate the array index of the element of the referenced property to be modified by this service. Otherwise, it shall be omitted.

15.2.1.1.4 List of Elements

This parameter specifies one or more elements that shall be removed from the property specified in the 'Property Identifier' parameter. The datatype of the elements of this parameter is determined by the definition of the object type for the object specified by the 'Object Identifier' parameter.

15.2.1.2 Result(+)

The 'Result(+)' parameter shall indicate that the service request succeeded and all of the specified elements have been removed.

15.2.1.3 Result(-)

The 'Result(-)' parameter shall indicate that the service request failed. The reason for failure is specified by the 'Error Type' parameter. None of the elements of the specified object shall be removed.

15.2.1.3.1 Error Type

This parameter consists of two component parameters: (1) an 'Error Class' and (2) an 'Error Code'. See Clause 18.

15.2.1.3.2 First Failed Element Number

This parameter, of type Unsigned, shall convey the numerical position, starting at 1, of the offending element in the 'List of Elements' parameter received in the request. If the request is considered invalid for reasons other than the 'List of Elements' parameter, the 'First Failed Element Number' shall be equal to zero.

15.2.2 Service Procedure

After verifying the validity of the request, the responding BACnet-user shall attempt to modify the object identified in the 'Object Identifier' parameter. If the identified object exists and it has the property specified in the 'Property Identifier' parameter, an attempt shall be made to remove the elements in the 'List of Elements' from the property of the object. If one or more of the elements does not exist or cannot be removed because of insufficient authority, none of the elements shall be removed and a 'Result(-)' response primitive shall be issued.

15.3 CreateObject Service

The CreateObject service is used by a client BACnet-user to create a new instance of an object. This service may be used to create instances of both standard and vendor specific objects. The standard object types supported by this service shall be specified in the PICS. The properties of standard objects created with this service may be initialized in two ways: initial values may be provided as part of the CreateObject service request or values may be written to the newly created object using the BACnet WriteProperty services. The initialization of non-standard objects is a local matter. The behavior of objects created by this service that are not supplied, or only partially supplied, with initial property values is dependent upon the device and is a local matter.

15.3.1 Structure

The structure of the CreateObject service primitives is shown in Table 15-3. The terminology and symbology used in this table are explained in 5.6.

Table 15-3. Structure of CreateObject Service Primitives

Parameter Name	Req	Ind	Rsp	Cnf
Argument	M	M(=)		
Object Specifier	M	M(=)		
List of Initial Values	U	U(=)		
Result(+)			S	S(=)
Object Identifier			M	M(=)
Result(-)			S	S(=)
Error Type			M	M(=)
First Failed Element Number			M	M(=)

15.3.1.1 Argument

This parameter shall convey the parameters for the CreateObject confirmed service request.

15.3.1.1.1 Object Specifier

This parameter shall convey information about the type of object that is to be created. The datatype is a choice between an object type and an object identifier. If the object type choice is used, the specified object type shall become the value of the Object_Type property of the newly created object and the responding BACnet-user shall select an object identifier. If the object identifier choice is used, an object with this particular object identifier shall be created.

15.3.1.1.2 List of Initial Values

This parameter shall convey a list of BACnetPropertyValues that shall be used to initialize the values of the specified properties of the newly created object.

15.3.1.2 Result(+)

The 'Result(+)' parameter shall indicate that the service request succeeded. A success includes successfully initializing all the properties specified in the 'List of Initial Values' parameter. The 'Result(+)' shall convey as a parameter an 'Object Identifier', which is the value of the Object_Identifier property of the newly created object. This identifier shall be unique within the server device.

15.3.1.3 Result(-)

The 'Result(-)' parameter shall indicate that the service request failed. The reason for failure is specified by the 'Error Type' parameter.

15.3.1.3.1 Error Type

This parameter consists of two component parameters: (1) an 'Error Class' and (2) an 'Error Code'. See Clause 18. The 'Error Class' and 'Error Code' to be returned for specific situations are as follows:

Situation	Error Class	Error Code
The device cannot allocate the space needed for the new object.	RESOURCES	NO_SPACE_FOR_OBJECT
The device does not support creation of this object for any reason other than space.	OBJECT	DYNAMIC_CREATION_NOT_SUPPORTED
The object being created already exists.	OBJECT	OBJECT_IDENTIFIER_ALREADY_EXISTS
A datatype of a property value specified in the List of Initial Values does not match the datatype of the property specified by the Property_Identifier.	PROPERTY	INVALID_DATATYPE
A value used in the List of Initial Values is outside the range of values defined for the property specified by the Property_Identifier.	PROPERTY	VALUE_OUT_OF_RANGE
A Property_Identifier has been specified in the List of Initial Values that is unknown for objects of the type being created.	PROPERTY	UNKNOWN_PROPERTY
A character string value was encountered in the List of Initial Values that is not a supported character set.	PROPERTY	CHARACTER_SET_NOT_SUPPORTED
A property specified by the Property_Identifier in the List of Initial Values does not support initialization during the CreateObject service.	PROPERTY	WRITE_ACCESS_DENIED

15.3.1.3.2 First Failed Element Number

This parameter, of type Unsigned, shall convey the numerical position, starting at 1, of the offending 'Initial Value' in the 'List of Initial Values' parameter received in the request. If the request is considered invalid for reasons other than the 'List of Initial Values' parameter, the 'First Failed Element Number' shall be equal to zero.

15.3.2 Service Procedure

After verifying the validity of the request, the responding BACnet-user shall attempt to create a new object of the type specified in the 'Object Specifier' parameter.

If the 'Object Specifier' parameter contains an object type, the Object_Identifier property of the newly created object shall be initialized to a value that is unique within the responding BACnet-user device. The method used to generate the object identifier is a local matter. The Object_Type property shall be initialized to the value of the 'Object Specifier' parameter. If a new object of the specified type cannot be created, a 'Result(-)' primitive shall be returned and the 'First Failed Element Number' parameter shall have a value of zero.

If the 'Object Specifier' parameter contains an object identifier, the responding BACnet-user shall determine if an object with this identifier already exists. If such an object exists, then a new object shall not be created, and a 'Result(-)' primitive shall be returned and the 'First Failed Element Number' parameter shall have a value of zero. If such an object does not exist and it cannot be created, a 'Result(-)' primitive shall be returned and the 'First Failed Element Number' parameter shall have a value of zero. If such an object does not exist but it can be created, the new object shall be created. The Object_Identifier property of the new object shall have the value specified in the 'Object Specifier' parameter, and the Object_Type property shall have a value consistent with the object type field of the Object_Identifier. See 20.2.14.

If the optional 'List of Initial Values' parameters is included, then all properties in the list shall be initialized as indicated. The initial values of all other properties are a local matter. If this initialization cannot be done, then a 'Result(-)' primitive shall be returned. The 'First Failed Element Number' parameter shall indicate the first property in the 'List of Initial Values' that cannot be initialized, and the object shall not be created. If the attempt to create the object is successful, a 'Result(+)' response primitive shall be issued that conveys the value of the Object_Identifier property of the newly created object.

15.4 DeleteObject Service

The DeleteObject service is used by a client BACnet-user to delete an existing object. Although this service is general in the sense that it can be applied to any object type, it is expected that most objects in a control system cannot be deleted by this service because they are protected as a security feature. There are some objects, however, that may be created and deleted dynamically. Group objects and Event Enrollment objects are examples. This service is primarily used to delete objects of these types but may also be used to remove vendor-specific deletable objects.

15.4.1 Structure

The structure of the DeleteObject service primitives is shown in Table 15-4. The terminology and symbology used in this table are explained in 5.6.

Table 15-4. Structure of DeleteObject Service Primitives

Parameter Name	Req	Ind	Rsp	Cnf
Argument	M	M(=)		
Object Identifier	M	M(=)		
Result(+)			S	S(=)
Result(-)			S	S(=)
Error Type			M	M(=)

15.4.1.1 Argument

This parameter shall convey the parameters for the DeleteObject confirmed service request.

15.4.1.1.1 Object Identifier

This parameter, of type BACnetObjectIdentifier, shall specify the object that is to be deleted by this service.

15.4.1.2 Result(+)

The 'Result(+)' parameter shall indicate that the service request succeeded and the specified object was deleted.

15.4.1.3 Result(-)

The 'Result(-)' parameter shall indicate that the service request failed and the specified object was not deleted. The reason for failure is specified in the 'Error type' parameter.

15.4.1.3.1 Error Type

This parameter consists of two component parameters: (1) an 'Error Class' and (2) an 'Error Code'. See Clause 18. The 'Error Class' and 'Error Code' to be returned for specific situations are as follows:

Situation	Error Class	Error Code
The object to be deleted does not exist.	OBJECT	UNKNOWN_OBJECT
The object exists but cannot be deleted.	OBJECT	OBJECT_DELETION_NOT_PERMITTED

15.4.2 Service Procedure

After verifying the validity of the request, the responding BACnet-user shall attempt to delete the object specified by the 'Object Identifier' parameter of the request/indication primitive. If the specified object exists and can be deleted, it shall be deleted and the 'Result(+)' primitive shall be issued. If the specified object does not exist or cannot be deleted, then the 'Result(-)' primitive shall be issued.

15.5 ReadProperty Service

The ReadProperty service is used by a client BACnet-user to request the value of one property of one BACnet Object. This service allows read access to any property of any object, whether a BACnet-defined object or not.

15.5.1 Structure

The structure of the ReadProperty service primitives is shown in Table 15-5. The terminology and symbology used in this table are explained in 5.6.

Table 15-5. Structure of ReadProperty Service Primitives

Parameter Name	Req	Ind	Rsp	Cnf
Argument	M	M(=)		
Object Identifier	M	M(=)		
Property Identifier	M	M(=)		
Property Array Index	U	U(=)		
Result (+)			S	S(=)
Object Identifier			M	M(=)
Property Identifier			M	M(=)
Property Array Index			U	U(=)
Property Value			M	M(=)
Result (-)			S	S(=)
Error Type			M	M(=)

15.5.1.1 Argument

This parameter shall convey the parameters for the ReadProperty confirmed service request.

15.5.1.1.1 Object Identifier

This parameter, of type BACnetObjectIdentifier, shall provide the means of identifying the object whose property is to be read and returned to the client BACnet-user.

15.5.1.1.2 Property Identifier

This parameter, of type BACnetPropertyIdentifier, shall provide the means of uniquely identifying the property to be read and returned by this service. Because this service is intended to read a single property of a single object, the value of this parameter shall not be one of the special property identifiers ALL, REQUIRED, or OPTIONAL.

15.5.1.1.3 Property Array Index

If the property identified above is of datatype array, this optional parameter of type Unsigned shall indicate the array index of the element of the property referenced by this service. If the 'Property Array Index' is omitted, this shall mean that the entire array shall be referenced.

If the property identified above is not of datatype array, this parameter shall be omitted.

15.5.1.2 Result(+)

The 'Result(+)' parameter shall indicate that the service request succeeded. A successful result includes the following parameters:

15.5.1.2.1 Object Identifier

This parameter, of type BACnetObjectIdentifier, shall identify the object whose property has been read and is being returned to the client BACnet-user.

15.5.1.2.2 Property Identifier

This parameter, of type BACnetPropertyIdentifier, shall identify the property that was read and is being returned by this service. Because this service is intended to read a single property of a single object, the value of this parameter shall not be one of the special property identifiers ALL, REQUIRED, or OPTIONAL.

15.5.1.2.3 Property Array Index

If the property identified above is of datatype array and a 'Property Array Index' was specified in the request, this parameter of type Unsigned shall be present and shall indicate the array index of the element of the property referenced by this service. Otherwise it shall be omitted.

15.5.1.2.4 Property Value

If access to the specified property of the specified object was successful, this parameter shall be returned. It shall be of the datatype appropriate to the specified property and shall contain the value of the requested property.

15.5.1.3 Result(-)

The 'Result(-)' parameter shall indicate that the service request has failed in its entirety. The reason for the failure shall be specified by the 'Error Type' parameter.

15.5.1.3.1 Error Type

This parameter consists of two component parameters: (1) an 'Error Class' and (2) an 'Error Code'. See Clause 18. The 'Error Class' and 'Error Code' to be returned for specific situations are as follows:

Situation	Error Class	Error Code
Specified object does not exist.	OBJECT	UNKNOWN_OBJECT
Specified property does not exist.	PROPERTY	UNKNOWN_PROPERTY
An array index is provided but the property is not an array.	PROPERTY	PROPERTY_IS_NOT_AN_ARRAY
An array index is provided that is outside the range existing in the property.	PROPERTY	INVALID_ARRAY_INDEX

15.5.2 Service Procedure

After verifying the validity of the request, the responding BACnet-user shall attempt to access the specified property of the specified object. If the access is successful, a 'Result(+)' primitive, which returns the accessed value, shall be generated. If the access fails, a 'Result(-)' primitive shall be generated, indicating the reason for the failure.

When the object-type in the Object Identifier parameter contains the value 'Device Object' and the instance in the 'Object Identifier' parameter contains the value 4194303, the responding BACnet-user shall treat the Object Identifier as if it correctly matched the local Device object. This allows the device instance of a device that does not generate I-Am messages to be determined.

15.6 ReadPropertyConditional Service

The ReadPropertyConditional service is used by a client BACnet-user to request the object identifiers and values of zero or more specified properties of all BACnet Objects that meet a list of selection criteria or "conditions." The conditions are Boolean expressions of the form (Property.Relational_Operator.Constant). In addition, the list of conditions may be logically OR'ed or AND'ed to determine whether a particular object is to be included in the set of objects whose properties are to be read. A final possibility is that ALL objects are selected and their specified properties read.

The difference between the ReadProperty service and this service is that in the ReadProperty service the objects whose properties are to be read are identified explicitly by the client BACnet-user; in the ReadPropertyConditional service, the objects are identified by the responding BACnet-user by means of the supplied selection criteria.

15.6.1 Structure

The structure of the ReadPropertyConditional service primitives is shown in Table 15-6. The terminology and symbology used in this table are explained in 5.6.

Table 15-6. Structure of ReadPropertyConditional Service Primitives

Parameter Name	Req	Ind	Rsp	Cnf
Argument	M	M(=)		
Object Selection Criteria	M	M(=)		
List of Property References	U	U(=)		
Result (+)			S	S(=)
List of Read Access Results			M	M(=)
Result (-)			S	S(=)
Error Type			M	M(=)

15.6.1.1 Argument

This parameter shall convey the parameters for the ReadPropertyConditional confirmed service request.

15.6.1.1.1 Object Selection Criteria

This parameter shall consist of a 'Selection Logic' parameter and conditionally, a list of one or more 'Selection Criteria'. Each explicit criterion consists of up to four parameters: (1) a 'Property Identifier', (2) an optional 'Property Array Index', (3) a 'Relation Specifier', and (4) a 'Comparison Value' appropriate to the identified property. See 15.6.3.1.

15.6.1.1.2 List of Property References

This optional parameter, if present, shall be a list of one or more BACnetPropertyReferences, each of which corresponds directly to a specific property of any object selected on the basis of the 'Object Selection Criteria'. Specifying the property ALL indicates that all properties of any selected object shall be returned, including any proprietary properties. Specifying the property REQUIRED means that only those properties having a conformance code of "R" or "W" shall be returned. Specifying the property identifier OPTIONAL means that only those properties that have a conformance code of "O" shall be returned. See the specification for the particular object type in Clause 12. If this parameter is absent, no properties shall be returned in the 'List of Read Access Results' portion of the 'Result(+)' primitive; however, the object identifier shall be returned in the 'Object Identifier' parameter.

15.6.1.2 Result(+)

The 'Result(+)' parameter shall indicate that the service request succeeded. A successful result includes the following parameter.

15.6.1.2.1 List of Read Access Results

The 'List of Read Access Results' parameter shall indicate, for each specified property of all objects selected on the basis of the 'Object Selection Criteria', the success or failure of the attempt to read that property. Each 'Read Access Result' shall provide the 'Object Identifier', 'Property Identifier', and, conditionally, the 'Property Array Index' for each read access attempted. If the read access was successful, the next parameter shall be the value of the specified property. If the property access failed, the next parameter shall be an error code indicating the reason for the access failure. If no objects are selected

on the basis of the 'Object Selection Criteria', then the 'List of Read Access Results' shall be of length zero. If the 'List of Property References' was not present in the request, the 'List of Read Access Results' shall consist of a list of 'Read Access Results' with empty property lists, thus conveying the object identifiers for objects that matched the Selection Criteria.

15.6.1.3 Result(-)

The 'Result(-)' parameter shall indicate that the service request has failed in its entirety. The reason for the failure shall be specified by the 'Error Type' parameter.

15.6.1.3.1 Error Type

This parameter consists of two component parameters: (1) an 'Error Class' and (2) an 'Error Code'. See Clause 18.

15.6.2 Service Procedure

After verifying the validity of the request, the responding BACnet-user shall search its object database for objects meeting the specified selection criteria. For each object found, a 'Read Access Result' shall be constructed based upon the specified 'List of Property References'. While there is no requirement that the request be carried out "atomically," nonetheless the responding BACnet-user shall ensure that all readings are taken in the shortest possible time subject only to higher priority processing. The request shall continue to be executed until an attempt has been made to access all specified properties of all selected objects. If no objects are found that meet the specified criteria, then a 'Result(+)' primitive shall be returned with a 'List of Results' of zero length. If one or more objects are found that meet the selection criteria, then a 'Result(+)' primitive shall be returned that conveys the 'Read Access Results' for all matches that are found.

15.6.3 Parameters Referenced by the ReadPropertyConditional Service

The following parameters appear in the ReadPropertyConditional service primitives:

15.6.3.1 Object Selection Criteria Parameter

The Object Selection Criteria' parameter is shown in Table 15-7. The terminology and symbology used in this table are explained in 5.6.

Table 15-7. Structure of 'Object Selection Criteria' Parameter

Parameter Name	Req Ind	Rsp Cnf	Datatype
Selection Logic	M	M(=)	Enumerated
List of Selection Criteria	C	C(=)	
Property Identifier	M	M(=)	BACnetPropertyIdentifier
Property Array Index	C	C(=)	Unsigned
Relation Specifier	M	M(=)	Enumerated
Comparison Value	M	M(=)	ANY

15.6.3.1.1 Selection Logic

This parameter shall indicate the scope of the objects whose properties are to be returned. Three values are permitted: AND, OR, and ALL. In cases where this parameter is either AND or OR, this parameter shall indicate the method by which multiple selection criteria are to be combined to determine if a particular object is to be included in the set of objects whose properties are to be returned. If AND is specified, all the Selection Criteria must evaluate to TRUE for the object to be included. If OR is specified, at least one of the Selection Criteria must evaluate to TRUE for the object to be included. If ALL is specified, all objects contained in the responding BACnet-user's object database are selected and the 'List of Selection Criteria' shall be omitted from the request primitive.

15.6.3.1.2 List of Selection Criteria

This parameter, if present, shall consist of a list of one or more property relationships that shall be used to select specific objects of the type specified above. Each relationship consists of a 'Property Identifier', a conditional 'Property Array Index', a 'Relation Specifier', and 'Comparison Value'. Taken together, each group of parameters forms a Boolean expression that when evaluated by the responding BACnet-user, is either TRUE or FALSE. If more than one property relationship is specified, the results of all the evaluations shall be combined based upon the 'Selection Logic' parameter to yield an overall TRUE or FALSE value. A value of TRUE causes the responding BACnet-user to return the values of the properties specified by the 'List of Property References' for this object.

15.6.3.1.2.1 Property Identifier

This parameter, of type BACnetPropertyIdentifier, shall identify the property whose value is to be used in the object selection process. The value of this parameter may not be one of the special property identifiers ALL, REQUIRED, or OPTIONAL.

15.6.3.1.2.2 Property Array Index

If the property identified above is of datatype array, this conditional parameter of type Unsigned shall be present and shall indicate the array index of the element of the property referenced by this service. Otherwise, it shall be omitted. A 'Property Array Index' value of zero is not permitted.

15.6.3.1.2.3 Relation Specifier

This parameter shall represent one of the six Boolean operators:

=	(equality)	>	(greater than)
≠	(inequality)	≤	(less than or equal to)
<	(less than)	≥	(greater than or equal to)

These Boolean operators may not be used for every property of every object. Some property values may only allow equals (=) and not equals (≠) comparisons. Table 15-8 shows which operators may be used with each datatype.

Table 15-8. Valid Boolean Operators for BACnet Datatypes

Datatype	Operators Allowed	Datatype	Operators Allowed
NULL	= ≠	CharacterString	= ≠ < > ≤ ≥
BOOLEAN	= ≠	OCTET STRING	= ≠
Unsigned	= ≠ < > ≤ ≥	ENUMERATED	= ≠
INTEGER	= ≠ < > ≤ ≥	Date	= ≠ < > ≤ ≥
REAL	= ≠ < > ≤ ≥	Time	= ≠ < > ≤ ≥
BIT STRING	= ≠	Others	= ≠
BACnetObjectIdentifier	= ≠ < > ≤ ≥		

15.6.3.1.2.4 Comparison Value

This parameter shall be a constant of the same datatype as the property being compared. If the Comparison Value is being used to select properties of type CharacterString, then the "wildcard" characters "?" and "*" may be used but only for the = or ≠ operators.

The "?" character is used to match exactly one character at the position at which it occurs in the string. A Comparison Value may contain multiple occurrences of "?". Thus "??1" matches "AC1", "CP1", "SF1". It does not match "C1", "P1", or "PUMP1".

The "*" is used to match zero or more characters starting at its position in the string and extending to the right. Only one "*" wildcard may be used per comparison value. Thus "C*1" matches "C1", "CP1", and "Chiller1".

When comparison value is being used to select object identifiers, the object identifier shall be considered as a 32-bit unsigned integer with the most significant octet being the first octet of the object identifier.

When the comparison value is being used to select character strings the comparisons are case sensitive. When character strings are compared for inequalities (<, >, ≤, ≥), they are compared based on the numerical encoding of each character, left to right, "phone book style." For example:

"ABC" < "ABCD" < "DE" < "DEF"

When properties with a Date or Time datatype are compared for inequalities, the comparison shall be chronological. For example:

1-Jan-1991 < 2-Feb-1992
7:00:00.00 < 23:00:00.00

Subcomponents of a Date or Time, such as the year, may be "wildcarded" if the value of the subcomponent is the special value "unspecified." Subcomponents of a Date or Time that have the value "unspecified" shall be skipped in the comparison and that subcomponent shall be considered as "equal." It is possible that either the comparison value, or the property value with which it is being compared, or both, may have subcomponent(s) whose value(s) are "unspecified." In either case, the rules for matching unspecified values shall apply. For example (using * to mean unspecified):

Comparison Value		Property Value	Because
- Sun	=	29-Jan-1995 Sun	Sun = Sun
1-Jan-*	<	2-Feb-1995	1-Jan < 2-Feb
-Aug-	>	10-Mar-1995	Aug > Mar
*:59:00.00	>	6:23:99.0	59:00.00 > 23:00.00
6:*.*.*	=	6:25:00.00	6 = 6

Properties with datatype NULL, BOOLEAN, BIT STRING, OCTET STRING, ENUMERATED, or any complex structure of primitive types may only be compared as equal or not equal (=, ≠).

15.6.3.2 Read Access Result

The 'Read Access Result' parameter is shown in Table 15-9. The terminology and symbology used in this table are explained in 5.6.

Table 15-9. Structure of 'Read Access Result' Parameter

Parameter Name	Rsp	Cnf	Datatype
Object Identifier	M	M(=)	BACnetObjectIdentifier
List of Results	U	U(=)	
Property Identifier	M	M(=)	BACnetPropertyIdentifier
Property Array Index	U	U(=)	Unsigned
Property Value	S	S(=)	ANY
Property Access Error	S	S(=)	Error

15.6.3.2.1 Object Identifier

This parameter, of type BACnetObjectIdentifier, shall identify the object whose properties are being returned to the service requester.

15.6.3.2.2 List of Results

The result of reading a given property is either the value of the property or an error code indicating why the access attempt failed. Note that if the 'List of Property References' parameter was not specified in the original request, the 'List of Results' parameter is to be omitted. This provides a means of obtaining a list of Object Identifiers of all objects meeting the specified 'Object Selection Criteria' without receiving the values of any particular properties of those objects.

15.6.3.2.2.1 Property Identifier

This parameter, of type BACnetPropertyIdentifier, shall identify the property whose value is to being returned.

15.6.3.2.2.2 Property Array Index

If the property identified above is of datatype array and a 'Property Array Index' was specified in the request, this parameter of type Unsigned shall be present and shall indicate the array index of the element of the property referenced by this service. Otherwise it shall be omitted.

15.6.3.2.2.3 Property Value

If access to the specified property of the selected object is successful, this parameter shall be returned. It shall be of the datatype appropriate to the specified property and shall contain the value of the requested property.

15.6.3.2.2.4 Property Access Error

If the responding BACnet-user is unable to access the specified property of the specified object, then this parameter shall be returned. It shall contain a value that indicates the reason for the access failure. This parameter consists of two component parameters: (1) an 'Error Class' and (2) an 'Error Code'. See Clause 18. Note that this parameter refers only to a failure of the

access to a specific property of a specific object, whereas the 'Error Type' parameter returned in the 'Result(-)' primitive refers to a failure of the entire ReadPropertyConditional service request.

15.7 ReadPropertyMultiple Service

The ReadPropertyMultiple service is used by a client BACnet-user to request the values of one or more specified properties of one or more BACnet Objects. This service allows read access to any property of any object, whether a BACnet-defined object or not. The user may read a single property of a single object, a list of properties of a single object, or any number of properties of any number of objects. A 'Read Access Specification' with the property identifier ALL can be used to learn the implemented properties of an object along with their values.

15.7.1 Structure

The structure of the ReadPropertyMultiple service primitives is shown in Table 15-10. The terminology and symbology used in this table are explained in 5.6.

Table 15-10. Structure of ReadPropertyMultiple Service Primitives

Parameter Name	Req	Ind	Rsp	Cnf
Argument	M	M(=)		
List of Read Access Specifications	M	M(=)		
Result (+)			S	S(=)
List of Read Access Results			M	M(=)
Result (-)			S	S(=)
Error Type			M	M(=)

15.7.1.1 Argument

This parameter shall convey the parameters for the ReadPropertyMultiple confirmed service request.

15.7.1.1.1 List of Read Access Specifications

This parameter shall consist of a list of one or more 'Read Access Specifications'. Each specification shall consist of two parameters: (1) an 'Object Identifier' and (2) a 'List of Property References'. See 15.7.3.1.

15.7.1.2 Result(+)

The 'Result(+)' parameter shall indicate that the service request succeeded. A successful result includes the following parameter.

15.7.1.2.1 List of Read Access Results

The 'List of Read Access Results' parameter shall indicate the success or failure of the access to each specified property. The contents of each Read Access Result are described in 15.7.3.2.

15.7.1.3 Result(-)

The 'Result(-)' parameter shall indicate that the service request has failed in its entirety. The reason for the failure shall be specified by the 'Error Type' parameter.

15.7.1.3.1 Error Type

This parameter consists of two component parameters: (1) an 'Error Class' and (2) an 'Error Code'. See Clause 18. The 'Error Class' and 'Error Code' to be returned for specific situations are as follows:

Situation	Error Class	Error Code
Specified object does not exist.	OBJECT	UNKNOWN_OBJECT
Specified property does not exist.	PROPERTY	UNKNOWN_PROPERTY
An array index is provided but the property is not an array.	PROPERTY	PROPERTY_IS_NOT_AN_ARRAY
An array index is provided that is outside the range existing in the property.	PROPERTY	INVALID_ARRAY_INDEX

15.7.2 Service Procedure

After verifying the validity of the request, the responding BACnet-user shall attempt to access the specified properties of the specified objects and shall construct a 'List of Read Access Results' in the order specified in the request. If the 'List of Property References' portion of the 'List of Read Access Specifications' parameter contains the property identifier ALL, REQUIRED, or OPTIONAL, then the 'List of Read Access Results' shall be constructed as if each property being returned had been explicitly referenced (see 15.7.3.1.2). While there is no requirement that the request be carried out "atomically," nonetheless the responding BACnet-user shall ensure that all readings are taken in the shortest possible time subject only to higher priority processing. The request shall continue to be executed until an attempt has been made to access all specified properties. If none of the specified objects is found or if none of the specified properties of the specified objects can be accessed, either a 'Result(-)' primitive or a Result(+) primitive that returns error codes for all properties shall be issued. If any of the specified properties of the specified objects can be accessed, then a 'Result(+)' primitive shall be issued, which returns all accessed values and error codes for all properties that could not be accessed.

When the object-type in the Object Identifier portion of the Read Access Specification parameter contains the value 'Device Object' and the instance of that 'Object Identifier' parameter contains the value 4194303, the responding BACnet-user shall treat the Object Identifier as if it correctly matched the local Device object. This allows the device instance of a device that does not generate I-Am messages to be determined.

15.7.3 Parameters Referenced by the ReadPropertyMultiple Service

The following parameters appear in the ReadPropertyMultiple service primitives.

15.7.3.1 Read Access Specification Parameter

The 'Read Access Specification' parameter is shown in Table 15-11. The terminology and symbology used in this table are explained in 5.6.

Table 15-11. Structure of 'Read Access Specification' Parameter

Parameter Name	Req Ind	Rsp Cnf	Datatype
Object Identifier	M	M(=)	BACnetObjectIdentifier
List of Property References	M	M(=)	List of BACnetPropertyReference

15.7.3.1.1 Object Identifier

This parameter, of type BACnetObjectIdentifier, shall provide the means of identifying the object whose properties are to be read and returned to the service requester.

15.7.3.1.2 List of Property References

This parameter shall be a list of one or more BACnetPropertyReferences, each of which corresponds directly to a specific property of the object identified above. The property identifier ALL means that all defined properties of the object are to be accessed, including any proprietary properties. The property identifier REQUIRED means that only those properties having a conformance code of "R" or "W" shall be returned. The property identifier OPTIONAL means that only those properties that have a conformance code "O" shall be returned. See the specification for the particular object type in Clause 12.

15.7.3.2 Read Access Result

The 'Read Access Result' parameter is shown in Table 15-12. The terminology and symbology used in this table are explained in 5.6.

Table 15-12. Structure of 'Read Access Result' Parameter

Parameter Name	Rsp	Cnf	Datatype
Object Identifier	M	M(=)	BACnetObjectIdentifier
List of Results	M	M(=)	
Property Identifier	M	M(=)	BACnetPropertyIdentifier
Property Array Index	U	U(=)	Unsigned
Property Value	S	S(=)	ANY
Property Access Error	S	S(=)	Error

15.7.3.2.1 Object Identifier

This parameter, of type BACnetObjectIdentifier, shall identify the object whose properties are being returned to the service requester.

15.7.3.2.2 List of Results

The result of reading a given property is either the present value of the property or an error code indicating why the access attempt failed. Each element in the 'List of Results' contains a 'Property Identifier' and conditionally a 'Property Array Index', followed by either a 'Property Value' or a 'Property Access Error'.

15.7.3.2.2.1 Property Identifier

This parameter, of type BACnetPropertyIdentifier, shall identify the property whose value has been read.

15.7.3.2.2.2 Property Array Index

If the property identified above is of datatype array and a 'Property Array Index' was specified in the request, this parameter of type Unsigned shall be present and shall indicate the array index of the element of the property referenced by this service. Otherwise it shall be omitted.

15.7.3.2.2.3 Property Value

If access to the specified property of the specified object is successful, this parameter shall be returned. It shall be of a datatype consistent with the requested property and shall contain the value of the requested property.

15.7.3.2.2.4 Property Access Error

If the responding BACnet-user is unable to access the specified property of the specified object, then this parameter shall be returned. It shall contain a value that indicates the reason for the access failure. This parameter consists of two component parameters: (1) an 'Error Class' and (2) an 'Error Code'. See Clause 18. Note that this parameter refers only to a failure of the access to a specific property of a specific object, whereas the 'Error Type' parameter returned in the 'Result(-)' primitive refers to a failure of the entire ReadPropertyMultiple service request.

15.8 ReadRange Service

The ReadRange service is used by a client BACnet-user to read a specific range of data items representing a subset of data available within a specified object property. The service may be used with any list or array of lists property.

15.8.1 Structure

The structure of the ReadRange primitive is shown in Table 15-13. The terminology and symbology used in this table are explained in 5.6.

Table 15-13. Structure of ReadRange Service Primitives

Parameter Name	Req	Ind	Rsp	Cnf
Argument	M	M(=)		
Object Identifier	M	M(=)		
Property Identifier	M	M(=)		
Property Array Index	C	C(=)		
Range	U	U(=)		
Result(+)			S	S(=)
Object Identifier			M	M(=)
Property Identifier			M	M(=)
Property Array Index			C	C(=)
Result Flags			M	M(=)
Item Count			M	M(=)
Item Data			M	M(=)
First Sequence Number			C	C(=)
Result(-)			S	S(=)
Error Type			M	M(=)

15.8.1.1 Argument

This parameter shall convey the parameters for the ReadRange confirmed service request.

15.8.1.1.1 Object Identifier

This parameter, of type BACnetObjectIdentifier, specifies the object and property is to be read.

15.8.1.1.2 Property Identifier

This parameter, of type BACnetPropertyIdentifier, specifies the property to be read by this service. Because this service is intended to read a single property of a single object, the value of this parameter shall not be one of the special property identifiers ALL, REQUIRED, or OPTIONAL.

15.8.1.1.3 Property Array Index

If the property identified above is of datatype array of lists, this optional parameter of type Unsigned shall indicate the array index of the element of the property referenced by this service. If the property identified above is not of datatype array of lists, this parameter shall be omitted. The index value shall not be zero.

15.8.1.1.4 Range

This optional parameter shall convey criteria for the consecutive range items within the referenced property that are to be returned, as described in 15.8.2. The 'Range' parameter is shown in Table 15-14. The terminology and symbology used in this table are explained in 5.6.

Table 15-14. Structure of the 'Range' Parameter

Parameter Name	Req	Ind	Datatype
By Position	S	S(=)	
Reference Index	M	M(=)	Unsigned
Count	M	M(=)	INTEGER
By Sequence Number	S	S(=)	
Reference Sequence Number	M	M(=)	Unsigned32
Count	M	M(=)	INTEGER
By Time	S	S(=)	
Reference Time	M	M(=)	BACnetDateTime
Count	M	M(=)	INTEGER

15.8.1.1.4.1 By Position

The 'By Position' parameter shall indicate that the particular items to be read are referenced by an index.

15.8.1.1.4.1.1 Reference Index

The 'Reference Index' parameter specifies the index of the first (if 'Count' is positive) or last (if 'Count' is negative) item to be read.

15.8.1.1.4.1.2 Count

The absolute value of the 'Count' parameter specifies the number of records to be read. If 'Count' is positive, the record specified by 'Reference Index' shall be the first and oldest record read and returned; if 'Count' is negative the record specified by 'Reference Index' shall be the last and newest record. 'Count' may not be zero.

15.8.1.1.4.2 By Sequence Number

The 'By Sequence Number' parameter shall indicate that the particular items to be read are referenced by a sequence number and that the response shall include the sequence number of the first returned item. This differs semantically from the 'By Position' parameter choice. The Reference Number provided in the 'By Position' choice references an item by its position in the list. In contrast, the Reference Number provided in the 'By Sequence Number' choice references an item by its sequence number, which it is given when the item is added to the list. Not all lists implement the concept of a sequence number. An example of a list that does implement the concept of a sequence number is the Log_Buffer property of the Trend Log object.

15.8.1.1.4.2.1 Reference Sequence Number

The 'Reference Sequence Number' parameter specifies the sequence number of the first (if 'Count' is positive) or last (if 'Count' is negative) item to be read.

15.8.1.1.4.2.2 Count

The absolute value of the 'Count' parameter specifies the number of records to be read. If 'Count' is positive, the record specified by 'Reference Sequence Number' shall be the first and oldest record read and returned. If 'Count' is negative the record specified by 'Reference Sequence Number' shall be the last and newest record read and returned. 'Count' shall not be zero.

15.8.1.1.4.3 By Time

The 'By Time' parameter shall indicate that the particular item to be read is referenced by timestamp and that the Sequence Number of the item shall be returned in the response. This form of the service is expected to be used when searching lists that are loosely indexed by time.

15.8.1.1.4.3.1 Reference Time

If 'Count' is positive, the first record to be read shall be the first record with a timestamp newer than the time specified by the 'Reference Time' parameter. If 'Count' is negative, the last record to be read shall be the newest record with a timestamp older than the time specified by the 'Reference Time' parameter.

15.8.1.1.4.3.2 Count

The absolute value of the 'Count' parameter specifies the number of records to be read. If 'Count' is positive, the first record with a timestamp newer than the time specified by 'Reference Time' shall be the first and oldest record read and returned; if

'Count' is negative, the newest record with a timestamp older than the time specified by 'Reference Time' shall be the last and newest record. 'Count' shall not be zero.

15.8.1.2 Result(+)

The 'Result(+)' parameter shall indicate that the service request succeeded. A successful result includes the following parameters.

15.8.1.2.1 Object Identifier

This parameter, of type BACnetObjectIdentifier, specifies the object that was read.

15.8.1.2.2 Property Identifier

This parameter, of type BACnetPropertyIdentifier, shall identify that property that was read.

15.8.1.2.3 Property Array Index

If the property identified above is of datatype array of lists, this parameter of type Unsigned shall indicate the array index of the element of the property referenced by this service. If the property identified above is not of datatype array of lists, this parameter shall be omitted.

15.8.1.2.4 Result Flags

This parameter, of type BACnetResultFlags, shall convey several flags that describe characteristics of the response data:

{FIRST_ITEM, LAST_ITEM, MORE_ITEMS}

The FIRST_ITEM flag indicates whether this response includes the first list or array element (in the case of positional indexing), or the oldest timestamped item (in the case of time indexing).

The LAST_ITEM flag indicates whether this response includes the last list or array element (in the case of positional indexing), or the newest timestamped item (in the case of time indexing)

The MORE_ITEMS flag indicates whether more items matched the request but were not transmittable within the PDU.

15.8.1.2.5 Item Count

This parameter, of type Unsigned, represents the number of items that were returned.

15.8.1.2.6 Item Data

This parameter consists of a list of the requested data.

15.8.1.2.7 First Sequence Number

This parameter, of type Unsigned32, specifies the sequence number of the first item returned. This parameter is only included if the 'Range' parameter of the request was of the type 'By Sequence Number' or 'By Time' and 'Item Count' is greater than 0.

15.8.1.3 Result(-)

The 'Result(-)' parameter shall indicate that the service request has failed. The reason for the failure shall be specified by the 'Error Type' parameter.

15.8.1.3.1 Error Type

This parameter consists of two component parameters: (1) the 'Error Class' and (2) the 'Error Code'. See Clause 18.

15.8.2 Service Procedure

The responding BACnet-user shall first verify the validity of the 'Object Identifier', 'Property Identifier' and "Property Array Index' parameters and return a 'Result(-)' response with the appropriate error class and code if the object or property is unknown, if the referenced data is not a list or array, or if it is currently inaccessible for another reason.

If the 'Range' parameter is not present, then the responding BACnet-user shall read and attempt to return all of the available items in the list or array.

If the 'Range' parameter is present and specifies the 'By Position' parameters, then the responding BACnet-user shall read and attempt to return all of the items specified. The items specified include the item at the index specified by 'Reference Index' plus up to 'Count' - 1 items following if 'Count' is positive, or up to -1 - 'Count' items preceding if 'Count' is negative. The first element of a list shall be associated with index 1.

If the 'Range' parameter is present and specifies the 'By Time' parameter, then the responding BACnet-user shall read and attempt to return all of the items specified. If 'By Time' parameters are specified and the property values are not timestamped an error shall be returned. If 'Count' is positive, the records specified include the first record with a timestamp newer than 'Reference Time' plus up to 'Count'-1 items following. If 'Count' is negative, the records specified include the newest record with a timestamp older than 'Reference Time' and up to -1-'Count' records preceding. The sequence number of the first item returned shall be included in the response. The items shall be returned in chronological order.

If the 'Range' parameter is present and specifies the 'By Sequence Number' parameters, then the responding BACnet-user shall read and attempt to return all of the items specified. The items specified are all items with a sequence number in the range 'Reference Sequence Number' to 'Reference Sequence Number' plus 'Count'-1 if 'Count' is positive, or in the range 'Reference Sequence Number' plus 'Count'+1 to 'Reference Sequence' if 'Count' is negative.

To avoid missing items when using chained time-based reads, the first item in the desired set should be found using the 'By Time' form of the 'Range' parameter. Subsequent requests to retrieve the remaining items in the desired set should use the 'By Sequence Number' form of the 'Range' parameter. The reason for this is that lists that include a timestamp but are ordered by time of arrival may have entries with out-of-order timestamps due to negative time changes in the local device's clock. If items are read that match the request parameters but cannot be returned in the response, the 'Result Flags' parameter shall contain the MORE_ITEMS flag set to TRUE, otherwise it shall be FALSE. Remaining items may be obtained with subsequent requests specifying appropriately chosen parameters.

The returned response shall convey the number of items read and returned using the 'Item Count' parameter. The actual items shall be returned in the 'Item Data' parameter. If the returned response includes the first positional index and a 'By Position' request had been made, or the oldest sequence number and a 'By Sequence Number' or 'By Time' request had been made, then the 'Result Flags' parameter shall contain the FIRST_ITEM flag set to TRUE; otherwise it shall be FALSE.

If the returned response includes the last positional index and a 'By Position' request had been made, or the newest sequence number and a 'By Sequence Number' or 'By Time' request had been made, then the 'Result Flags' shall contain the LAST_ITEM flag set to TRUE; otherwise it shall be FALSE.

If there are no items in the list that match the 'Range' parameter criteria, then a Result(+) shall be returned with an 'Item Count' of 0 and no 'First Sequence Number' parameter.

15.9 WriteProperty Service

The WriteProperty service is used by a client BACnet-user to modify the value of a single specified property of a BACnet object. This service potentially allows write access to any property of any object, whether a BACnet-defined object or not. Some implementors may wish to restrict write access to certain properties of certain objects. In such cases, an attempt to modify a restricted property shall result in the return of an error of 'Error Class' PROPERTY and 'Error Code' WRITE_ACCESS_DENIED. Note that these restricted properties may be accessible through the use of Virtual Terminal services or other means at the discretion of the implementor.

15.9.1 Structure

The structure of the WriteProperty service primitives is shown in Table 15-15. The terminology and symbology used in this table are explained in 5.6.

Table 15-15. Structure of WriteProperty Service Primitives

Parameter Name	Req	Ind	Rsp	Cnf
Argument	M	M(=)		
Object Identifier	M	M(=)		
Property Identifier	M	M(=)		
Property Array Index	U	U(=)		
Property Value	M	M(=)		
Priority	C	C(=)		
Result (+)			S	S(=)
Result (-)			S	S(=)
Error Type			M	M(=)

15.9.1.1 Argument

This parameter shall convey the parameters for the WriteProperty confirmed service request.

15.9.1.1.1 Object Identifier

This parameter, of type BACnetObjectIdentifier, shall provide the means of identifying the object whose property is to be modified.

15.9.1.1.2 Property Identifier

This parameter, of type BACnetPropertyIdentifier, shall provide the means of uniquely identifying the property to be written by this service. Because this service is intended to write a single property of a single object, the value of this parameter shall not be one of the special property identifiers ALL, REQUIRED, or, OPTIONAL.

15.9.1.1.3 Property Array Index

If the property identified above is of datatype array, this optional parameter of type Unsigned shall indicate the array index of the element of the property referenced by this service. If the 'Property Array Index' is omitted for an array, this shall mean that the entire array shall be referenced.

If the property identified above is not of datatype array, this parameter shall be omitted.

15.9.1.1.4 Property Value

If access to the specified property of the specified object is successful, this parameter shall be used to replace the value of the property at the time of access. It shall be of any datatype that is valid for the property being modified.

15.9.1.1.5 Priority

This parameter shall be an integer in the range 1-16, which indicates the priority assigned to this write operation. If an attempt is made to write to a commandable property without specifying the priority, a default priority of 16 (the lowest priority) shall be assumed. If an attempt is made to write to a property that is not commandable with a specified priority, the priority shall be ignored. See Clause 19.

15.9.1.2 Result(+)

The 'Result(+)' parameter shall indicate that the service request succeeded in its entirety.

15.9.1.3 Result(-)

The 'Result(-)' parameter shall indicate that the service request has failed in its entirety. The reason for the failure shall be specified by the 'Error Type' parameter.

15.9.1.3.1 Error Type

This parameter consists of two component parameters: (1) an 'Error Class' and (2) an 'Error Code'. See Clause 18. The 'Error Class' and 'Error Code' to be returned for specific situations are as follows:

Situation	Error Class	Error Code
Specified object does not exist.	OBJECT	UNKNOWN_OBJECT
Specified property does not exist.	PROPERTY	UNKNOWN_PROPERTY
An array index is provided but the property is not an array.	PROPERTY	PROPERTY_IS_NOT_AN_ARRAY
An array index is provided that is outside the range existing in the property.	PROPERTY	INVALID_ARRAY_INDEX
The specified property is currently not writable by the requestor.	PROPERTY	WRITE_ACCESS_DENIED
The datatype of the value provided is incorrect for the specified property.	PROPERTY	INVALID_DATATYPE
The property is Object_Name and the name is already in use in the device.	PROPERTY	DUPLICATE_NAME
The property is Object Identifier and the identifier is already in use in the device.	PROPERTY	DUPLICATE_OBJECT_ID
The value provided is outside the range of values that the property can take on.	PROPERTY	VALUE_OUT_OF_RANGE
There is not enough space to store the new value.	RESOURCES	NO_SPACE_TO_WRITE_PROPERTY

15.9.2 Service Procedure

After verifying the validity of the request, the responding BACnet-user shall attempt to modify the specified property of the specified object using the value provided in the 'Property Value' parameter. If the modification attempt is successful, a 'Result(+)' primitive shall be issued. If the modification attempt fails, a 'Result(-)' primitive shall be issued indicating the reason for the failure. Interpretation of the conditional Priority parameter shall be as defined in Clause 19.

15.10 WritePropertyMultiple Service

The WritePropertyMultiple service is used by a client BACnet-user to modify the value of one or more specified properties of a BACnet object. This service potentially allows write access to any property of any object, whether a BACnet-defined object or not.

Properties shall be modified by the WritePropertyMultiple service in the order specified in the 'List of Write Access Specifications' parameter, and execution of the service shall continue until all of the specified properties have been written to or a property is encountered that for some reason cannot be modified as requested.

Some implementors may wish to restrict write access to certain properties of certain objects. In such cases, an attempt to modify a restricted property shall result in the return of an error of 'Error Class' PROPERTY and 'Error Code' WRITE_ACCESS_DENIED. Note that these restricted properties may be accessible through the use of Virtual Terminal services or other means at the discretion of the implementor.

15.10.1 Structure

The structure of the WritePropertyMultiple service primitives is shown in Table 15-16. The terminology and symbology used in this table are explained in 5.6.

Table 15-16. Structure of WritePropertyMultiple Service Primitives

Parameter Name	Req	Ind	Rsp	Cnf
Argument	M	M(=)		
List of Write Access Specifications	M	M(=)		
Result (+)			S	S(=)
Result (-)			S	S(=)
Error Type			M	M(=)
First Failed Write Attempt			M	M(=)

15.10.1.1 Argument

This parameter shall convey the parameters for the WritePropertyMultiple confirmed service request.

15.10.1.1.1 List of Write Access Specifications

Each 'Write Access Specification' conveys the information required to carry out the modification of a property or properties of a BACnet object. The specification consists of up to five parameters: (1) an 'Object Identifier'; a List of Properties each of which consists of (2) a 'Property Identifier'; (3) an optional 'Property Array Index'; (4) a 'Property Value'; and (5) an optional 'Priority'.

15.10.1.2 Result(+)

The 'Result(+)' parameter shall indicate that the service request succeeded in its entirety and that all specified properties were correctly modified.

15.10.1.3 Result(-)

The 'Result(-)' parameter shall indicate that at least one of the specified properties could not be modified as requested. The reason for the failure shall be conveyed by the 'Error Type' parameter along with the 'Object Identifier', 'Property Identifier', and 'Property Array Index' of the first encountered property that, for the reason specified by the 'Error Type' parameter, could not be properly written.

15.10.1.3.1 Error Type

This parameter consists of two component parameters: (1) an 'Error Class' and (2) an 'Error Code'. See Clause 18. The 'Error Class' and 'Error Code' to be returned in a 'Result(-)' for specific situations are as follows:

Situation	Error Class	Error Code
Specified object does not exist.	OBJECT	UNKNOWN_OBJECT
Specified property does not exist.	PROPERTY	UNKNOWN_PROPERTY
An array index is provided but the property is not an array.	PROPERTY	PROPERTY_IS_NOT_AN_ARRAY
An array index is provided that is outside the range existing in the property.	PROPERTY	INVALID_ARRAY_INDEX
The specified property is currently read-only.	PROPERTY	WRITE_ACCESS_DENIED
The datatype of the value provided is incorrect for the specified property.	PROPERTY	INVALID_DATATYPE
The property is Object_Name and the name is already in use in the device.	PROPERTY	DUPLICATE_NAME
The property is Object Identifier and the identifier is already in use in the device.	PROPERTY	DUPLICATE_OBJECT_ID
The value provided is outside the range of values that the property can take on.	PROPERTY	VALUE_OUT_OF_RANGE
There is not enough space to store the new value.	RESOURCES	NO_SPACE_TO_WRITE_PROPERTY

15.10.1.3.2 First Failed Write Attempt

This parameter, of type BACnetObjectPropertyReference, shall convey the 'Object Identifier', 'Property Identifier', and 'Property Array Index' of the first failed element in the 'List of Write Access Specification' for which the write attempt failed.

15.10.2 Service Procedure

For each 'Write Access Specification' contained in the 'List of Write Access Specifications', the value of each specified property shall be replaced by the property value provided in the 'Write Access Specification' and a 'Result(+)' primitive shall be issued, indicating that the service request was carried out in its entirety. Interpretation of the conditional Priority parameter shall be as specified in Clause 19.

If, in the process of carrying out the modification of the indicated properties in the order specified in the 'List of Write Access Specifications', a property is encountered that cannot be modified, the responding BACnet-user shall issue a 'Result(-)' response primitive indicating the reason for the failure. The result of this service shall be either that all of the specified properties or only the properties up to, but not including, the property specified in the 'First Failed Write Attempt' parameter were successfully modified.

15.10.3 Parameters Referenced by the WritePropertyMultiple Service

The 'Write Access Specification' parameter is shown in Table 15-17. The terminology and symbology used in this table are explained in 5.6.

Table 15-17. Structure of 'Write Access Specification' Parameter

Parameter Name	Req Ind	Rsp Cnf	Datatype
Object Identifier	M	M(=)	BACnetObjectIdentifier
List of Properties	M	M(=)	
Property Identifier	M	M(=)	BACnetPropertyIdentifier
Property Array Index	U	U(=)	Unsigned
Property Value	M	M(=)	ANY
Priority	C	C(=)	Unsigned(1..16)

15.10.3.1 Object Identifier

This parameter, of type BACnetObjectIdentifier, shall provide the means of identifying the object whose property or properties are to be modified.

15.10.3.2 List of Properties

This parameter shall specify a list of one or more properties of the object identified above, the value to be written to each property, and the priority assigned to the write operation. Each element of the list shall specify the following parameters.

15.10.3.2.1 Property Identifier

This parameter, of type BACnetPropertyIdentifier, shall provide the means of uniquely identifying the property to be written by this service. The value of this parameter shall not be one of the special property identifiers ALL, REQUIRED, or, OPTIONAL.

15.10.3.2.2 Property Array Index

If the property identified above is of datatype array, this optional parameter of type Unsigned shall indicate the array index of the element of the property referenced by this service. If the 'Property Array Index' is omitted for an array, this shall mean that the entire array shall be referenced.

If the property identified above is not of datatype array, this parameter shall be omitted.

15.10.3.2.3 Property Value

If access to the specified property of the specified object is successful, this parameter shall be used to replace the value of the property at the time of access. It shall be of any datatype that is valid for the property being modified.

15.10.3.2.4 Priority

This parameter shall be an integer in the range 1-16, which indicates the priority assigned to this write operation. If an attempt is made to write to a commandable property without specifying the priority, a default priority of 16 (the lowest priority) shall be assumed. If an attempt is made to write to a property that is not commandable with a specified priority, the priority shall be ignored. See Clause 19.

16 REMOTE DEVICE MANAGEMENT SERVICES

16.1 DeviceCommunicationControl Service

The DeviceCommunicationControl service is used by a client BACnet-user to instruct a remote device to stop initiating and optionally stop responding to all APDUs (except DeviceCommunicationControl or, if supported, ReinitializeDevice) on the communication network or internetwork for a specified duration of time. This service is primarily used by a human operator for diagnostic purposes. A password may be required from the client BACnet-user prior to executing the service. The time duration may be set to "indefinite," meaning communication must be re-enabled by a DeviceCommunicationControl or, if supported, ReinitializeDevice service, not by time.

16.1.1 Structure

The structure of the DeviceCommunicationControl service primitives is shown in Table 16-1. The terminology and symbology used in this table are explained in 5.6.

Table 16-1. Structure of DeviceCommunicationControl Service Primitives

Parameter Name	Req	Ind	Rsp	Cnf
Argument	M	M(=)		
Time Duration	U	U(=)		
Enable/Disable	M	M(=)		
Password	U	U(=)		
Result (+)			S	S(=)
Result (-)			S	S(=)
Error Type			M	M(=)

16.1.1.1 Argument

This parameter shall convey the parameters for the DeviceCommunicationControl confirmed service request.

16.1.1.1.1 Time Duration

This optional parameter, of type Unsigned16, indicates the number of minutes that the remote device shall ignore all APDUs except DeviceCommunicationControl and, if supported, ReinitializeDevice APDUs. If the 'Time Duration' parameter is not present, then the time duration shall be considered indefinite, meaning that only an explicit DeviceCommunicationControl or ReinitializeDevice APDU shall enable communications. The 'Time Duration' parameter shall be ignored and the time period considered to be indefinite if the 'Enable/Disable' parameter has a value of ENABLE.

16.1.1.1.2 Enable/Disable

This parameter is an enumeration that may take on the values ENABLE, DISABLE, or DISABLE_INITIATION. It is used to indicate whether the responding BACnet-user is to enable all, disable initiation, or disable all communications on the network interface. When this parameter has a value of ENABLE, communications shall be enabled. When this parameter has a value of DISABLE, all communications shall be disabled. When this parameter has a value of DISABLE_INITIATION, the initiation of communications shall be disabled. When network communications are completely disabled, only DeviceCommunicationControl and ReinitializeDevice APDUs shall be processed and no messages shall be initiated. When the initiation of communications is disabled, all APDUs shall be processed and responses returned as required and no messages shall be initiated with the exception of I-Am requests, which shall be initiated only in response to Who-Is messages. In this state, a device that supports I-Am request initiation shall send one I-Am request for any Who-Is request that is received if and only if the Who-Is request does not contain an address range or the device is included in the address range.

16.1.1.1.3 Password

This optional parameter shall be a CharacterString of up to 20 characters. For those devices that require password protection, the service shall be denied if the parameter is absent or if the password is incorrect. For those devices that do not require a password, this parameter shall be ignored.

16.1.1.2 Result(+)

This parameter shall indicate that the service request succeeded.

16.1.1.3 Result(-)

This parameter shall indicate that the service request has failed. The reason for failure shall be specified by the 'Error Type' parameter.

16.1.1.3.1 Error Type

This parameter consists of two components parameters: (1) the 'Error Class' and (2) the 'Error Code'. See Clause 18.

16.1.2 Service Procedure

After verifying the validity of the request, including the password, the responding BACnet-user shall respond with a 'Result(+)' service primitive and, if the 'Enable/Disable' parameter is DISABLE, discontinue responding to any subsequent messages except DeviceCommunicationControl and, if supported, ReinitializeDevice messages and discontinue initiating messages. Communication shall be disabled until either the 'Time Duration' has expired or a valid DeviceCommunicationControl (with 'Enable/Disable' = ENABLE) or, if supported, a valid ReinitializeDevice message is received. If the responding BACnet-user does not have a clock and the 'Time Duration' parameter is not set to "indefinite," the APDU shall be ignored and a 'Result(-)' service primitive shall be issued. If the password is invalid or absent when one is required, the APDU shall be ignored and a 'Result(-)' response primitive shall be issued.

16.2 ConfirmedPrivateTransfer Service

The ConfirmedPrivateTransfer is used by a client BACnet-user to invoke proprietary or non-standard services in a remote device. The specific proprietary services that may be provided by a given device are not defined by this standard. The PrivateTransfer services provide a mechanism for specifying a particular proprietary service in a standardized manner. The only required parameters for these services are a vendor identification code and a service number. Additional parameters may be supplied for each service if required. The form and content of these additional parameters, if any, are not defined by this standard. The vendor identification code and service number together serve to unambiguously identify the intended purpose of the information conveyed by the remainder of the APDU or the service to be performed by the remote device based on parameters in the remainder of the APDU.

The vendor identification code is a unique code assigned to the vendor by ASHRAE. The mechanism for administering these codes is not defined in this standard. See Clause 23.

16.2.1 ConfirmedPrivateTransfer Service Structure

The structure of the ConfirmedPrivateTransfer service primitives is shown in Table 16-2. The terminology and symbology used in this table are explained in 5.6.

Table 16-2. Structure of ConfirmedPrivateTransfer Service Primitives

Parameter Name	Req	Ind	Rsp	Conf
Argument	M	M(=)		
Vendor ID	M	M(=)		
Service Number	M	M(=)		
Service Parameters	U	U(=)		
Result(+)			S	S(=)
Vendor ID			M	M(=)
Service Number			M	M(=)
Result Block			C	C(=)
Result(-)			S	S(=)
Error Type			M	M(=)
Vendor ID			M	M(=)
Service Number			M	M(=)
Error Parameters			M	M(=)

16.2.1.1 Argument

This parameter shall convey the parameters for the ConfirmedPrivateTransfer confirmed service request.

16.2.1.1.1 Vendor ID

This parameter, of type Unsigned, shall specify the unique vendor identification code for the type of vendor-proprietary service to be performed.

16.2.1.1.2 Service Number

This parameter, of type Unsigned, shall specify the desired service to be performed.

16.2.1.1.3 Service Parameters

This optional parameter shall convey additional parameters for the service specified by 'Vendor ID' and 'Service Number'. The datatype and interpretation of these parameters is a local matter.

16.2.1.2 Result(+)

The 'Result(+)' parameter shall indicate that the service request succeeded. A successful result indicates that the request APDU was received and the recipient was able to perform the indicated proprietary service.

16.2.1.2.1 Vendor ID

This parameter, of type Unsigned, shall specify the unique vendor identification code for the vendor-proprietary service for which this is the result.

16.2.1.2.2 Service Number

This parameter, of type Unsigned, shall indicate the proprietary service for which this is the result.

16.2.1.2.3 Result Block

This conditional parameter, of type List of ANY, shall convey any additional results from the execution of the requested service. Interpretation of these results is a local matter.

16.2.1.3 Result(-)

The 'Result(-)' parameter shall indicate that the service request has failed. The reason for failure shall be specified by the 'Error Type' parameter.

16.2.1.3.1 Error Type

This parameter consists of two component parameters: (1) the 'Error Class' and (2) the 'Error Code'. See Clause 18.

16.2.1.3.2 Vendor ID

This parameter, of type Unsigned, shall specify the unique vendor identification code for the vendor-proprietary service for which this error is the result.

16.2.1.3.3 Service Number

This parameter, of type Unsigned, shall indicate the proprietary service for which this error is the result.

16.2.1.3.4 Error Parameters

This optional parameter shall convey any additional error results from the execution of the requested service. Interpretation of these results is a local matter.

16.2.2 Service Procedure

After verifying the validity of the request, the responding BACnet-user shall attempt to perform the specified proprietary service request. If successful, a 'Result(+)' response primitive shall be issued. If the request fails, a 'Result(-)' response primitive shall be issued.

16.3 UnconfirmedPrivateTransfer Service

The UnconfirmedPrivateTransfer is used by a client BACnet-user to invoke proprietary or non-standard services in a remote device. The specific proprietary services that may be provided by a given device are not defined by this standard. The PrivateTransfer services provide a mechanism for specifying a particular proprietary service in a standardized manner. The only required parameters for these services are a vendor identification code and a service number. Additional parameters may be supplied for each service if required. The form and content of these additional parameters, if any, are not defined by this standard. The vendor identification code and service number together serve to unambiguously identify the intended purpose of the information conveyed by the remainder of the APDU or the service to be performed by the remote device based on parameters in the remainder of the APDU.

The vendor identification code is a unique code assigned to the vendor by ASHRAE. The mechanism for administering these codes is not defined in this standard. See Clause 23.

16.3.1 UnconfirmedPrivateTransfer Service Structure

The structure of the UnconfirmedPrivateTransfer service primitive is shown in Table 16-3. The terminology and symbology used in this table are explained in 5.6.

Table 16-3. Structure of UnconfirmedPrivateTransfer Service Primitive

Parameter Name	Req	Ind
Argument	M	M(=)
Vendor ID	M	M(=)
Service Number	M	M(=)
Service Parameters	U	U(=)

16.3.1.1 Argument

This parameter shall convey the parameters for the UnconfirmedPrivateTransfer confirmed service request.

16.3.1.1.1 Vendor ID

This parameter, of type Unsigned, shall specify the unique vendor identification code for the type of vendor-proprietary service to be performed.

16.3.1.1.2 Service Number

This parameter, of type Unsigned, shall specify the desired service to be performed.

16.3.1.1.3 Service Parameters

This optional parameter shall convey additional parameters for the service specified by 'Vendor ID' and 'ServiceNumber'. The datatype and interpretation of these parameters is a local matter.

16.3.2 Service Procedure

Since this is an unconfirmed service, no response primitives are expected. Actions taken in response to this service request are a local matter.

16.4 ReinitializeDevice Service

The ReinitializeDevice service is used by a client BACnet-user to instruct a remote device to reboot itself (cold start), reset itself to some predefined initial state (warm start), or to control the backup or restore procedure. Resetting or rebooting a device is primarily initiated by a human operator for diagnostic purposes. Use of this service during the backup or restore procedure is usually initiated on behalf of the user by the device controlling the backup or restore. Due to the sensitive nature of this service, a password may be required from the responding BACnet-user prior to executing the service.

16.4.1 Structure

The structure of the ReinitializeDevice service primitives is shown in Table 16-4. The terminology and symbology used in this table are explained in 5.6.

Table 16-4. Structure of ReinitializeDevice Service Primitives

Parameter Name	Req	Ind	Rsp	Cnf
Argument	M	M(=)		
Reinitialized State of Device	M	M(=)		
Password	U	U(=)		
Result (+)			S	S(=)
Result (-)			S	S(=)
Error Type			M	M(=)

16.4.1.1 Argument

This parameter shall convey the parameters for the ReinitializeDevice confirmed service request.

16.4.1.1.1 Reinitialized State of Device

This parameter allows the client user to specify the desired state of the device after its reinitialization. The value of the parameter may be one of COLDSTART, WARMSTART, STARTBACKUP, ENDBACKUP, STARTRESTORE, ENDRESTORE, or ABORTRESTORE.. WARMSTART shall mean to reboot the device and start over, retaining all data and programs that would normally be retained during a brief power outage. The precise interpretation of COLDSTART shall be defined by the vendor.

The use of the backup and restore commands are defined in 19.1.

16.4.1.1.2 Password

This optional parameter shall be a CharacterString of up to 20 characters. For those devices that require the password as a protection, the service request shall be denied if the parameter is absent or if the password is incorrect. For those devices that do not require a password, this parameter shall be ignored.

16.4.1.2 Result(+)

This parameter shall indicate that the service request succeeded.

16.4.1.3 Result(-)

This parameter shall indicate that the service request has failed. The reason for the failure shall be specified by the 'Error Type' parameter.

16.4.1.3.1 Error Type

This parameter consists of two component parameters: (1) the 'Error Class' and (2) the 'Error Code'. See Clause 18.

16.4.2 Service Procedure

After verifying the validity of the request, including the password, the responding BACnet-user shall pre-empt all other outstanding requests and respond with a 'Result(+)' primitive. If the request is WARMSTART or COLDSTART the responding BACnet-user will immediately proceed to perform any applicable shut-down procedures prior to reinitializing

the device as specified by the requesting BACnet-user in the request. If the service request is for WARMSTART and the device is not ready due to its initial characterization being in progress, a 'Result (-)' response primitive shall be issued.

If the requested state is one of STARTBACKUP, ENDBACKUP, STARTRESTORE, ENDRESTORE, or ABORTRESTORE, then the device shall behave as described in 19.1.

If the password is invalid or is absent when one is required, a 'Result (-)' response primitive shall be issued.

16.5 ConfirmedTextMessage Service

The ConfirmedTextMessage service is used by a client BACnet-user to send a text message to another BACnet device. This service is not a broadcast or multicast service. This service may be used in cases when confirmation that the text message was received is required. The confirmation does not guarantee that a human operator has seen the message. Messages may be prioritized into normal or urgent categories. In addition, a given text message may be optionally classified by a numeric class code or class identification string. This classification may be used by the receiving BACnet device to determine how to handle the text message. For example, the message class might indicate a particular output device on which to print text or a set of actions to take when the text is received. In any case, the interpretation of the class is a local matter.

16.5.1 ConfirmedTextMessage Service Structure

The structure of the ConfirmedTextMessage service primitives is shown in Table 16-5. The terminology and symbology used in this table are explained in 5.6.

Table 16-5. Structure of ConfirmedTextMessage Service Primitives

Parameter Name	Req	Ind	Rsp	Cnf
Argument	M	M(=)		
Text Message Source Device	M	M(=)		
Message Class	U	U(=)		
Message Priority	M	M(=)		
Message	M	M(=)		
Result(+)			S	S(=)
Result(-)			S	S(=)
Error Type			M	M(=)

16.5.1.1 Argument

This parameter shall convey the parameters for the ConfirmedTextMessage service request.

16.5.1.1.1 Text Message Source Device

This parameter, of type BACnetObjectIdentifier, shall convey the value of the Object_Identifier property of the Device object of the device that initiated this text message.

16.5.1.1.2 Message Class

This parameter, if present, shall indicate the class of the received message. The datatype of this parameter shall be a choice of Unsigned or CharacterString. The interpretation of the meaning of any particular value for this parameter shall be a local matter.

16.5.1.1.3 Message Priority

This parameter, of type ENUMERATED, shall indicate the priority for message handling:

{NORMAL, URGENT}.

16.5.1.1.4 Message

This parameter, of type CharacterString, shall be used to convey the text message.

16.5.1.2 Result(+)

The 'Result(+)' parameter shall indicate that the requested service has succeeded.

16.5.1.3 Result(-)

The 'Result(-)' parameter shall indicate that the requested service has failed. The reason for the failure shall be specified by the 'Error Type' parameter.

16.5.1.3.1 Error Type

This parameter shall consist of two component parameters: (1) the 'Error Class' and (2) the 'Error Code'. See Clause 18.

16.5.2 ConfirmedTextMessage Service Procedure

After verifying the validity of the request, the responding BACnet-user shall take whatever local actions have been assigned to the indicated 'Message Class' and issue a 'Result(+)' service primitive. If the service request cannot be executed, a 'Result(-)'service primitive shall be issued indicating the encountered error.

Other than the requirement to return a success or failure response, actions taken in response to this notification are a local matter. However, typically the receiving device would take the text specified by the 'Message' parameter and display, print, or file it according to the classification specified by the 'Message Class' parameter. If the 'Message Class' parameter is omitted, then some general class might be assumed. If 'Message Priority' is URGENT, then clearly the messages should be considered as more important than existing NORMAL messages, which may be awaiting printing or some other action.

16.6 UnconfirmedTextMessage Service

The UnconfirmedTextMessage service is used by a client BACnet-user to send a text message to one or more BACnet devices. This service may be broadcast, multicast, or addressed to a single recipient. This service may be used in cases where confirmation that the text message was received is not required. Messages may be prioritized into normal or urgent categories. In addition, a given text message may optionally be classified by a numeric class code or class identification string. This classification may be used by receiving BACnet devices to determine how to handle the text message. For example, the message class might indicate a particular output device on which to print text or a set of actions to take when the text message is received. In any case, the interpretation of the class is a local matter.

16.6.1 UnconfirmedTextMessage Service Structure

The structure of the UnconfirmedTextMessage service primitive is shown in Table 16-6. The terminology and symbology used in this table are explained in 5.6.

Table 16-6. Structure of UnconfirmedTextMessage Service Primitive

Parameter Name	Req	Ind
Argument	M	M(=)
Text Message Source Device	M	M(=)
Message Class	U	U(=)
Message Priority	M	M(=)
Message	M	M(=)

The 'Argument' parameter shall convey the parameters for the UnconfirmedTextMessage service request.

16.6.1.1 Text Message Source Device

This parameter, of type BACnetObjectIdentifier, shall convey the value of the Object_Identifier property of the Device object of the device that initiated this text message.

16.6.1.2 Message Class

This parameter, if present, shall indicate the classification of the received message. The datatype of this parameter shall be a choice of Unsigned or CharacterString. The interpretation of the meaning of any particular value for this parameter shall be a local matter.

16.6.1.3 Message Priority

This parameter, of type ENUMERATED, shall indicate the priority for message handling:

{NORMAL, URGENT}.

16.6.1.4 Message

This parameter, of type CharacterString, shall be used to convey the text message.

16.6.2 UnconfirmedTextMessage Service Procedure

Since this is an unconfirmed service, no response primitives are expected. Actions taken in response to this service request are a local matter. However, typically the receiving device(s) would take the text block specified by the 'Message' parameter and display or print or file them according to the classification specified by the 'Message Class' parameter. If the 'Message Class' parameter is omitted, then some general class might be assumed. If 'Message Priority' is URGENT, then clearly the messages should be considered as more important than existing NORMAL messages, which may be awaiting printing or some other action.

16.7 TimeSynchronization Service

The TimeSynchronization service is used by a requesting BACnet-user to notify a remote device of the correct current time. This service may be broadcast, multicast, or addressed to a single recipient. Its purpose is to notify recipients of the correct current time so that devices may synchronize their internal clocks with one another.

16.7.1 Structure

The structure of the TimeSynchronization service primitive is shown in Table 16-7. The terminology and symbology used in this table are explained in 5.6.

Table 16-7. Structure of TimeSynchronization Service Primitive

Parameter Name	Req	Ind
Argument	M	M(=)
Time	M	M(=)

16.7.1.1 Argument

The 'Argument' parameter shall convey the parameters for the TimeSynchronization service request.

16.7.1.1.1 Time

This parameter, of type BACnetDateTime, shall convey the current date and time as determined by the clock in the device issuing the service request.

16.7.2 Service Procedure

Since this is an unconfirmed service, no response primitives are expected. A device receiving a TimeSynchronization service indication shall update its local representation of time. This change shall be reflected in the Local_Time and Local_Date properties of the Device object.

No restrictions on the use of this service exist when it is invoked at the request of an operator. Otherwise, the use of this service is controlled by the value of the Time_Synchronization_Recipients property in the Device object. When the Time_Synchronization_Recipients list is of length zero, a device may not automatically send a TimeSynchronization request. When Time_Synchronization_Recipients list is of length one or more, a device may automatically send a TimeSynchronization request but only to the devices or addresses contained in the Time_Synchronization_Recipients list.

16.8 UTCTimeSynchronization Service

The UTCTimeSynchronization service is used by a requesting BACnet-user to notify one or more remote devices of the correct Universal Time Coordinated (UTC). This service may be broadcast, multicast, or addressed to a single recipient. Its purpose is to notify recipients of the correct UTC so that devices may synchronize their internal clocks with one another.

16.8.1 Structure

The structure of the UTCTimeSynchronization service primitive is shown in Table 16-8. The terminology and symbology used in this table are explained in 5.6.

Table 16-8. Structure of UTCTimeSynchronization Service Primitive

Parameter Name	Req	Ind
Argument	M	M(=)
Time	M	M(=)

16.8.1.1 Argument

The 'Argument' parameter shall convey the parameters for the UTCTimeSynchronization service request.

16.8.1.1.1 Time

This parameter, of type BACnetDateTime, shall convey the UTC date and time.

16.8.2 Service Procedure

Since this is an unconfirmed service, no response primitives are expected. A device receiving a UTCTimeSynchronization service indication shall update its local representation of time and date by subtracting the value of the 'UTC_Offset' property of the Device object from the 'Time' parameter and taking the 'Daylight_Savings_Status' property of the Device object into account as appropriate to the locality. This change shall be reflected in the Local_Time and Local_Date properties of the Device object.

No restrictions on the use of this service exist when it is invoked at the request of an operator. Otherwise, the initiation of this service by a device is controlled by the value of the Time_Synchronization_Recipients property in the Device object. When the Time_Synchronization_Recipients list is of length zero, a device may not automatically send a TimeSynchronization request. When Time_Synchronization_Recipients list is of length one or more, a device may automatically send a UTCTimeSynchronization request but only to the devices or addresses contained in the Time_Synchronization_Recipients list.

16.9 Who-Has and I-Have Services

The Who-Has service is used by a sending BACnet-user to identify the device object identifiers and network addresses of other BACnet devices whose local databases contain an object with a given Object_Name or a given Object_Identifier. The I-Have service is used to respond to Who-Has service requests or to advertise the existence of an object with a given Object_Name or Object_Identifier. The I-Have service request may be issued at any time and does not need to be preceded by the receipt of a Who-Has service request. The Who-Has and I-Have services are unconfirmed services.

16.9.1 Who-Has Service Structure

The structure of the Who-Has service primitive is shown in Table 16-9. The terminology and symbology used in this table are explained in 5.6.

Table 16-9. Structure of Who-Has Service Primitive

Parameter Name	Req	Ind
Argument	M	M(=)
Device Instance Range Low Limit	U	U(=)
Device Instance Range High Limit	U	U(=)
Object Identifier	S	S(=)
Object Name	S	S(=)

16.9.1.1 Argument

The 'Argument' parameter shall convey the parameters for the Who-Has unconfirmed service request.

16.9.1.1.1 Device Instance Range Low Limit

This parameter is an unsigned integer in the range 0 - 4,194,303. In conjunction with the 'Device Instance Range High Limit' parameter, it defines the devices that are qualified to respond with an I-Have service request if the 'Object Identifier' or 'Object Name' criteria are satisfied as described in 16.9.1.1.3 and 16.9.1.1.4. If the 'Device Instance Range Low Limit' parameter is present, then the 'Device Instance Range High Limit' parameter shall also be present, and only those devices whose Device Object_Identifier instance number falls within the range 'Device Instance Range Low Limit'≤ Object_Identifier Instance Number ≤ 'Device Instance Range High Limit' shall be qualified to respond. The value of the 'Device Instance Range Low Limit' shall be less than or equal to the value of the 'Device Instance Range High Limit'. If the 'Device Instance Range Low Limit' and 'Device Instance Range High Limit' parameters are omitted, then all devices that receive this message are qualified to respond with an I-Have service request.

16.9.1.1.2 Device Instance Range High Limit

This parameter is an unsigned integer in the range 0 - 4,194,303. In conjunction with the 'Device Instance Range Low Limit' parameter, it defines the devices that are qualified to respond with an I-Have service request if the 'Object Identifier' or 'Object Name' criteria are satisfied as described in 16.9.1.1.3 and 16.9.1.1.4. If the 'Device Instance Range High Limit' parameter is present, then the 'Device Instance Range Low Limit' parameter shall also be present, and only those devices whose Device Object_Identifier instance number falls within the range 'Device Instance Range Low Limit'≤ Object_Identifier Instance Number ≤ 'Device Instance Range High Limit' shall be qualified to respond. The value of the 'Device Instance Range High Limit' shall be greater than or equal to the value of the 'Device Instance Range Low Limit'. If the 'Device Instance Range Low Limit' and 'Device Instance Range High Limit' parameters are omitted, then all devices that receive this message are qualified to respond with an I-Have service request.

16.9.1.1.3 Object Identifier

The 'Object Identifier' parameter, of type BACnetObjectIdentifier, shall convey the Object_Identifier of the object that is to be located. If the 'Object Identifier' parameter is omitted, then the 'Object Name' parameter shall be present. If the 'Object Identifier' parameter is present, then only those devices that contain an object with an Object_Identifier property value matching the 'Object Identifier' parameter, which are qualified to respond as described in 16.9.1.1.1 and 16.9.1.1.2, shall respond with an I-Have service request.

16.9.1.1.4 Object Name

The 'Object Name' parameter, of type CharacterString, shall convey the value of the Object_Name property of the object that is to be located. If the 'Object Name' parameter is omitted, then the 'Object Identifier' parameter shall be present. If the 'Object Name' parameter is present, then only those devices that contain an object with an Object_Name property value matching the 'Object Name' parameter, which are qualified to respond as described in 16.9.1.1.1 and 16.9.1.1.2, shall respond with an I-Have service request.

16.9.2 Who-Has Service Procedure

The sending BACnet-user shall transmit the Who-Has unconfirmed request, normally using a broadcast address. If the 'Device Instance Range Low Limit' and 'Device Instance Range High Limit' parameters are present, then only those receiving BACnet-users whose Device Object_Identifier instance number falls in the range 'Device Instance Range Low Limit ≤ Object_Identifier Instance Number ≤ 'Device Instance Range High Limit' shall be qualified to respond. If the 'Object Name' parameter is present, then only those qualified receiving BACnet-users that contain an object with an Object_Name property value matching the 'Object Name' parameter shall respond with an I-Have service request. If the 'Object Identifier' parameter is present, then only those qualified receiving BACnet-users that contain an object with an Object_Identifier property value matching the 'Object Identifier' parameter shall respond with an I-Have service request.

16.9.3 I-Have Service Structure

The structure of the I-Have service primitive is shown in Table 16-10. The terminology and symbology used in this table are explained in 5.6.

Table 16-10. Structure of I-Have Service Primitive

Parameter Name	Req	Ind
Argument	M	M(=)
Device Identifier	M	M(=)
Object Identifier	M	M(=)
Object Name	M	M(=)

16.9.3.1 Argument

The 'Argument' parameter shall convey the parameters for the I-Have unconfirmed service request.

16.9.3.1.1 Device Identifier

The 'Device Identifier' parameter, of type BACnetObjectIdentifier, is the Device Object_Identifier of the device initiating the I-Have service request.

16.9.3.1.2 Object Identifier

The 'Object Identifier' parameter, of type BACnetObjectIdentifier, shall convey the Object_Identifier of the object that is being advertised as located in the initiating device. This identifier shall correspond to the value of the Object_Identifier property of the object being advertised.

16.9.3.1.3 Object Name

The 'Object Name' parameter, of type CharacterString, shall convey the name of the object that is being advertised as located in the initiating device. This name shall correspond to the value of the Object_Name property of the object being advertised.

16.9.4 I-Have Service Procedure

The sending BACnet-user shall broadcast the I-Have unconfirmed request. Such broadcasts may be on the local network only, a remote network only, or globally on all networks at the discretion of the application. If the I-Have is being transmitted in response to a previously received Who-Has, then the I-Have shall be transmitted in such a manner that the BACnet-user that sent the Who-Has will receive the resulting I-Have. Since the request is unconfirmed, no further action is required. A BACnet-user may issue an I-Have service request at any time.

16.10 Who-Is and I-Am Services

The Who-Is service is used by a sending BACnet-user to determine the device object identifier, the network address, or both, of other BACnet devices that share the same internetwork. The Who-Is service is an unconfirmed service. The Who-Is service may be used to determine the device object identifier and network addresses of all devices on the network, or to determine the network address of a specific device whose device object identifier is known, but whose address is not. The I-Am service is also an unconfirmed service. The I-Am service is used to respond to Who-Is service requests. However, the I-Am service request may be issued at any time. It does not need to be preceded by the receipt of a Who-Is service request. In particular, a device may wish to broadcast an I-Am service request when it powers up. The network address is derived either from the MAC address associated with the I-Am service request, if the device issuing the request is on the local network, or from the NPCI if the device is on a remote network.

16.10.1 Who-Is Service Structure

The structure of the Who-Is service primitive is shown in Table 16-11. The terminology and symbology used in this table are explained in 5.6.

Table 16-11. Structure of Who-Is Service Primitive

Parameter Name	Req	Ind
Argument	M	M(=)
Device Instance Range Low Limit	U	U(=)
Device Instance Range High Limit	U	U(=)

16.10.1.1 Argument

The 'Argument' parameter shall convey the parameters for the Who-Is unconfirmed service request.

16.10.1.1.1 Device Instance Range Low Limit

This parameter is an unsigned integer in the range 0 - 4,194,303. In conjunction with the 'Device Instance Range High Limit' parameter, it defines the devices that are qualified to respond with an I-Am service request. If the 'Device Instance Range Low Limit' parameter is present, then the 'Device Instance Range High Limit' parameter shall also be present, and only those devices whose Device Object_Identifier instance number falls within the range 'Device Instance Range Low Limit' ≤ Device Object_Identifier Instance Number ≤ 'Device Instance Range High Limit' shall be qualified to respond. The value of the 'Device Instance Range Low Limit' shall be less than or equal to the value of the 'Device Instance Range High Limit'. If the 'Device Instance Range Low Limit' and 'Device Instance Range High Limit' parameters are omitted, then all devices that receive this message are qualified to respond with an I-Am service request.

16.10.1.1.2 Device Instance Range High Limit

This parameter is an unsigned integer in the range 0 - 4,194,303. In conjunction with the 'Device Instance Range Low Limit' parameter, it defines the devices that are qualified to respond with an I-Am service request. If the 'Device Instance Range High Limit' parameter is present, then the 'Device Instance Range Low Limit' parameter shall also be present, and only those devices whose Device Object_Identifier instance number falls within the range 'Device Instance Range Low Limit' ≤ Device Object_Identifier Instance Number ≤ 'Device Instance Range High Limit' shall be qualified to respond. The value of the 'Device Instance Range High Limit' shall be greater than or equal to the value of the 'Device Instance Range Low Limit'. If the 'Device Instance Range Low Limit' and 'Device Instance Range High Limit' parameters are omitted, then all devices that receive this message are qualified to respond with an I-Am service request.

16.10.2 Who-Is Service Procedure

The sending BACnet-user shall transmit the Who-Is unconfirmed request, normally using a broadcast address. If the 'Device Instance Range Low Limit' and 'Device Instance Range High Limit' parameters are omitted, then all receiving BACnet-users shall return their Device Object_Identifier in individual responses using the I-Am service. If the 'Device Instance Range Low Limit' and 'Device Instance Range High Limit' parameters are present, then only those receiving BACnet-users whose Device Object_Identifier instance number falls within the range 'Device Instance Range Low Limit ≤ Device Object_Identifier Instance Number ≤ 'Device Instance Range High Limit' shall return their Device Object_Identifier using the I-Am service.

If the receiving BACnet-user has a Slave_Proxy_Enable property and the Slave_Proxy_Enable for the receiving port is TRUE, then the BACnet-user shall respond with an I-Am unconfirmed request for each of the slave devices on the MS/TP network that are present in the Slave_Address_Binding property and that match the device range parameters. The I-Am unconfirmed requests that are generated shall be generated as if the slave device originated the service. If the I-Am confirmed request is to be placed onto the MS/TP network on which the slave resides, then the MAC address included in the packet shall be that of the slave device. In the case where the I-Am confirmed request is to be placed onto a network other than that on which the slave resides, then the network layer shall contain SLEN and SNET filled in with the slave's MAC address as if it were routing a packet originally generated by the slave device

16.10.3 I-Am Service Structure

The structure of the I-Am service primitive is shown in Table 16-12. The terminology and symbology used in this table are explained in 5.6.

Table 16-12. Structure of I-Am Service Primitive

Parameter Name	Req	Ind
Argument	M	M(=)
I-Am Device Identifier	M	M(=)
Max APDU Length Accepted	M	M(=)
Segmentation Supported	M	M(=)
Vendor Identifier	M	M(=)

16.10.3.1 Argument

The 'Argument' parameter shall convey the parameters for the I-Am unconfirmed service request.

16.10.3.1.1 I-Am Device Identifier

The 'I-Am Device Identifier' parameter, of type BACnetObjectIdentifier, is the Device Object_Identifier of the device initiating the I-Am service request.

16.10.3.1.2 Max APDU Length Accepted

This parameter, of type Unsigned, shall convey the maximum number of octets that may be contained in a single, indivisible APDU. The value of this parameter shall be the same as the value of the Max_APDU_Length_Accepted property of the Device object. See 12.11.17.

16.10.3.1.3 Segmentation Supported

This parameter, of type BACnetSegmentation, conveys the capabilities of the device initiating the I-Am service request with respect to processing segmented messages. The value of this parameter shall be the same as the value of the Segmentation_Supported property of the Device object. See 12.11.18.

16.10.3.1.4 Vendor Identifier

This parameter, of type Unsigned16, shall convey the identity of the vendor who manufactured the device initiating the I-Am service request. The value of this parameter shall be the same as the value of the Vendor_Identifier property of the Device object. See 12.11.6 and Clause 23.

16.10.4 I-Am Service Procedure

The sending BACnet-user shall broadcast the I-Am unconfirmed request. This broadcast may be on the local network only, a remote network only, or globally on all networks at the discretion of the application. If the I-Am is being broadcast in response to a previously received Who-Is, then the I-Am shall be broadcast in such a manner that the BACnet-user that sent the Who-Is will receive the resulting I-Am. Since the request is unconfirmed, no further action is required. A BACnet-user may issue an I-Am service request at any time.

17 VIRTUAL TERMINAL SERVICES

Virtual Terminal (VT) services are used by a client BACnet-user to establish a connection to an application program server in another BACnet device. The purpose of this connection is to facilitate the bi-directional exchange of character-oriented data. Normally, these services would be used to permit an application program in one BACnet device to act as a "terminal emulator" that interacts with an "operator interface" application program in another BACnet device.

These connections will be referred to here as VT-sessions. Once a VT-session is established, both the client application program and server application program will be referred to as VT-users.

The VT services provide the following features and services to the VT-user:

(a) the means to establish a VT-session between two peer VT-users for the purpose of enabling virtual terminal information exchange;

(b) the means to select between different VT-class types, including character repertoire and encoding;

(c) the means to control the integrity of the communication;

(d) the means to terminate the VT-session unilaterally;

(e) the means to exchange virtual terminal data.

17.1 Virtual Terminal Model

Each VT-session is a bi-directional connection between two peer application processes. Once a session is established between these peers, data are exchanged through the use of Virtual Terminal Queues (VTQ). The VTQs are first-in, first-out (FIFO) queues. The purpose of modeling the data flow between peer processes as FIFO queues is to isolate the implementation of the peer application process that is on each side of the VT-session from the other. Normally a human operator using a BACnet device will request the operator interface application program to establish a VT-session with an operator interface application program in another BACnet device. The VTQ model uses two FIFO queues to allow those operator interfaces, or other application programs that can provide simultaneous bi-directional interaction, to do so through the BACnet protocol.

Figure 17-1 shows a typical relationship between an operator interface application program and a physical device, such as a CRT terminal.

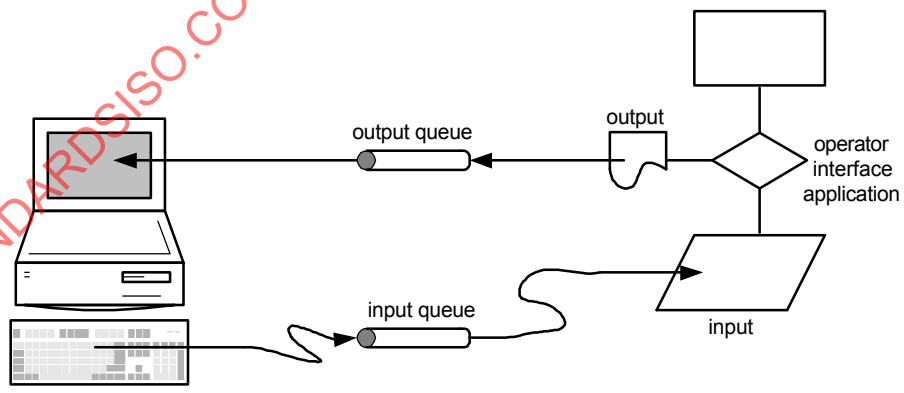


Figure 17-1. Relationship between an operator interface program and a physical device.

Figure 17-2 shows how this model is extended using the VTQ concept.

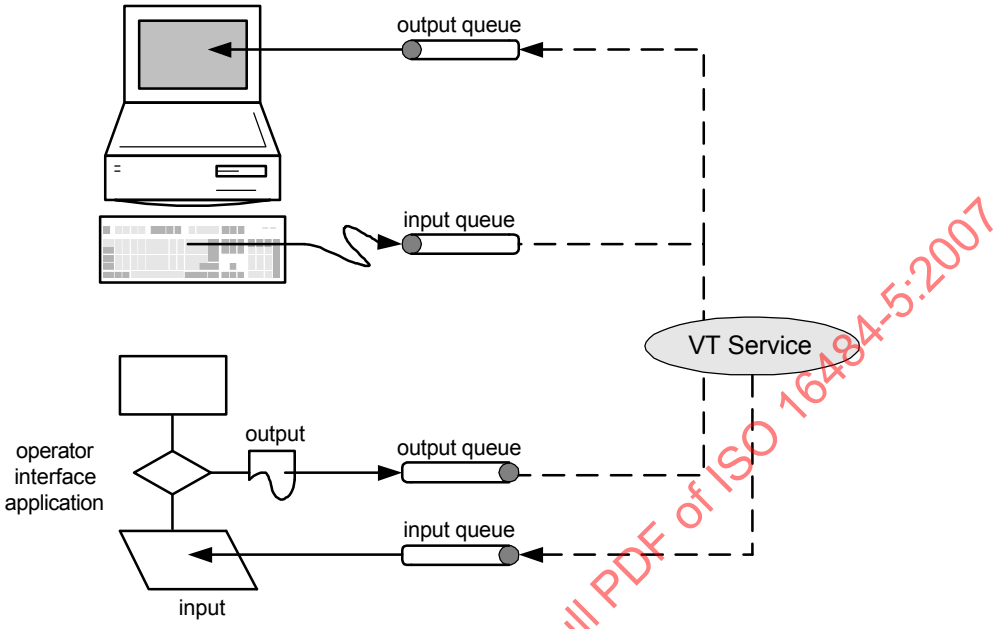


Figure 17-2. Extending the VT model to accommodate queues.

The peer processes at each end of a VT-session may not actually be agents for a physical device such as a CRT terminal. The VTQ model permits flexibility in the implementation of each BACnet device. There may, in fact, be several different processes that coordinate their use of VT Services within each BACnet device. For example, in a multi-window operator interface program, there may be several windows, each with its own interactive virtual terminal session to some other BACnet device. For this reason, each VT-session exists between two unique processes rather than between two BACnet devices. Each session, therefore, consists of two processes that act as agents for their respective application programs and their respective VTQs. These agent processes and their VTQs are called VT-Users. The model of this data flow is shown in Figure 17-3.

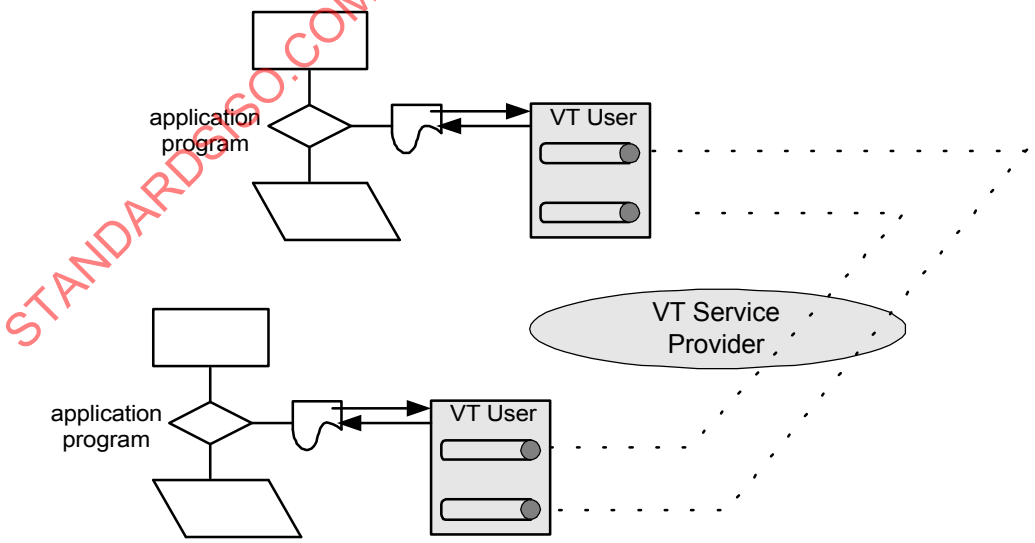


Figure 17-3. Virtual terminal data flow.

Once the virtual terminal session is established, character data are exchanged by the two peers through their respective VTQs. Normally, characters typed by a human operator would be passed directly to an input queue for forwarding to the peer's input queue, without any echoing by the local device to which the operator is connected. The peer application program, upon receiving these characters from its input queue, would respond with characters placed in its output queue for forwarding back to the output queue of the operator device and so on. In particular, it shall be the responsibility of the operator interface application to generate all "screen" output, including carriage control and character echoing when appropriate.

Although the VT services model does provide a true peer-to-peer connection, as shown in Figure 17-3, a human operator typically uses one BACnet device to establish a virtual terminal connection to a second BACnet device. This second BACnet device contains an "operator interface" application program to which the operator's keystrokes are sent without filtering. The operator interface application program's output is conveyed through the VT services and ultimately displayed for the human operator. Normally, the BACnet device to which the operator is actually connected would recognize some local signal meaning "end virtual terminal session" but otherwise would not filter the operator's keystrokes.

17.1.1 Basic Services

There are three basic services provided: VT-Open, VT-Close, and VT-Data. The VT-Open service is used to establish a VT-session between peer processes. The VT-Close service is used to terminate a previously established session. The VT-Data service is used to exchange data between peer processes.

17.1.2 VT-classes

The classes of virtual terminal that are available in a peer VT service may be determined by examining the BACnet Device object in the peer device. The BACnet Device object has a property called VT_Classes_Supported, which may be read with the ReadProperty, ReadPropertyMultiple, or ReadPropertyConditional services to determine which VT-classes are available in that device.

17.1.3 Active VT-sessions

The active VT-sessions within a peer VT service may be determined by examining the BACnet Device object in the peer device. The BACnet Device object has a property called Active_VT_Sessions, which may be read with the ReadProperty, ReadPropertyMultiple, or ReadPropertyConditional services to determine which VT-session IDs are in use within that device.

17.1.4 State Diagram for VT-Open, VT-Data, and VT-Close

There are three phases of operation within a VT-session context: IDLE, DATA EXCHANGE, and HOLD. In the IDLE phase, no VT-session exists. Once a VT-session is created through the use of the VT-Open service, the VT context enters the DATA EXCHANGE phase. The VT context remains in the DATA EXCHANGE phase until one of two events occurs:

(a) a successful VT-Close is performed, terminating the VT-session context, or
(b) a VT-Data request returns 'Result (-)'.

The HOLD phase occurs when a VT-Data request cannot be confirmed. Figure 17-4 shows the relationship between phases.

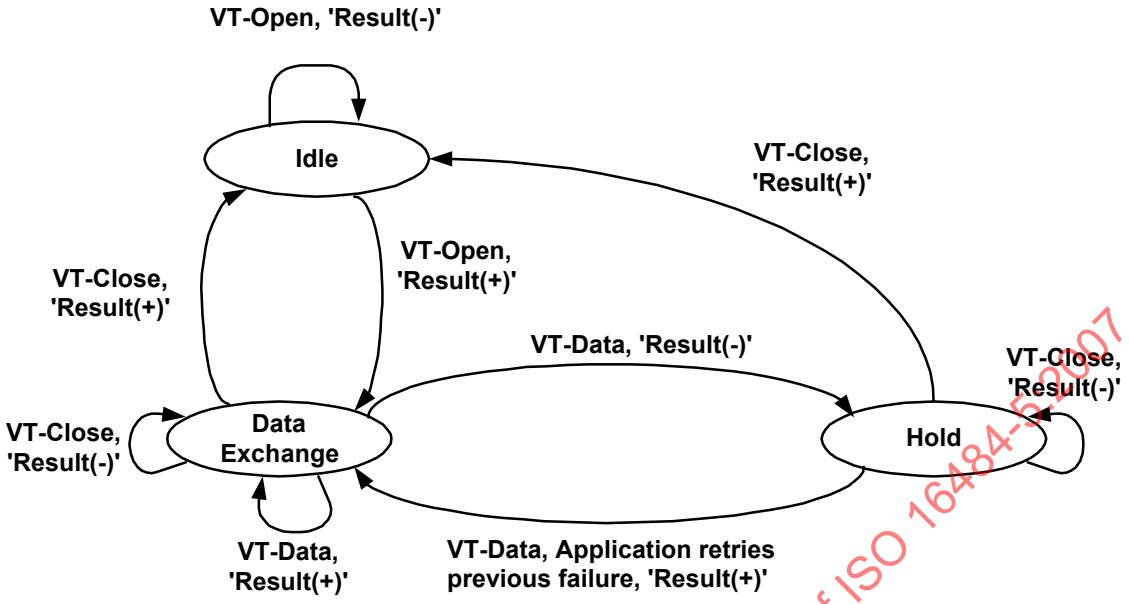


Figure 17-4. Virtual terminal services state diagram.

17.1.5 VT Session Synchronization

Each of the peers participating in a VT-session shall maintain two VT-data flags that are used to synchronize the VT-session. One flag represents the device's role as a sending BACnet-user and is the sequence number, which that alternates between zero and one, of the next VT-data that is to be sent. This sequence number is incremented modulo 2 upon receipt of a 'Result(+)' response to a VT-Data request. This flag is initialized to 0 (the first VT-Data request uses a sequence number of 0) when the VT-session is established.

The other flag represents the device's role as a receiving BACnet-user and is the sequence number of the last VT-Data request that was correctly received. This sequence number is initialized to 1 (the next VT-Data indication is expected to have a sequence number of 0) when the VT session is established. The sequence number is incremented modulo 2 when a VT-Data indication with the expected sequence number is received and successfully processed.

The receiving VT-session context shall also store the last received 'All New Data Accepted' and 'Accepted Octet Count'. This is required in the event that a 'Result(+)' response to a VT-Data indication is lost, which would be detected by the receipt of a VT-Data indication with the same 'VT-Data Flag' as the previously received 'VT-Data Flag' saved in the VT session context.

17.1.6 VT Session Identifiers

Associated with each VT-session are two session identifiers, a 'Local VT Session Identifier' and a 'Remote VT Session Identifier'. These session identifiers provide a way to associate the data from a particular VT-Data request with the correct process. The value of the session identifiers is established as part of the VT-Open service. Each device selects its own 'Local VT Session Identifier', which shall be unique to all active VT-sessions in the device, without regard to whether the device's role in the session is a client or a server. The appropriate 'Remote VT Session Identifier' is obtained from the VT-Open service parameters.

When VT data are conveyed to a remote device, the 'Remote VT Session Identifier' is conveyed with the data. This session identifier is used by the remote device to identify the correct VT-session.

17.2 VT-Open Service

The VT-Open service is used to establish a VT-session with a peer VT-user. The service request includes a VT-class type that identifies a particular set of assumptions about the character repertoire and encoding to be used with this session.

17.2.1 Structure

The structure of the VT-Open service primitives is shown in Table 17-1. The terminology and symbology used in this table are explained in 5.6.

Table 17-1. Structure of VT-Open Service Primitives

Parameter Name	Req	Ind	Rsp	Cnf
Argument	M	M(=)		
VT-class	M	M(=)		
Local VT Session Identifier	M	M(=)		
Result (+)			S	S(=)
Remote VT Session Identifier			M	M(=)
Result (-)			S	S(=)
Error Type			M	M(=)

17.2.1.1 Argument

This parameter shall convey the parameters for the VT-Open confirmed service request.

17.2.1.1.1 VT-class

This parameter, of type BACnetVTClass, shall identify the name of the desired class of session to be established. The standard enumeration is:

DEFAULT_TERMINAL
ANSI_X3.64
DEC_VT52
DEC_VT100
DEC_VT220
HP_700/94
IBM_3130

The enumeration DEFAULT_TERMINAL shall refer to a terminal with the characteristics defined in 17.5. All VT-users are required to support at least one VT-class, namely DEFAULT_TERMINAL. Other VT-class types may also be supported by a given VT-user. It is possible to discover which VT-class types are supported by reading the VT_Classes_Supported property of the Device object.

17.2.1.1.2 Local VT Session Identifier

The 'Local VT Session Identifier' parameter shall be an unsigned integer in the range 0-255 indicating the unique VT-session in the requesting VT-user, which shall be used to receive VT-Data output from the opened virtual terminal. This identifier becomes the 'Remote VT Session Identifier' to the responding VT-user.

17.2.1.2 Result (+)

The 'Result (+)' parameter shall indicate that the service request succeeded. A successful result includes the following parameter.

17.2.1.2.1 Remote VT Session Identifier

The 'Remote VT Session Identifier' parameter shall be an unsigned integer in the range 0-255 indicating a unique VT-session that exists within the responding VT-user's context. From the perspective of the responding VT-user, this parameter is the 'Local VT Session Identifier'.

17.2.1.3 Result (-)

The 'Result (-)' parameter shall indicate that the service request has failed in its entirety. The reason for the failure shall be specified by the 'Error Type' parameter.

17.2.1.3.1 Error Type

This parameter consists of two component parameters: (1) the 'Error Class' and (2) the 'Error Code'. See Clause 18.

17.2.2 Service Procedure

After verifying the validity of the request, the responding BACnet-user shall attempt to allocate the resources necessary to establish a VT-session. If there is no VT-user that can provide the desired VT-class, then the 'Result (-)' response shall be returned. If there is a VT-user that can provide the desired VT-class, then the responding BACnet-user shall attempt to establish a new VT-session. If the VT-user does not have the resources to establish another session, then the 'Result (-)' response shall be returned. If the VT-user has the resources, a new VT-session shall be created, the local VT-data Flags shall be initialized as specified in 17.1.5, and a new VT-session Identifier shall be returned in the 'Result (+)' response.

17.3 VT-Close Service

The VT-Close service is used to terminate a previously established VT-session with a peer VT-user. The service request may specify a particular VT-session to be terminated or a list of VT-sessions to be terminated.

17.3.1 Structure

The structure of the VT-Close service primitives is shown in Table 17-2. The terminology and symbology used in this table are explained in 5.6.

Table 17-2. Structure of VT-Close Service Primitives

Parameter Name	Req	Ind	Rsp	Cnf
Argument	M	M(=)		
List of Remote VT Session Identifiers	M	M(=)		
Result (+)			S	S(=)
Result (-)			S	S(=)
Error Type			M	M(=)
List of VT Session Identifiers			C	C(=)

17.3.1.1 Argument

This parameter shall convey the parameters for the VT-Close confirmed service request.

17.3.1.1.1 List of Remote VT Session Identifiers

The 'List of Remote VT Session Identifiers' parameter shall consist of a list of one or more 'Remote VT Session Identifiers'. Each 'Remote VT Session Identifier' shall indicate the particular VT-session that is to be terminated.

17.3.1.2 Result (+)

The 'Result (+)' parameter shall indicate that the service request succeeded.

17.3.1.3 Result (-)

The 'Result (-)' parameter shall indicate that the service request has failed in its entirety. The reason for the failure shall be specified by the 'Error Type' parameter.

17.3.1.3.1 Error Type

This parameter consists of two component parameters: (1) the 'Error Class' and (2) the 'Error Code'. See Clause 18.

17.3.1.3.2 List of VT Session Identifiers

If the 'Error Type' parameter returns an 'Error Code' of VT_SESSION_TERMINATION_FAILURE, then this parameter shall be included. If the 'Error Type' parameter indicates some other error, then this parameter shall be omitted. The 'List of VT Session Identifiers' parameter shall consist of a list of one or more 'VT Session Identifiers'. Each 'VT Session Identifier' shall indicate the particular VT-session that could not be terminated. The Session Identifiers returned are the ones that are local with respect to the device receiving the 'Result(-)' primitive, the requesting VT-user.

17.3.2 Service Procedure

After verifying the validity of the request, the responding BACnet-user shall attempt to terminate each VT-session specified by the 'List of Remote VT Session Identifiers' parameter. From the viewpoint of the responding BACnet-user, these are 'Local VT Session Identifiers'. If one or more of the specified VT-sessions cannot be terminated for some reason, then all of the specified sessions that can be terminated shall be terminated and a 'Result (-)' response shall be returned. If all of the specified VT-sessions are successfully terminated, then the 'Result (+)' response shall be returned.

17.4 VT-Data Service

The VT-Data service is used to exchange data with a peer VT-user through a previously established VT-session. The sending BACnet-user provides new input for the peer VT-user, which may accept or reject the new data. If the new data are rejected, then it is up to the sending BACnet-user to retry the request at a later time.

17.4.1 Structure

The structure of the VT-Data service primitives is shown in Table 17-3. The terminology and symbology used in this table are explained in 5.6.

Table 17-3. Structure of VT-Data Service Primitives

Parameter Name	Req	Ind	Rsp	Cnf
Argument	M	M(=)		
VT-session Identifier	M	M(=)		
VT-new Data	M	M(=)		
VT-data Flag	M	M(=)		
Result (+)			S	S(=)
All new Data Accepted			M	M(=)
Accepted Octet Count			C	C(=)
Result (-)			S	S(=)
Error Type			M	M(=)

17.4.1.1 Argument

This parameter shall convey the parameters for the VT-Data confirmed service request.

17.4.1.1.1 VT-session Identifier

The 'VT-session Identifier' parameter shall indicate the particular VT-session to which data will be sent. It is the 'Remote VT Session Identifier' from the perspective of the requesting BACnet-user and the 'Local VT Session Identifier' from the perspective of the responding BACnet-user.

17.4.1.1.2 VT-new Data

The 'VT-new Data' parameter shall specify the octets of new data that are to be sent to the peer VT-user.

17.4.1.1.3 VT-data Flag

The 'VT-data Flag' parameter, of type Unsigned, shall indicate the expected sequence of VT-Data requests. It shall have values of 0 or 1, which alternate with each new VT-Data request for the same VT session.

17.4.1.2 Result (+)

The 'Result (+)' parameter shall indicate that the service request succeeded. A successful result includes the following parameters.

17.4.1.2.1 All New Data Accepted

The 'All New Data Accepted' parameter, of type BOOLEAN, shall be equal to TRUE if all of the 'VT-new Data' octets were accepted by the peer VT-user. In this case, the service shall be considered completed. If the 'All New Data Accepted' parameter is FALSE, then some of the 'VT-new Data' octets were unable to be accepted by the peer VT-user. Typically this could occur because of resource limitations in the peer VT-user. In this case, it is up to the sending BACnet user to re-issue the VT-Data request at a later time, including only those octets that could not be accepted.

17.4.1.2.2 Accepted Octet Count

The 'Accepted Octet Count' parameter shall only be present if the 'All New Data Accepted' parameter is FALSE. In this case, the 'Accepted Octet Count' parameter shall indicate the number of octets that were actually accepted from those presented in the

'VT-new Data' parameter of the service request. If the 'All New Data Accepted' parameter is TRUE, then the 'Accepted Octet Count' parameter shall be omitted.

17.4.1.3 Result (-)

The 'Result (-)' parameter shall indicate that the service request has failed in its entirety. The reason for the failure shall be specified by the 'Error Type' parameter.

17.4.1.3.1 Error Type

This parameter consists of two component parameters: (1) the 'Error Class' and (2) the 'Error Code'. See Clause 18.

17.4.2 Service Procedure

The sending BACnet user shall send the initial VT-Data request using a 'VT-data Flag' value of 0. Subsequent VT-Data requests for the same session, except retries, shall alternate sequence numbers 0 and 1. New VT-Data requests with the alternate sequence number shall not be transmitted until a 'Result(+)' has been received for the previous VT-Data request.

After verifying the validity of the request, the receiving BACnet-user shall attempt to locate the session specified by the 'VT-session Identifier' parameter. If the VT-session cannot be located, then the 'Result (-)' response shall be returned.

If the specified VT-session is found and the received 'VT-Data Flag' is the same as the last received 'VT-Data Flag' for this session, then this is a duplicate VT-Data request. The data shall be discarded, and a 'Result(+)' shall be returned containing the 'All New Data Accepted' and 'Accepted Octet Count' values that have been saved in the VT-session context from the previous VT-Data response.

If the specified VT-session is found and the received 'VT-Data Flag' is different from the last received 'VT-Data Flag' for this session, then this is a new VT-Data request. If all of the data can be added to the session's input queue, then the data shall be queued and a 'Result(+)' shall be returned with 'All New Data Accepted' = TRUE and the 'Accepted Octet Count' absent. If all of the data in the VT-Data request cannot be added to the session's input queue, then as many data as possible shall be queued and the remainder discarded. A 'Result(+)' shall be returned with 'All New Data Accepted' = FALSE and 'Accepted octet Count' equal to the number of octets that were able to be queued. In either case, the returned 'All New Data Accepted' and 'Accepted Octet Count' values shall be saved in the VT-session context.

17.5 Default-terminal Characteristics

Every VT-user shall implement at least one VT-class, DEFAULT_TERMINAL. The default terminal is based on a limited set of functions that are commonly found in all types of interactive terminal devices. The characteristics of the default terminal include a character repertoire, control functions, and page size assumptions. No other assumptions may be made about the behavior of the VT-user.

17.5.1 Default-terminal Character Repertoire

The default terminal character repertoire shall be defined as a particular mapping between single octet values and their associated meanings. The default terminal character repertoire shall include three types of meanings for a given octet value:

(a) a particular symbol (SYMBOL), e.g., "A";
(b) a particular implied control function (CONTROL), e.g., Carriage Return;
(c) a null meaning (NUL), e.g., Unused, shall be ignored.

Those octets specified as NUL within the default terminal character repertoire have no assumed meanings and shall be ignored if they are received by either VT-user. The default terminal character repertoire is a subset of ASCII. Table 17.4 summarizes the octet encodings for each character in the default terminal character repertoire. The "Octet Value" field indicates octet encodings as decimal (base 10) values from 0 to 255. A range of octet values is indicated by two decimal numbers, e.g., 000-006.

Table 17-4. Default Terminal Character Repertoire

Octet Value	Type	Meaning
000-006	NUL	none
007	CONTROL	audible indication (BEL)
008	CONTROL	non-destructive backspace (BS)
009	CONTROL	horizontal tab (TAB)
010	CONTROL	line feed (LF)
011-012	NUL	none
013	CONTROL	carriage return (CR)
014-031	NUL	none
032	SYMBOL	space
033	SYMBOL	! exclamation
034	SYMBOL	" double quote
035	SYMBOL	# pound sign
036	SYMBOL	$ dollar sign
037	SYMBOL	% percent
038	SYMBOL	& ampersand
039	SYMBOL	' apostrophe
040	SYMBOL	(left parenthesis
041	SYMBOL	) right parenthesis
042	SYMBOL	* asterisk
043	SYMBOL	+ plus sign
044	SYMBOL	, comma
045	SYMBOL	- minus sign
046	SYMBOL	. period
047	SYMBOL	/ forward slash

Table 17-4. Default Terminal Character Repertoire *(continued)*

Octet Value	Type	Meaning
048	SYMBOL	0 zero
049	SYMBOL	1 one
050	SYMBOL	2 two
051	SYMBOL	3 three
052	SYMBOL	4 four
053	SYMBOL	5 five
054	SYMBOL	6 six
055	SYMBOL	7 seven
056	SYMBOL	8 eight
057	SYMBOL	9 nine
058	SYMBOL	: colon
059	SYMBOL	; semicolon
060	SYMBOL	< left angle bracket (or less than)
061	SYMBOL	= equal sign
062	SYMBOL	> right angle bracket (or greater than)
063	SYMBOL	? question mark
064	SYMBOL	@ commercial at sign
065	SYMBOL	A
066	SYMBOL	B
067	SYMBOL	C
068	SYMBOL	D
069	SYMBOL	E
070	SYMBOL	F
071	SYMBOL	G
072	SYMBOL	H
073	SYMBOL	I
074	SYMBOL	J
075	SYMBOL	K
076	SYMBOL	L
077	SYMBOL	M
078	SYMBOL	N
079	SYMBOL	O
080	SYMBOL	P
081	SYMBOL	Q
082	SYMBOL	R
083	SYMBOL	S
084	SYMBOL	T
085	SYMBOL	U
086	SYMBOL	V
087	SYMBOL	W
088	SYMBOL	X
089	SYMBOL	Y
090	SYMBOL	Z

Table 17-4. Default Terminal Character Repertoire *(concluded)*

Octet Value	Type	Meaning
091	SYMBOL	[left square bracket
092	SYMBOL	\ back slash
093	SYMBOL	] right square bracket
094	SYMBOL	^ caret
095	SYMBOL	_ underscore
096	SYMBOL	` accent grave
097	SYMBOL	a
098	SYMBOL	b
099	SYMBOL	c
100	SYMBOL	d
101	SYMBOL	e
102	SYMBOL	f
103	SYMBOL	g
104	SYMBOL	h
105	SYMBOL	i
106	SYMBOL	j
107	SYMBOL	k
108	SYMBOL	l
109	SYMBOL	m
110	SYMBOL	n
111	SYMBOL	o
112	SYMBOL	p
113	SYMBOL	q
114	SYMBOL	r
115	SYMBOL	s
116	SYMBOL	t
117	SYMBOL	u
118	SYMBOL	v
119	SYMBOL	w
120	SYMBOL	x
121	SYMBOL	y
122	SYMBOL	z
123	SYMBOL	{ left curly bracket
124	SYMBOL	| vertical bar
125	SYMBOL	} right curly bracket
126	SYMBOL	~ tilde
127	CONTROL	non-destructive backspace (DEL)
128-255	NUL	none

17.5.2 Control Functions

There are six octet codes within the default terminal character repertoire that perform control functions. In this context, control function means some action that is implied by the receipt of one of these control codes.

17.5.2.1 Octet Code 007

The octet code 007 shall represent an audible indication (BEL). Normally this would be used to sound a tone or bell signal.

17.5.2.2 Octet Codes 008 and 127

The octet code 008 shall represent a non-destructive backspace (BS). This shall cause the cursor of the virtual "output device" to be moved one character position to the left. If the cursor is at the extreme left or beginning of a line, then BS shall have no effect.

This function shall only change the current position, without overwriting or altering any characters that have been previously displayed on the same "line." The octet code 127 shall be considered to be equivalent to 008 and shall have the same effects.

17.5.2.3 Octet Code 013

The octet code 013 shall represent a carriage return (CR). This shall cause the cursor to be reset to the beginning of the current "line." Subsequently received characters would then overwrite existing characters on the current "line."

17.5.2.4 Octet Code 010

The octet code 010 shall represent a line feed (LF). This shall cause the cursor to be advanced to the next line but shall not change cursor position within the line. Typically this code is used in conjunction with CR.

17.5.2.5 Octet Code 009

The octet code 009 shall represent a horizontal advance to the next tab stop (TAB). This shall cause the cursor to be advanced to the next tab position on the current line. This function shall only change the cursor position, without overwriting or altering any characters that have previously been displayed on the same line. Tab stops shall exist at every eight character positions, as shown in Figure 17-5.

17.5.3 Page Size Assumptions

There shall be only two assumptions about page size within the Default-terminal context. First, pages are assumed to have 80 columns.

```
        1         2         3         4         5
12345678901234567890123456789012345678901234567890 ...
        T         T         T         T         T   ...
```

Figure 17-5. VT tab positions.

Each SYMBOL character received is assumed to occupy one column. NUL characters have no effect on column position. CONTROL characters have different effects, as described under 17.5.2. Second, each page is assumed to be unconstrained in length. This is equivalent to a printer with continuous form paper.

18 ERROR, REJECT, and ABORT CODES

All errors associated with the BACnet protocol are enumerated according to a category called "Error Class." Within each Error Class, the errors are further enumerated individually by "Error Code." It is thus possible for an application to take remedial action based upon two levels of granularity.

18.1 Error Class - DEVICE

This Error Class pertains to circumstances that affect the functioning of an entire BACnet device. The presence of one of these errors generally indicates that the entire service request has failed.

18.1.1 DEVICE_BUSY - A service request has been temporarily declined because the addressed BACnet device expects to be involved in higher priority processing for a time in excess of the usual request/confirm timeout period.

18.1.2 CONFIGURATION_IN_PROGRESS - A service request has been temporarily declined because the addressed BACnet device is in the process of being configured, either by means local to the device or by means of other protocol services.

18.1.3 OPERATIONAL_PROBLEM - A service request has been declined because the addressed BACnet device has detected an operational problem that prevents it from carrying out the requested service.

18.1.4 OTHER - This error code is returned for a reason other than any of those previously enumerated for this Error Class.

18.2 Error Class - OBJECT

This Error Class pertains to problems related to identifying, accessing, and manipulating BACnet objects, whether BACnet-defined or not. Since these errors generally apply to individual object characteristics, they do not necessarily signal that an entire service request has failed.

18.2.1 DYNAMIC_CREATION_NOT_SUPPORTED - An attempt has been made to create an object using an object type that cannot be created dynamically.

18.2.2 NO_OBJECTS_OF_SPECIFIED_TYPE - A search of the addressed BACnet device's object database has failed to find any objects of the object type specified in the service request.

18.2.3 OBJECT_DELETION_NOT_PERMITTED - An attempt has been made to delete an object that cannot be deleted or is currently protected from deletion.

18.2.4 OBJECT_IDENTIFIER_ALREADY_EXISTS - An attempt has been made to create a new object using an object identifier already in use.

18.2.5 READ_ACCESS_DENIED - An attempt has been made to read the properties of an object defined as inaccessible through the BACnet protocol read services.

18.2.6 UNKNOWN_OBJECT - An Object_Identifier has been specified for an object that does not exist in the object database of the addressed BACnet device.

18.2.7 UNSUPPORTED_OBJECT_TYPE - An object type has been specified in a service parameter that is unknown or unsupported in the addressed BACnet device.

18.2.8 OTHER - This error code is returned for a reason other than any of those previously enumerated for this Error Class.

18.3 Error Class - PROPERTY

This Error Class pertains to problems related to identifying, accessing, and manipulating the properties of BACnet objects, whether BACnet-defined or not. Since these errors generally apply to individual property characteristics, they do not necessarily signal that an entire service request has failed.

18.3.1 CHARACTER_SET_NOT_SUPPORTED - A character string value was encountered that is not a supported character set.

18.3.2 DATATYPE_NOT_SUPPORTED - The data is of, or contains, a datatype not supported by this instance of this property.

18.3.3 INCONSISTENT_SELECTION_CRITERION - A property has been referenced with a datatype inconsistent with the 'Comparison Value' specified in an 'Object Selection Criteria' service parameter. This error would arise, for example, if an analog property were compared against a Boolean constant, or vice-versa.

18.3.4 INVALID_ARRAY_INDEX - An attempt was made to access an array property using an array index that is outside the range permitted for this array.

18.3.5 INVALID_DATATYPE - The datatype of a property value specified in a service parameter does not match the datatype of the property referenced by the specified Property_Identifier.

18.3.6 NOT_COV_PROPERTY - The property is not conveyed by COV notification.

18.3.7 OPTIONAL_FUNCTIONALITY_NOT_SUPPORTED - An attempt has been made to write a value to a property that would require the device to exhibit non-supported optional functionality.

18.3.8 PROPERTY_IS_NOT_AN_ARRAY - An attempt has been made to access a property as an array and that property does not have an array datatype.

18.3.9 READ_ACCESS_DENIED - An attempt has been made to read a property defined as inaccessible through the BACnet protocol read services.

18.3.10 UNKNOWN_PROPERTY - A Property_Identifier has been specified in a service parameter that is unknown or unsupported in the addressed BACnet device for objects of the referenced object type.

18.3.11 VALUE_OUT_OF_RANGE - An attempt has been made to write to a property with a value that is outside the range of values defined for the property.

18.3.12 WRITE_ACCESS_DENIED - An attempt has been made to write to a property defined as inaccessible through the BACnet protocol write services.

18.3.13 OTHER - This error code is returned for a reason other than any of those previously enumerated for this Error Class.

18.4 Error Class - RESOURCES

This Error Class pertains to problems related to the resources of a BACnet device that affect its capacity to carry out protocol service requests.

18.4.1 NO_SPACE_FOR_OBJECT - An attempt to create an object has failed because not enough dynamic memory space exists in the addressed BACnet device.

18.4.2 NO_SPACE_TO_ADD_LIST_ELEMENT - An attempt to add an element to a list has failed because not enough dynamic memory space exists in the addressed BACnet device.

18.4.3 NO_SPACE_TO_WRITE_PROPERTY - An attempt to write a property has failed because not enough dynamic memory space exists in the addressed BACnet device.

18.4.4 OTHER - This error code is returned for a reason other than any of those previously enumerated for this Error Class.

18.5 Error Class - SECURITY

This Error Class pertains to problems related to the execution of security services. Without exception, these errors signal the inability of the responding BACnet-user to carry out the desired service in its entirety and are thus "fatal."

18.5.1 AUTHENTICATION_FAILED - The message being authenticated was not generated by the service provider.

18.5.2 CHARACTER_SET_NOT_SUPPORTED - A character string value was encountered that is not a supported character set.

18.5.3 INCOMPATIBLE_SECURITY_LEVELS - The two clients do not have the same security authorization.

18.5.4 INVALID_OPERATOR_NAME - The 'Operator Name' is not associated with any known operators.

18.5.5 KEY_GENERATION_ERROR - The key server was unable to generate a Session Key (SK).

18.5.6 PASSWORD_FAILURE - The 'Operator Name' and 'Operator Password' did not associate correctly.

18.5.7 SECURITY_NOT_SUPPORTED - The remote client does not support any BACnet security mechanisms.

18.5.8 TIMEOUT - The APDU with the expected Invoke ID was not received before the waiting time expired.

18.5.9 OTHER - This error code is returned for a reason other than any of those previously enumerated for this Error Class.

18.6 Error Class - SERVICES

This Error Class pertains to problems related to the execution of protocol service requests, whether BACnet-defined or not. Without exception, these errors signal the inability of the responding BACnet-user to carry out the desired service in its entirety and are thus "fatal."

18.6.1 CHARACTER_SET_NOT_SUPPORTED - A character string value was encountered that is not a supported character set.

18.6.2 COV_SUBSCRIPTION_FAILED - COV Subscription failed for some reason.

18.6.3 DUPLICATE_NAME - An attempt has been made to write to an Object_Name property with a value that is already in use in a different Object_Name property within the device.

18.6.4 DUPLICATE_OBJECT_ID – An attempt has been made to write to an Object_Identifier property with a value that is already in use in a different Object_Identifier within the same device.

18.6.5 FILE_ACCESS_DENIED - Generated in response to an AtomicReadFile or AtomicWriteFile service request for access to a file that is currently locked or otherwise not accessible.

18.6.6 INCONSISTENT_PARAMETERS - A conflict exists because two or more of the parameters specified in the service request are mutually exclusive.

18.6.7 INVALID_CONFIGURATION_DATA - The configuration data provided was invalid or corrupt.

18.6.8 INVALID_FILE_ACCESS_METHOD - Generated in response to an AtomicReadFile or AtomicWriteFile request that specifies a 'File Access Method' that is not valid for the specified file.

18.6.9 INVALID_FILE_START_POSITION - Generated in response to an AtomicReadFile or AtomicWriteFile request that specifies an invalid 'File Start Position' or invalid 'File Start Record' parameter.

18.6.10 INVALID_PARAMETER_DATATYPE - The datatype of a value specified for a service parameter is not appropriate to the parameter.

18.6.11 INVALID_TIME_STAMP - The 'Time Stamp' parameter conveyed by an AcknowledgeAlarm service request does not match the time of the most recent occurrence of the event being acknowledged.

18.6.12 MISSING_REQUIRED_PARAMETER - A parameter required for the execution of a service request has not been supplied.

18.6.13 OPTIONAL_FUNCTIONALITY_NOT_SUPPORTED - The parameters of a service are such that the device would be required to exhibit non-supported optional functionality.

18.6.14 PROPERTY_IS_NOT_A_LIST - An attempt has been made to access a property via either the AddListElement service or the RemoveListElement service and that property does not have a list datatype.

18.6.15 PROPERTY_IS_NOT_AN_ARRAY - An attempt has been made to access a property as an array and that property does not have an array datatype.

18.6.16 SERVICE_REQUEST_DENIED - A request has been made to execute a service for which the requesting BACnet device does not have the appropriate authorization.

18.6.17 OTHER - This error code is returned for a reason other than any of those previously enumerated for this Error Class.

18.7 Error Class - VT

This Error Class pertains to problems related to the execution of Virtual Terminal services.

18.7.1 UNKNOWN_VT_CLASS - This error indicates that the 'VT-Class' specified in a VT-Open request was not recognized by the target device.

18.7.2 UNKNOWN_VT_SESSION - This error indicates that the 'VT-Session ID' specified in a VT-Data or VT-Close request was not recognized by the target device.

18.7.3 NO_VT_SESSIONS_AVAILABLE - This error indicates that the target device could not fulfill a VT-Open request because of resource limitations.

18.7.4 VT_SESSION_ALREADY_CLOSED - This error indicates that an attempt has been made to close a VT-session that has been previously terminated.

18.7.5 VT_SESSION_TERMINATION_FAILURE - This error indicates that one of the 'VT-Sessions' specified in a VT-Close request could not be released for some implementation-dependent reason.

18.7.6 OTHER - This error code is returned for a reason other than any of those previously enumerated for this Error Class

18.8 Reject Reason

Only confirmed request PDUs can be rejected. The possible reasons for rejecting the PDU are enumerated in this subclause.

18.8.1 BUFFER_OVERFLOW - An input buffer capacity has been exceeded.

18.8.2 INCONSISTENT_PARAMETERS - Generated in response to a confirmed request APDU that omits a conditional service argument that should be present or contains a conditional service argument that should not be present. This condition could also elicit a Reject PDU with a Reject Reason of INVALID_TAG.

18.8.3 INVALID_PARAMETER_DATA_TYPE - Generated in response to a confirmed request APDU in which the encoding of one or more of the service parameters does not follow the correct type specification. This condition could also elicit a Reject PDU with a Reject Reason of INVALID_TAG.

18.8.4 INVALID_TAG - While parsing a message, an invalid tag was encountered. Since an invalid tag could confuse the parsing logic, any of the following Reject Reasons may also be generated in response to a confirmed request containing an invalid tag: INCONSISTENT_PARAMETERS, INVALID_PARAMETER_DATA_TYPE, MISSING_REQUIRED_PARAMETER, and TOO_MANY_ARGUMENTS.

18.8.5 MISSING_REQUIRED_PARAMETER - Generated in response to a confirmed request APDU that is missing at least one mandatory service argument. This condition could also elicit a Reject PDU with a Reject Reason of INVALID_TAG.

18.8.6 PARAMETER_OUT_OF_RANGE - Generated in response to a confirmed request APDU that conveys a parameter whose value is outside the range defined for this service.

18.8.7 TOO_MANY_ARGUMENTS - Generated in response to a confirmed request APDU in which the total number of service arguments is greater than specified for the service. This condition could also elicit a Reject PDU with a Reject Reason of INVALID_TAG.

18.8.8 UNDEFINED_ENUMERATION - Generated in response to a confirmed request APDU in which one or more of the service parameters is decoded as an enumeration that is not defined by the type specification of this parameter.

18.8.9 UNRECOGNIZED_SERVICE - Generated in response to a confirmed request APDU in which the Service Choice field specifies an unknown or unsupported service.

18.8.10 OTHER - Generated in response to a confirmed request APDU that contains a syntax error for which an error code has not been explicitly defined.

18.9 Abort Reason

18.9.1 BUFFER_OVERFLOW - An input buffer capacity has been exceeded.

18.9.2 INVALID_APDU_IN_THIS_STATE - Generated in response to an APDU that is not expected in the present state of the Transaction State Machine.

18.9.3 PREEMPTED_BY_HIGHER_PRIORITY_TASK - The transaction shall be aborted to permit higher priority processing.

18.9.4 SEGMENTATION_NOT_SUPPORTED - Generated in response to an APDU that has its segmentation bit set to TRUE when the receiving device does not support segmentation. It is also generated when a BACnet-ComplexACK-PDU is large enough to require segmentation but it cannot be transmitted because either the transmitting device or the receiving device does not support segmentation.

18.9.5 OTHER - This abort reason is returned for a reason other than any of those previously enumerated.

19 BACnet PROCEDURES

This clause defines several procedures that are commonly required in building automation and control systems. Each procedure makes use of BACnet capabilities defined elsewhere in this standard.

19.1 Backup and Restore

This clause describes the procedures to be used to backup and restore the configuration of BACnet devices.

19.1.1 The Backup and Restore Procedures

In BACnet building control systems, many devices will have configuration data that is set up by a vendor's proprietary configuration tool. This setup may consist of network visible BACnet objects and/or non-network visible settings. This section outlines the standard method that BACnet devices will employ if an interoperable device backup and restore feature is to be provided.

The backup and restore procedures use File objects to hold and transfer the configuration data. The content and format of the configuration files is a local matter. The choice of whether to use stream-based files or record-based files is a local matter. The services required to support the backup and restore procedures are ReinitializeDevice, ReadProperty, WriteProperty, AtomicWriteFile, AtomicReadFile, and optionally CreateObject.

19.1.2 Backup

For the purposes of this discussion, the device performing the backup procedure will be referred to as device A, and the device being backed up will be device B.

19.1.2.1 Initiation of the Backup Procedure

Device A sends a ReinitializeDevice(STARTBACKUP, <password>) message to device B. Device A will await a response from device B before continuing with the backup procedure.

19.1.2.2 Preparation for Backup

Upon receipt of the ReinitializeDevice(STARTBACKUP, <password>) message, if device B is able to perform a backup procedure, device B will prepare for the backup procedure and respond with a 'Result(+)' to the ReinitializeDevice service request.

If device B is unable to perform a backup procedure or is already performing a backup procedure, then it will respond to the ReinitializeDevice service request with a 'Result(-)' response. Assuming device B supports the backup procedure and the request was properly formulated, the valid Error Class:Error Codes that can be returned are :

DEVICE:CONFIGURATION_IN_PROGRESS - if device B is already processing a backup or a restore request.

SERVICES:SERVICE_REQUEST_DENIED – if the password that was provided was incorrect or if a password is required and one was not provided.

After device B responds to the ReinitializeDevice request with a 'Result(+)', the configuration File objects must exist in the device. It is a local matter as to whether device B will respond to other requests while it is in backup mode. The exception to this is that device B must accept and fulfill read requests by device A that consist of accesses to device B's Device object and/or its configuration File objects. Any services that are rejected due to an in-progress backup procedure will be rejected with an error class of DEVICE and error code of DEVICE_BUSY.

It is a local matter as to whether device B will continue to perform control actions while it is in backup mode. If device B changes its operational behavior during a backup procedure, then the System_Status property of the Device object shall be set to BACKUP_IN_PROGRESS.

19.1.2.3 Loading the Backup Parameters

Upon receipt of a 'Result(+)' response from device B to the ReinitializeDevice(STARTBACKUP, <password>) message, device A will read the Configuration_Files property of the Device object. This property will be used to determine the files to read and in what order the files will be read. The value of the Configuration_Files property is not guaranteed to contain a complete or correct set of configuration File object references before the backup request is accepted by device B.

19.1.2.4 Backing Up the Configuration Files

Once device A has determined the files that make up the device configuration image, device A will determine the type of each file and will use the AtomicReadFile service to read each configuration file from device B. Each file will be read as a stream of bytes, or as a sequence of records depending on the File_Access_Method property of the File object. Stream access files will be read in byte order and record access files will be read in record order. The files will be read in the same order as they appear in the Configuration_Files property.

It is a local matter as to what device A does with the configuration files, although the intent of the service is to allow an operator to archive the setup of device B such that device B may be restored at a later date should its configuration become corrupt.

It is left up to the implementer of device A as to whether the files read from device B will be made available for examination by tools developed by the implementer of device B. It is recommended that record access files be stored on device A as a sequence of BACnet OCTET STRINGS.

19.1.2.5 Ending the Backup Procedure

When the all of the configuration files have been read, device A sends a ReinitializeDevice(ENDBACKUP, <password>) message to device B. Device B will perform whatever actions are required to complete the backup in order to place the device back into the state it was in before the backup procedure or into any other state as defined by the vendor. Device B must not remain in the BACKUP_IN_PROGRESS mode after the backup procedure has ended.

If device A needs to abort the backup for any reason (i.e., the user aborts the procedure, device B fails to allow reads from a configuration file, or device A detects any other condition that inhibits the backup procedure), device A shall attempt to send ReinitializeDevice(ENDBACKUP, <password>) to device B. Upon receipt of this message, device B shall end the backup procedure. If the backup procedure is aborted, device A should not assume that the configuration files are still valid and continue to read them.

The receipt of the ReinitializeDevice(ENDBACKUP, <password>) message shall cause device B to exit backup mode.

If device B does not receive any messages related to the backup procedure from device A for the number of seconds specified in the Backup_Failure_Timeout property of its Device object, device B should assume that the backup procedure has been aborted, and device B should exit backup mode. A message related to the backup procedure is defined to be any ReadProperty, WriteProperty, CreateObject, or AtomicReadFile request that directly accesses a configuration File object.

19.1.3 Restore

For the purposes of this discussion, the device performing the restore procedure will be referred to as device A, and the device being restored will be device B.

19.1.3.1 Initiation of the Restore Procedure

Device A sends a ReinitializeDevice(STARTRESTORE, <password>) message to device B. Device A will await a response from device B before continuing the restore procedure.

19.1.3.2 Preparation for Restore

Upon receipt of the restore request, if device B is able to perform a restore procedure, device B will prepare for the restore procedure and will respond with a 'Result(+)' to the ReinitializeDevice service request.

If device B is unable to perform a restore procedure, then it will respond to the ReinitializeDevice service request with a 'Result(-)' response. Assuming device B supports the restore procedure and the request was properly formulated, the valid Error Class:Error Codes that can be returned are:

DEVICE:CONFIGURATION_IN_PROGRESS – if device B is already processing a backup or a restore request.

SERVICES:SERVICE_REQUEST_DENIED – if the password that was provided was incorrect or if a password is required and one was not provided.

After device B responds to the ReinitializeDevice request with a 'Result(+)', the configuration File objects must exist in the device, or device B must be able to accept CreateObject requests from device A to create the configuration File objects. It is a local matter as to whether device B will respond to other requests while it is in restore mode. The exception to this is that device B must accept and fulfill read and write requests by device A that consist of accesses to device B's Device object and/or its configuration File objects. Any services that are rejected due to an in-progress backup procedure will be rejected with an error class of DEVICE and error code of CONFIGURATION_IN_PROGRESS.

Device B must be prepared to answer device A's requests for information from device B's Device object. If device B cannot service requests from devices other than device A, then device B shall reject those services with an error class of DEVICE and an error code of CONFIGURATION_IN_PROGRESS.

It is a local matter as to whether device B will continue to perform control actions while it is in restore mode. If device B changes its operational behavior during a restore procedure, then the System_Status property of the Device object shall be set to DOWNLOAD_IN_PROGRESS.

19.1.3.3 Restoring the Configuration Files

Device A will use the AtomicWriteFile service to write each configuration file to device B. If any of the files do not exist in device B, then device A will attempt to create the files using the CreateObject service. Any files that already exist in the device, and differ in size from the image being written to them, will be truncated by writing 0 to the File_Size property of the File object before the contents are written to the file.

The configuration files will be written as a stream of bytes, or as a sequence of records, depending on the value of the File_Access_Method property of the File object. Note that there is no standard file format for record-based files, whereas any file can be written as a stream of bytes.

Each configuration file written to the device should be a valid configuration file obtained from the vendor, from a vendor's configuration tool, or from a previous backup procedure. The files will be written to the device in the same order as they were retrieved during the backup procedure, or as specified by the vendor if the files were obtained from another source.

Device B is allowed to reject any write operation to the configuration file if it has determined that the content of the write is invalid (internal CRC error, Invalid type code, etc). If this is the case, device B will respond with an error class of SERVICES and an error code of INVALID_CONFIGURATION_DATA. It is a local matter as to whether device A will retry the request and how many times device A will retry, but device A should abort the restore procedure if device B continues to return an error.

19.1.3.4 Ending the Restore Procedure

When device A has completely written all of the configuration files to device B, device A will send ReinitializeDevice(ENDRESTORE, <password>). Device B will perform whatever actions are required to complete the restore procedure, which should include a validation of the restored configuration. If the validation fails, it is a local matter as to what device B will do beyond changing its System_State property to something other than DOWNLOAD_IN_PROGRESS.

If device A needs to abort the restore for any reason (i.e., the user aborts the procedure, device B fails to allow writes to a configuration file, or device A detects any other condition that inhibits the restore procedure), device A shall attempt to send ReinitializeDevice(ABORTRESTORE, <password>) to device B. Upon receipt of this message, device B shall abort the restore procedure.

If device B does not receive any messages related to the restore procedure from device A for the number of seconds specified in the Backup_Failure_Timeout property of its Device object, device B should assume that the restore procedure has been aborted, and device B should exit restore mode. A message related to the restore procedure is defined to be any ReadProperty, WriteProperty, CreateObject, or AtomicWriteFile request that directly accesses a configuration File object.

Once the restore procedure has ended, whether it was successful or not, device B must change its System_Status property to something other than DOWNLOAD_IN_PROGRESS.

If the restore is successful, no other actions by device A shall be required, and device B will update the Last_Restore_Time property in its Device object.

If the restore failed or was aborted and device B is unable to recover its old configuration, or cannot establish a default configuration, device B shall set its System_State to DOWNLOAD_REQUIRED. Every attempt shall be made to leave device B in a state that will accept additional restore procedures.

19.2 Command Prioritization

In building control systems, an object may be manipulated by a number of entities. For example, the present value of a Binary Output object may be set by several applications, such as demand metering, optimum start/stop, etc. Each such application program has a well-defined function it needs to perform. When the actions of two or more application programs conflict with regard to the value of a property, there is a need to arbitrate between them. The objective of the arbitration process is to ensure the desired behavior of an object that is manipulated by several program (or non-program) entities. For example, a start/stop program may specify that a particular Binary Output should be ON, while demand metering may specify that the same Binary Output should be OFF. In this case, the OFF should take precedence. An operator may be able to override the demand metering program and force the Binary Output ON, in which case the ON should take precedence.

In BACnet, this arbitration is provided by a prioritization scheme that assigns varying levels of priorities to commanding entities on a system-wide basis. Each object that contains a commandable property is responsible for acting upon prioritized commands in the order of their priorities. While there is a trade-off between the complexity and the robustness of any such mechanism, the scheme described here is intended to be effective but applicable to even simple BACnet devices.

The following property types are involved in the prioritization mechanism:

(a) Commandable Property: Each object that supports command prioritization has one or more distinguished properties that are referred to as "Commandable Properties." The value of these properties is controlled by the command prioritization mechanism.

(b) Priority_Array: This property is a read only array that contains prioritized commands or NULLs in the order of decreasing priority. The highest priority (lowest array index) with a non-NULL value is the active command.

(c) Relinquish_Default: This property shall be of the same datatype (and engineering units) as the Commandable Property. When all entries in the Priority_Array are NULL, the value of the Commandable Property shall have the value specified by the Relinquish_Default property.

Although the Command object is used to write a set of values to a group of object properties, command prioritization is not involved unless the properties are commandable.

19.2.1 Prioritization Mechanism

For BACnet objects, commands are prioritized based upon a fixed number of priorities that are assigned to command-issuing entities. A prioritized command (one that is directed at a commandable property of an object) is performed via a WriteProperty service request or a WritePropertyMultiple service request. The request primitive includes a conditional 'Priority' parameter that ranges from 1 to 16. Each commandable property of an object has an associated priority table that is represented by the Priority_Array property. The Priority_Array consists of an array of commanded values in order of decreasing priority. The first value in the array corresponds to priority 1 (highest), the second value corresponds to priority 2, and so on, to the sixteenth value that corresponds to priority 16 (lowest).

An entry in the Priority_Array may have a commanded value or a NULL. A NULL value indicates that there is no existing command at that priority. An object continuously monitors all entries within the priority table in order to locate the entry with the highest priority non-NULL value and sets the commandable property to this value.

A commanding entity (application program, operator, etc.) may issue a command to write to the commandable property of an object, or it may relinquish a command issued earlier. Relinquishing of a command is performed by a write operation similar to the command itself, except that the commandable property value is NULL. Relinquishing a command places a NULL value in the Priority_Array corresponding to the appropriate priority. This prioritization approach shall be applied to local actions that change the value of commandable properties as well as to write operations via BACnet services.

If an attempt is made to write to a commandable property without explicitly specifying the priority, a default priority of 16 (the lowest priority) shall be assumed. If an attempt is made to write to a property that is not commandable with a specified priority, the priority shall be ignored. The Priority_Array property is read only. Its values are changed indirectly by writing to the commandable property itself.

19.2.1.1 Commandable Properties

The prioritization scheme is applied to certain properties of objects. The standard commandable properties and objects are as follows:

OBJECT	COMMANDABLE PROPERTY
Analog Output	Present_Value
Binary Output	Present_Value
Multi-state Output	Present_Value
Multi-state Value	Present_Value
Analog Value	Present_Value
Binary Value	Present_Value

The designated properties of the Analog Output, Binary Output and Multi-state Output objects are commandable (prioritized) by definition. The designated properties of the Analog Value, Binary Value and Multi-state Value objects may optionally be commandable. Individual vendors, however, may decide to apply prioritization to any of the vendor-specified properties. These additional commandable properties shall have associated Priority_Array and Relinquish_Default properties with appropriate names. See 23.3.

19.2.1.2 Prioritized Commands

Prioritized commands, i.e., commands directed at commandable properties, are either WriteProperty service requests or WritePropertyMultiple service requests. In either case, the request primitive shall contain (among others) the following parameters:

Property Identifier:	Commandable_Property
Property Value:	Desired Value
Priority:	Priority

The end result of a successful write operation is to place a desired value in the priority table at the appropriate priority. If another value was already present at that priority, it shall be overwritten with the new value, without any regard to the identity of the previous commanding entity.

19.2.1.3 Relinquish Commands

When a commanding entity no longer desires to control a commandable property, it issues a relinquish command. A relinquish command is also either a WriteProperty service request or a WritePropertyMultiple service request. In either case, the request primitive shall contain (among others) the following parameters:

Property Identifier:	Commandable_Property
Property Value:	NULL
Priority:	Priority

The placement of NULL in the value parameter indicates the absence of any command at that priority. When all elements of the priority table array contain NULL, the commandable property shall assume the value defined in the Relinquish_Default property of the object.

It is possible for an application entity to relinquish at a priority other than its own, resulting in unpredictable behavior. If more than one application is assigned the same priority, it is possible for one application entity to write-over (or relinquish) the commands from the other application entity, resulting in unpredictable operation. To minimize this possibility, it is very important not to allow more than one commanding entity to assume the same priority level within the system.

19.2.1.4 Command Source ID

There is no provision for maintaining command source identification as part of the priority table. Any implementation of command source identification is vendor-specific in nature.

19.2.1.5 Command Overwrite

Whenever a command is issued to a commandable property, the value is placed in the Priority_Array at the appropriate priority position, without any regard to the current value residing there. The new command overwrites the existing command. No notification of such overwrite is made to the original commanding entity.

19.2.2 Application Priority Assignments

Commanding entities are assigned one of the 16 possible priority levels. The assignment of most priorities is site dependent and represents the objectives of the site management. Table 19-1 contains the standard priorities. Other applications that need prioritization include Temperature Override, Demand Limiting, Optimum Start/Stop, Duty Cycling, and Scheduling. The relative priorities of these applications may vary from site to site and are not standardized. For interoperability at any particular site, the only requirement is that all devices implement the same priority scheme. The positions marked Available are open for assignment to DDC programs, EMS programs, etc. The interpretation of what conditions constitute Manual-Life Safety or Automatic-Life Safety decisions is a local matter.

Table 19-1. Standard Command Priorities

Priority Level	Application	Priority Level	Application
1	Manual-Life Safety	9	Available
2	Automatic-Life Safety	10	Available
3	Available	11	Available
4	Available	12	Available
5	Critical Equipment Control	13	Available
6	Minimum On/Off	14	Available
7	Available	15	Available
8	Manual Operator	16	Available

19.2.3 Minimum_On_Time and Minimum_Off_Time

If the commandable property is the Present_Value property of a Binary Output object or a Binary Value object and that object possesses the optional Minimum_On_Time and Minimum_Off_Time properties, then minimum on and minimum off times shall behave according to the algorithm described in this subclause.

Command priority 6 is reserved for use by this algorithm and may not be used for other purposes in any object.

(a) If the Present_Value is ACTIVE and the time since the last change of state of the Present_Value is less than the Minimum_On_Time, then element 6 of the Priority_Array shall contain a value of ACTIVE.

(b) If the Present_Value is INACTIVE and the time since the last change of state of the Present_Value is less than the Minimum_Off_Time, then element 6 of the Priority_Array shall contain a value of INACTIVE.

(c) If neither (a) nor (b) is true, then element 6 of the Priority_Array shall contain a value of NULL.

These rules imply actions to be taken when the Present_Value is written and actions to be taken based on elapsed time. The means by which these actions are implemented is a local matter, so long as the behavior described in this subclause is achieved.

When a write to a commandable property occurs at any priority, the specified value or relinquish (NULL) is always written to the appropriate slot in the priority table, regardless of any minimum on or off times.

The Priority_Array is then examined by the local priority maintenance entity to determine the highest priority that contains a non-NULL value. If this value differs from the Present_Value immediately before the write, then a change of state has occured. If such a change of state occurs, the new value is also written to priority 6 in the Priority_Array and the time of the change is noted. The means by which the timing is performed is a local matter.

When the minimum on or off time signified by a non-NULL value in priority 6 has elapsed, the local minimum time maintenance entity shall write a NULL to priority 6 and re-examine the Priority_Array to determine the new Present_Value. If this value indicates a change of state, then the appropriate actions shall be taken as described above.

The effect of a non-NULL value in priority 6 is that writes at any lower priority (larger priority number) cannot cause a change of state. Thus, minimum on or off time protection may be achieved relative to these priorities.

Writes to any priority higher than 6 (smaller priority number) may cause changes of state regardless of Minimum_On_Time or Minimum_Off_Time. Thus, these priorities should be used only for critical or emergency use. Note, however, that changes of state caused by a write to these high priorities will also cause writes to priority 6 as described above. Thus, if a NULL is subsequently written to the high priority while minimum time is in effect, that time shall be observed before any change of state is made as a result of a value at a lower priority.

For additional discussion of minimum on and off time processing see Annex I.

19.2.4 Prioritization for Command Objects

A Command object is capable of issuing commands just as any other command-issuing entity. A Command object may be related to an application with any priority. The Action property of the Command object contains all of the parameters necessary for writing to commandable properties. See 12.10.8.

19.2.5 Prioritization for Loop Objects

Loop objects may need to interact with objects that have a commandable property, even though, in general, they will not use BACnet services to do so. Each Loop object has a Priority_For_Writing property that designates the appropriate priority of this control loop with respect to the commandable property. See 12.17.28.

19.2.6 Prioritization for Schedule Objects

Schedule objects may need to interact with objects that have a commandable property, even though, in general, they will not use BACnet services to do so. Each Schedule object has a Priority_For_Writing property that designates the appropriate priority of this schedule with respect to the commandable property. See 12.24.11.

20 ENCODING BACnet PROTOCOL DATA UNITS

Application Layer Protocol Data Units (APDUs) are used in BACnet to convey the information contained in the application service primitives and associated parameters.

ISO Standard 8824, Specification of Abstract Syntax Notation One (ASN.(1), has been selected as the method for representing the data content of BACnet services. Clause 21 contains an ASN.1 definition for each service defined by this standard. ASN.1 provides an abstract syntax. However, the exact bit-by-bit layout of an APDU may have several forms, depending on the encoding rules that are selected.

Within the Open Systems Interconnection model, the encoding rules to be used are chosen by the presentation layer through a process of negotiation. This negotiation is used by cooperating systems to determine not only the basic encoding rules, of which ISO 8825, Specification of Basic Encoding Rules for ASN.1, is an example, but also whether or not the APDU is to be subjected to other manipulations such as data compression, encryption, or character code conversion.

Because BACnet's collapsed OSI architecture does not incorporate any presentation layer functionality, APDU encoding must be defined and agreed to by communicating devices in advance. BACnet's encoding rules have been designed to take into account the requirements of building automation and control systems for simplicity and compactness. As a result, they differ, in some respects, from ISO 8825 while still permitting the encoding of BACnet APDUs that have been represented using ASN.1. This means that BACnet services and procedures could be used in their entirety in a future OSI-compliant network by adding the presentation layer capability to negotiate either the encoding rules contained in this standard or any other encoding rules that might later be available.

The encoding of ASN.1 specified in ISO 8825 is intended to apply uniformly to all data elements in a PDU. Each data element is represented by three components: (1) identifier octets, (2) length octets, and (3) contents octets. The explicit identification of each data element allows parsers to be developed that can decode any PDU without prior knowledge of its format or semantic content. The alternative is to implicitly identify each data element, generally by mutual agreement as to its data format and location within the PDU. The former approach tends to result in greater generality at the expense of greater overhead; the latter approach tends to reduce overhead while limiting future extensibility.

The approach taken in BACnet is a compromise. The fixed portion of each APDU containing protocol control information is encoded implicitly and is described in 20.1. The variable portion of each APDU containing service-specific information is encoded explicitly and is described in 20.2. The resulting scheme significantly reduces overhead while preserving the possibility of easily adding new services in the future.

20.1 Encoding the Fixed Part of BACnet APDUs

BACnet APDUs consist of protocol control information and, possibly, user data.

"Protocol control information" (PCI) comprises data required for the operation of the application layer protocol, including the type of APDU, information to match service requests and service responses, and information to carry out the reassembly of segmented messages. This information is contained in the "header," or fixed part, of the APDU.

"User data" comprises information specific to individual service requests or responses. This portion of the APDU will be referred to as the 'variable part' of the APDU.

Because every APDU contains PCI fields, BACnet encodes the PCI without the use of tags or length information even though the ASN.1 might indicate the presence of tags in the syntactical descriptions of the APDUs. Tags are used to encode the variable-content user data as specified in 20.2. This selective use of tags results in a considerable reduction in overhead.

The remainder of this subclause lays out the format of each APDU type.

20.1.1 Encoding the BACnetPDU CHOICE Tag

All BACnet messages are defined by an ASN.1 production called the BACnetPDU. See Clause 21. BACnetPDU is a choice of one of eight BACnet APDU types. For all BACnet APDUs, this choice shall be encoded as a four-bit binary number in the bits 4 through 7 of the first octet of the APDU header, with bit 7 being the most significant bit. These bits indicate the value

of the tag (0 - 7), which represents the APDU type choice. This encoding is illustrated in the examples below for each APDU type.

20.1.2 BACnet-Confirmed-Request-PDU

The BACnet-Confirmed-Request-PDU is used to convey the information contained in confirmed service request primitives.

```
BACnet-Confirmed-Request-PDU ::= SEQUENCE {
        pdu-type                        [0] Unsigned (0..15), -- 0 for this PDU type
        segmented-message               [1] BOOLEAN,
        more-follows                    [2] BOOLEAN,
        segmented-response-accepted     [3] BOOLEAN,
        reserved                        [4] Unsigned (0..3), -- must be set to zero
        max-segments-accepted           [5] Unsigned (0..7),  -- as per 20.1.2.4
        max-APDU-length-accepted        [6] Unsigned (0..15), -- as per 20.1.2.5
        invokeID                        [7] Unsigned (0..255),
        sequence-number                 [8] Unsigned (0..255) OPTIONAL, -- only if segmented msg
        proposed-window-size            [9] Unsigned (1..127) OPTIONAL, -- only if segmented msg
        service-choice                  [10] BACnetConfirmedServiceChoice,
        service-request                 [11] BACnet-Confirmed-Service-Request
-- Context specific tags 0..11 are NOT used in header encoding
        }
```

The parameters of the BACnet-Confirmed-Request-PDU have the following meanings.

20.1.2.1 segmented-message

This parameter indicates whether or not the confirmed service request is entirely, or only partially, contained in the present PDU. If the request is present in its entirety, the value of the 'segmented-message' parameter shall be FALSE. If the present PDU contains only a segment of the request, this parameter shall be TRUE.

20.1.2.2 more-follows

This parameter is only meaningful if the 'segmented-message' parameter is TRUE. If 'segmented-message' is TRUE, then the 'more-follows' parameter shall be TRUE for all segments comprising the confirmed service request except for the last and shall be FALSE for the final segment. If 'segmented-message' is FALSE, then 'more-follows' shall be set FALSE by the encoder and shall be ignored by the decoder.

20.1.2.3 segmented-response-accepted

This parameter shall be TRUE if the device issuing the confirmed request will accept a segmented complex acknowledgment as a response. It shall be FALSE otherwise. This parameter is included in the confirmed request so that the responding device may determine how to convey its response.

20.1.2.4 max-segments-accepted

This optional parameter specifies the maximum number of segments that the device will accept. This parameter is included in the confirmed request so that the responding device may determine how to convey its response. The parameter shall be encoded as follows:

B'000'	Unspecified number of segments accepted.
B'001'	2 segments accepted.
B'010'	4 segments accepted.
B'011'	8 segments accepted.
B'100'	16 segments accepted.
B'101'	32 segments accepted.
B'110'	64 segments accepted.
B'111'	Greater than 64 segments accepted.

20.1.2.5 max-APDU-length-accepted

This parameter specifies the maximum size of a single APDU that the issuing device will accept. This parameter is included in the confirmed request so that the responding device may determine how to convey its response. The parameter shall be encoded as follows:

B'0000' Up to MinimumMessageSize (50 octets)
B'0001' Up to 128 octets
B'0010' Up to 206 octets (fits in a LonTalk frame)
B'0011' Up to 480 octets (fits in an ARCNET frame)
B'0100' Up to 1024 octets
B'0101' Up to 1476 octets (fits in an ISO 8802-3 frame)
B'0110' reserved by ASHRAE
B'0111' reserved by ASHRAE
B'1000' reserved by ASHRAE
B'1001' reserved by ASHRAE
B'1010' reserved by ASHRAE
B'1011' reserved by ASHRAE
B'1100' reserved by ASHRAE
B'1101' reserved by ASHRAE
B'1110' reserved by ASHRAE
B'1111' reserved by ASHRAE

20.1.2.6 invokeID

This parameter shall be an integer in the range 0 - 255 assigned by the service requester. It shall be used to associate the response to a confirmed service request with the original request. In the absence of any error, the 'invokeID' shall be returned by the service provider in a BACnet-SimpleACK-PDU or a BACnet-ComplexACK-PDU. In the event of an error condition, the 'invokeID' shall be returned by the service provider in a BACnet-Error-PDU, BACnet-Reject-PDU, or BACnet-Abort-PDU as appropriate.

The 'invokeID' shall be generated by the device issuing the service request. It shall be unique for all outstanding confirmed request APDUs generated by the device. The same 'invokeID' shall be used for all segments of a segmented service request. Once an 'invokeID' has been assigned to an APDU, it shall be maintained within the device until either a response APDU is received with the same 'invokeID' or a no response timer expires (see 5.3). In either case, the 'invokeID' value shall then be released for reassignment. The algorithm used to pick a value out of the set of unused values is a local matter. The storage mechanism for maintaining the used 'invokeID' values within the requesting and responding devices is also a local matter. The requesting device may use a single 'invokeID' space for all its confirmed APDUs or multiple 'invokeID' spaces (one per destination device address) as desired. Since the 'invokeID' values are only source-device-unique, the responding device shall maintain the 'invokeID' as well as the requesting device address until a response has been sent. The responding device may discard the 'invokeID' information after a response has been sent.

20.1.2.7 sequence-number

This optional parameter is only present if the 'segmented-message' parameter is TRUE. In this case, the 'sequence-number' shall be a sequentially incremented unsigned integer, modulo 256, which identifies each segment of a segmented request. The value of the received 'sequence-number' is used by the responder to acknowledge the receipt of one or more segments of a segmented request. The 'sequence-number' of the first segment of a segmented request shall be zero.

20.1.2.8 proposed-window-size

This optional parameter is only present if the 'segmented-message' parameter is TRUE. In this case, the 'proposed-window-size' parameter shall specify as an unsigned binary integer the maximum number of message segments containing 'invokeID' the sender is able or willing to send before waiting for a segment acknowledgment PDU (see 5.2 and 5.3). The value of the 'proposed-window-size' shall be in the range 1 - 127.

20.1.2.9 service-choice

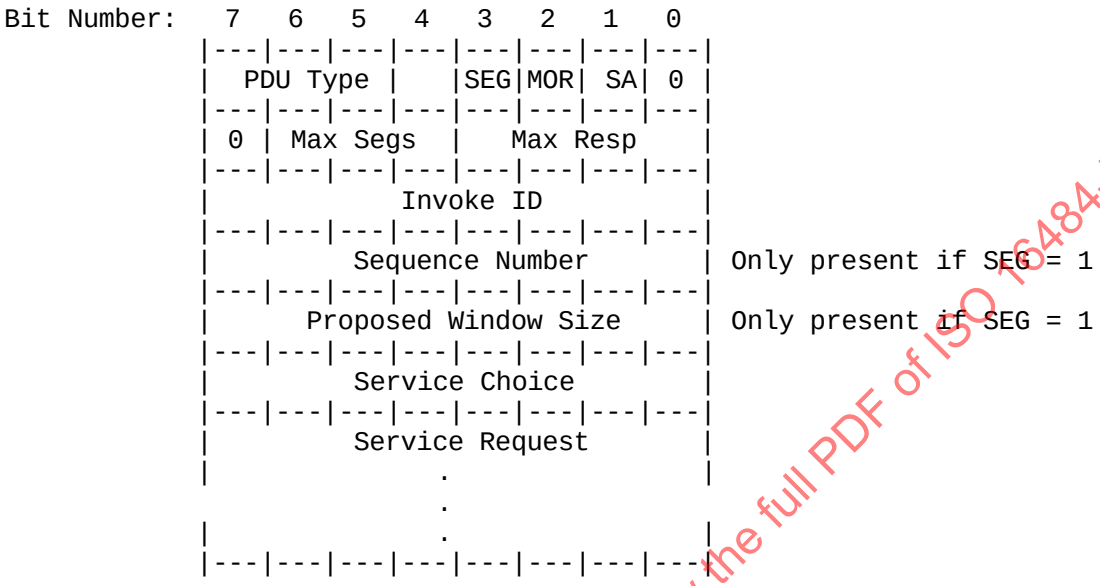
This parameter shall contain the value of the BACnetConfirmedServiceChoice. See Clause 21.

20.1.2.10 service-request

This parameter shall contain the parameters of the specific service that is being requested, encoded according to the rules of 20.2. These parameters are defined in the individual service descriptions in this standard and are represented in Clause 21 in accordance with the rules of ASN.1.

20.1.2.11 Format of the BACnet-Confirmed-Request-PDU

The format of the BACnet-Confirmed-Request-PDU is:



The PDU fields have the following values:

PDU Type =	0 (BACnet-Confirmed-Service-Request-PDU)
SEG =	0 (Unsegmented Request)
	1 (Segmented Request)
MOR =	0 (No More Segments Follow)
	1 (More Segments Follow)
SA =	0 (Segmented Response not accepted)
	1 (Segmented Response accepted)
Max Segs =	(0..7) (Number of response segments accepted per 20.1.2.4)
Max Resp =	(0..15) (Size of Maximum APDU accepted per 20.1.2.5)
Invoke ID =	(0..255)
Sequence Number =	(0..255) Only present if SEG = 1
Proposed Window Size =	(1..127) Only present if SEG = 1
Service Choice =	BACnetConfirmedServiceChoice
Service Request =	Variable Encoding per 20.2.

Bits shown in the diagram as '0' shall be set to zero. These bits are currently unused and are reserved by ASHRAE.

20.1.3 BACnet-Unconfirmed-Request-PDU

The BACnet-Unconfirmed-Request-PDU is used to convey the information contained in unconfirmed service request primitives.

```
BACnet-Unconfirmed-Request-PDU ::= SEQUENCE {
        pdu-type            [0] Unsigned (0..15), -- 1 for this PDU type
        reserved            [1] Unsigned (0..15), -- must be set to zero
        service-choice      [2] BACnetUnconfirmedServiceChoice,
        service-request     [3] BACnet-Unconfirmed-Service-Request
```

```
-- Context specific tags 0..3 are NOT used in header encoding
        }
```

The parameters of the BACnet-Unconfirmed-Request-PDU have the following meanings.

20.1.3.1 service-choice

This parameter shall contain the value of the BACnetUnconfirmedServiceChoice. See Clause 21.

20.1.3.2 service-request

This parameter shall contain the parameters of the specific service that is being requested, encoded according to the rules of 20.2. These parameters are defined in the individual service descriptions in this standard and are represented in Clause 21 in accordance with the rules of ASN.1.

20.1.3.3 Format of the BACnet-Unconfirmed-Request-PDU

The format of the BACnet-Unconfirmed-Request-PDU is:

```
Bit Number:  7   6   5   4   3   2   1   0
          |---|---|---|---|---|---|---|---|
          |  PDU Type     | 0 | 0 | 0 | 0 |
          |---|---|---|---|---|---|---|---|
          |          Service Choice       |
          |---|---|---|---|---|---|---|---|
          |          Service Request      |
          |                .              |
                           .
          |                .              |
          |---|---|---|---|---|---|---|---|
```

The PDU fields have the following values:

PDU Type = 1 (BACnet-Unconfirmed-Service-Request-PDU)
Service Choice = BACnetUnconfirmedServiceChoice
Service Request = Variable Encoding per 20.2.

Bits shown in the diagram as '0' shall be set to zero. These bits are currently unused and are reserved by ASHRAE.

20.1.4 BACnet-SimpleACK-PDU

The BACnet-SimpleACK-PDU is used to convey the information contained in a service response primitive ('Result(+)') that contains no other information except that the service request was successfully carried out.

```
BACnet-SimpleACK-PDU ::= SEQUENCE {
        pdu-type            [0] Unsigned (0..15), -- 2 for this PDU type
        reserved            [1] Unsigned (0..15), -- must be set to zero
        original-invokeID   [2] Unsigned (0..255),
        service-ACK-choice  [3] BACnetConfirmedServiceChoice
-- Context specific tags 0..3 are NOT used in header encoding
        }
```

The parameters of the BACnet-SimpleACK-PDU have the following meanings.

20.1.4.1 original-invokeID

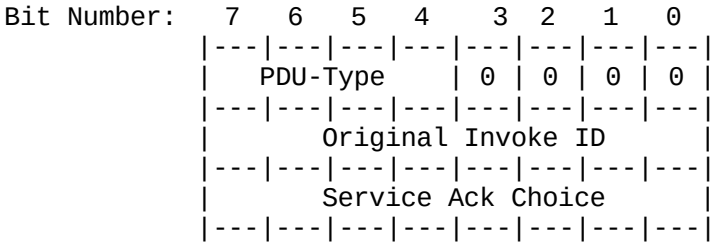
This parameter shall be the 'invokeID' contained in the confirmed service request being acknowledged.

20.1.4.2 service-ACK-choice

This parameter shall contain the value of the BACnetConfirmedServiceChoice corresponding to the service contained in the previous BACnet-Confirmed-Service-Request that has resulted in this acknowledgment. See Clause 21.

20.1.4.3 Format of the BACnet-SimpleACK-PDU

The format of the BACnet-SimpleACK-PDU is:



Note that this APDU is always three octets long.

The PDU fields have the following values:

PDU Type = 2 (BACnet-SimpleACK-PDU)
Original Invoke ID = (0..255)
Service ACK Choice = BACnetConfirmedServiceChoice

Bits shown in the diagram as '0' shall be set to zero. These bits are currently unused and are reserved by ASHRAE.

20.1.5 BACnet-ComplexACK-PDU

The BACnet-ComplexACK-PDU is used to convey the information contained in a service response primitive ('Result(+)') that contains information in addition to the fact that the service request was successfully carried out.

```
BACnet-ComplexACK-PDU ::= SEQUENCE {
        pdu-type                [0] Unsigned (0..15), --.3 for this PDU type
        segmented-message       [1] BOOLEAN,
        more-follows            [2] BOOLEAN,
        reserved                [3] Unsigned (0..3), -- must be set to zero
        original-invokeID       [4] Unsigned (0..255),
        sequence-number         [5] Unsigned (0..255) OPTIONAL, --only if segment
        proposed-window-size    [6] Unsigned (1..127) OPTIONAL, -- only if segment
        service-ACK-choice      [7] BACnetConfirmedServiceChoice,
        service-ACK             [8] BACnet-Confirmed-Service-ACK
-- Context specific tags 0..8 are NOT used in header encoding
        }
```

The parameters of the BACnet-ComplexACK-PDU have the following meanings.

20.1.5.1 segmented-message

This parameter indicates whether or not the confirmed service response is entirely, or only partially, contained in the present PDU. If the response is present in its entirety, the 'segmented-message' parameter shall be FALSE. If the present PDU contains only a segment of the response, this parameter shall be TRUE.

20.1.5.2 more-follows

This parameter is only meaningful if the 'segmented-message' parameter is TRUE. If 'segmented-message' is TRUE, then the 'more-follows' parameter shall be TRUE for all segments comprising the confirmed service response except for the last and shall be FALSE for the final segment. If 'segmented-message' is FALSE, then 'more-follows' shall be set FALSE by the encoder and shall be ignored by the decoder.

20.1.5.3 original-invokeID

This parameter shall be the 'invokeID' contained in the confirmed service request being acknowledged. The same 'original-invokeID' shall be used for all segments of a segmented acknowledgment.

20.1.5.4 sequence-number

This optional parameter is only present if the 'segmented-message' parameter is TRUE. In this case, the 'sequence-number' shall be a sequentially incremented unsigned integer, modulo 256, which identifies each segment of a segmented response. The value of the received 'sequence-number' is used by the original requester to acknowledge the receipt of one or more segments of a segmented response. The sequence-number of the first segment of a segmented response shall be zero.

20.1.5.5 proposed-window-size

This optional parameter is only present if the 'segmented-message' parameter is TRUE. In this case, the 'proposed-window-size' parameter shall specify as an unsigned binary integer the maximum number of message segments containing 'original-invokeID' the sender is able or willing to send before waiting for a segment acknowledgment PDU (see 5.2 and 5.3). The value of the 'proposed-window-size' shall be in the range 1 - 127.

20.1.5.6 service-ACK-choice

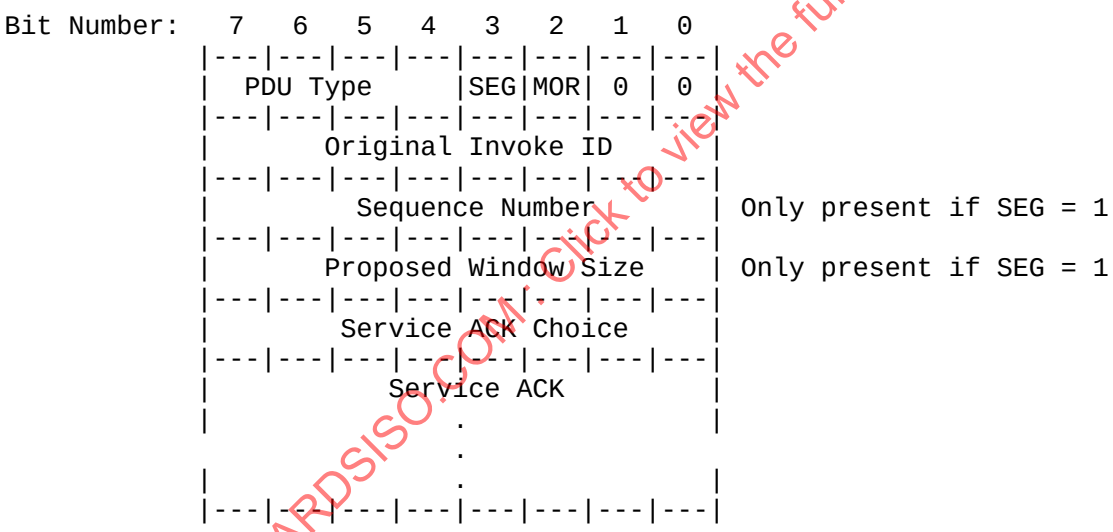
This parameter shall contain the value of the BACnetConfirmedServiceChoice corresponding to the service contained in the previous BACnet-Confirmed-Service-Request that has resulted in this acknowledgment. See Clause 21.

20.1.5.7 service-ACK

This parameter shall contain the parameters of the specific service acknowledgment that is being encoded according to the rules of 20.2. These parameters are defined in the individual service descriptions in this standard and are represented in Clause 21 in accordance with the rules of ASN.1.

20.1.5.8 Format of the BACnet-ComplexACK-PDU

The format of the BACnet-ComplexACK-PDU is:



The PDU fields have the following values:

PDU Type =	3 (BACnet-ComplexACK-PDU)
SEG =	0 (Unsegmented Response)
	1 (Segmented Response)
MOR =	0 (No More Segments Follow)
	1 (More Segments Follow)
Original Invoke ID =	(0..255)
Sequence Number =	(0..255) Only present if SEG = 1
Proposed Window Size =	(1..127) Only present if SEG = 1
Service ACK Choice =	BACnetConfirmedServiceChoice
Service ACK =	Variable Encoding per 20.2.

Bits shown in the diagram as '0' shall be set to zero. These bits are currently unused and are reserved by ASHRAE.

20.1.6 BACnet-SegmentACK-PDU

The BACnet-SegmentACK-PDU is used to acknowledge the receipt of one or more PDUs containing portions of a segmented message. It may also request the next segment or segments of the segmented message.

```
BACnet-SegmentACK-PDU ::= SEQUENCE {
        pdu-type            [0] Unsigned (0..15), -- 4 for this PDU type
        reserved            [1] Unsigned (0..3), -- must be set to zero
        negative-ACK        [2] BOOLEAN,
        server              [3] BOOLEAN,
        original-invokeID   [4] Unsigned (0..255),
        sequence-number     [5] Unsigned (0..255),
        actual-window-size  [6] Unsigned (1..127)
-- Context specific tags 0..6 are NOT used in header encoding
        }
```

The parameters of the BACnet-SegmentACK-PDU have the following meanings.

20.1.6.1 negative-ACK

This parameter shall be TRUE if the Segment-ACK PDU is being sent to indicate a segment received out of order. Otherwise, it shall be FALSE.

20.1.6.2 server

This parameter shall be TRUE when the SegmentACK PDU is sent by a server, that is, when the SegmentACK PDU is in acknowledgment of a segment or segments of a Confirmed-Request PDU.

This parameter shall be FALSE when the SegmentACK PDU is sent by a client, that is, when the SegmentACK PDU is in acknowledgment of a segment or segments of a ComplexACK PDU.

20.1.6.3 original-invokeID

This parameter shall be the 'invokeID' contained in the segment being acknowledged.

20.1.6.4 sequence-number

This parameter shall contain the 'sequence-number' of a previously received message segment. It is used to acknowledge the receipt of that message segment and all earlier segments of the message.

If the 'more-follows' parameter of the received message segment is TRUE, then the 'sequence-number' also requests continuation of the segmented message beginning with the segment whose 'sequence-number' is one plus the value of this parameter, modulo 256.

20.1.6.5 actual-window-size

This parameter shall specify as an unsigned binary integer the number of message segments containing 'original-invokeID' the sender will accept before sending another SegmentACK. See 5.3 for additional details. The value of the 'actual-window-size' shall be in the range 1 - 127.

20.1.6.6 Format of the BACnet-SegmentACK-PDU

The format of the BACnet-SegmentACK-PDU is:

```
Bit Number:     7   6   5   4   3   2   1   0
              |---|---|---|---|---|---|---|---|
              |    PDU-Type   | 0 | 0 |NAK|SRV|
              |---|---|---|---|---|---|---|---|
              |       Original Invoke ID      |
              |---|---|---|---|---|---|---|---|
              |        Sequence Number        |
              |---|---|---|---|---|---|---|---|
              |       Actual Window Size      |
              |---|---|---|---|---|---|---|---|
```

Note that this PDU is always four octets long.

The PDU fields have the following values:

PDU Type =	4 (BACnet-SegmentACK-PDU)
NAK =	0 (Normal Acknowledgment)
	1 (Negative Acknowledgment, Segment Out of Order)
SRV =	0 (Sent by Client)
	1 (Sent by Server)
Original Invoke ID =	(0..255)
Sequence Number =	(0..255)
Actual Window Size =	(1..127)

Bits shown in the diagram as '0' shall be set to zero. These bits are currently unused and are reserved by ASHRAE.

20.1.7 BACnet-Error-PDU

The BACnet-Error-PDU is used to convey the information contained in a service response primitive ('Result(-)') that indicates the reason why a previous confirmed service request failed in its entirety.

```
BACnet-Error-PDU ::= SEQUENCE {
        pdu-type            [0] Unsigned (0..15), -- 5 for this PDU type
        reserved            [1] Unsigned (0..15), -- must be set to zero
        original-invokeID   [2] Unsigned (0..255),
        error-choice        [3] BACnetConfirmedServiceChoice,
        error               [4] BACnet-Error
-- Context specific tags 0..4 are NOT used in header encoding
        }
```

The parameters of the BACnet-Error-PDU have the following meanings.

20.1.7.1 original-invokeID

This parameter shall be the 'invokeID' contained in the confirmed service request to which the error is a response.

20.1.7.2 error-choice

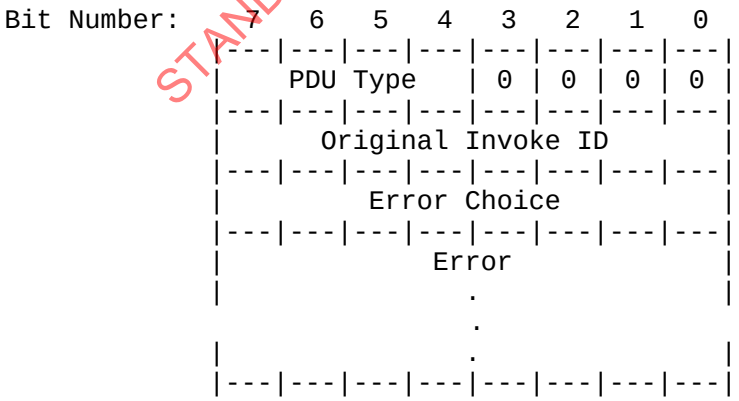
This parameter, of type BACnetConfirmedServiceChoice, shall contain the tag value of the BACnet-Error choice. See Clause 21.

20.1.7.3 error

This parameter, of type BACnet-Error, indicates the reason the indicated service request could not be carried out. This parameter shall be encoded according to the rules of 20.2.

20.1.7.4 Format of the BACnet-Error-PDU

The format of the BACnet-Error-PDU is:



The PDU fields have the following values:

PDU Type = 5 (BACnet-Error-PDU)
Original Invoke ID = (0..255)
Error Choice = BACnetConfirmedServiceChoice
Error = Variable Encoding per 20.2.

Bits shown in the diagram as '0' shall be set to zero. These bits are currently unused and are reserved by ASHRAE.

20.1.8 BACnet-Reject-PDU

The BACnet-Reject-PDU is used to reject a received confirmed request PDU based on syntactical flaws or other protocol errors that prevent the PDU from being interpreted or the requested service from being provided. Only confirmed request PDUs may be rejected (see 18.8).

```
BACnet-Reject-PDU ::= SEQUENCE {
        pdu-type            [0] Unsigned (0..15), -- 6 for this PDU type
        reserved            [1] Unsigned (0..15), -- must be set to zero
        original-invokeID   [2] Unsigned (0..255),
        reject reason       [3] BACnetRejectReason
-- Context specific tags 0..3 are NOT used in header encoding
        }
```

The parameters of the BACnet-Reject-PDU have the following meanings.

20.1.8.1 original-invokeID

This parameter shall be the 'invokeID' of the PDU being rejected.

20.1.8.2 reject-reason

This parameter, of type BACnetRejectReason, contains the reason the PDU with the indicated 'invokeID' is being rejected.

20.1.8.3 Format of the BACnet-Reject-PDU

The format of the BACnet-Reject-PDU is:

```
Bit Number:  7   6   5   4   3   2   1   0
            |---|---|---|---|---|---|---|---|
            |    PDU-Type   | 0 | 0 | 0 | 0 |
            |---|---|---|---|---|---|---|---|
            |       Original Invoke ID      |
            |---|---|---|---|---|---|---|---|
            |         Reject Reason         |
            |---|---|---|---|---|---|---|---|
```

Note that this PDU is always three octets long.

The PDU fields have the following values:

PDU Type = 6 (BACnet-Reject-PDU)
Original Invoke ID = (0..255)
Reject Reason = One octet containing the reject reason enumeration

Bits shown in the diagram as '0' shall be set to zero. These bits are currently unused and are reserved by ASHRAE.

20.1.9 BACnet-Abort-PDU

The BACnet-Abort-PDU is used to terminate a transaction between two peers.

```
BACnet-Abort-PDU ::= SEQUENCE {
        pdu-type            [0] Unsigned (0..15), -- 7 for this PDU type
        reserved            [1] Unsigned (0..7), -- must be set to zero
        server              [2] BOOLEAN,
        original-invokeID   [3] Unsigned (0..255),
        abort-reason        [4] BACnetAbortReason
-- Context specific tags 0..4 are NOT used in header encoding
        }
```

The parameters of the BACnet-Abort-PDU have the following meanings.

20.1.9.1 server

This parameter shall be TRUE when the Abort PDU is sent by a server. This parameter shall be FALSE when the Abort PDU is sent by a client.

20.1.9.2 original-invokeID

This parameter shall be the 'invokeID' of the transaction being aborted.

20.1.9.3 abort-reason

This parameter, of type BACnetAbortReason, contains the reason the transaction with the indicated invoke ID is being aborted.

20.1.9.4 Format of the BACnet-Abort-PDU

The format of the BACnet-Abort-PDU is:

```
Bit Number:   7   6   5   4   3   2   1   0
            |---|---|---|---|---|---|---|---|
            |    PDU Type   | 0 | 0 | 0 |SRV|
            |---|---|---|---|---|---|---|---|
            |      Original Invoke ID       |
            |---|---|---|---|---|---|---|---|
            |          Abort Reason         |
            |---|---|---|---|---|---|---|---|
```

Note that this PDU is always three octets long.

The PDU fields have the following values:

```
PDU Type =              7 (BACnet-Abort-PDU)
SRV =                   0 (Sent by Client)
                        1 (Sent by Server)
Original Invoke ID =    (0..255)
```

Bits shown in the diagram as '0' shall be set to zero. These bits are currently unused and are reserved by ASHRAE.

20.2 Encoding the Variable Part of BACnet APDUs

The encoding of the header portions of BACnet APDUs has been specified in 20.1. This subclause describes the encoding procedures for the variable portion of BACnet APDUs referred to hereafter as "service parameters." These parameters are of types BACnet-Confirmed-Service-Request, BACnet-Unconfirmed-Service-Request, BACnet-Confirmed-Service-ACK, and BACnet-Error. Each parameter is unambiguously defined by means of ASN.1 productions in Clause 21.

All data elements in service parameters are identified by constructs known as "tags." Each tag refers to a unique parameter or subparameter.

BACnet encoding uses two classes of tag. The first identifies fundamental datatypes used or defined in this standard, such as BOOLEANs, Unsigneds, CharacterStrings, Date, Time, or BACnetObjectIdentifiers. Where a datatype appears in upper case, its semantics are identical to the corresponding ASN.1 universal datatype of the same name, as indicated in Clause 21.

Such tags are called "application" tags, and the values of the tags are specified in 20.2.1.4.

The second class of tag is used to identify data elements whose datatype may be inferred from the context in which they appear. These tags are called "context specific" tags.

ASN.1 defines two other classes of tags, "universal" and "private." The encoding scheme used by BACnet does not allow, nor do any BACnet ASN.1 productions require, the use of these classes of tags.

In some instances, the datatype of a parameter cannot be deduced from the context in which it appears. In such cases, both a context specific and one or more application tags are required. These cases are indicated in the ASN.1 productions by service parameters whose datatypes are indicated by the keywords ABSTRACT-SYNTAX.&TYPE (ANY), CHOICE, SEQUENCE, or SEQUENCE OF.

The subclauses that follow show how each tagged element is identified, its length specified, and its value encoded.

20.2.1 General Rules For Encoding BACnet Tags

BACnet tags are encoded in an initial octet and zero or more conditional subsequent octets. The initial octet is defined as follows:

```
Bit Number:   7      6    5     4     3     2     1     0
            |-----|-----|-----|-----|-----|-----|-----|-----|
            |       Tag Number      |Class|Length/Value/Type|
            |-----|-----|-----|-----|-----|-----|-----|-----|
```

where Tag Number = the tag number within the class
Class = the class of tag (application or context specific)
Length/Value/Type = whether the data following the tag is primitive or constructed and specifies the length or value of primitive data.

20.2.1.1 Class

The Class bit shall be zero for application tags. The Class bit shall be one for context specific tags.

20.2.1.2 Tag Number

Tag numbers ranging from zero to 14 (inclusive) shall be encoded in the Tag Number field of the initial octet as a four bit binary integer with bit 7 the most significant bit.

Tag numbers ranging from 15 to 254 (inclusive) shall be encoded by setting the Tag Number field of the initial octet to B'1111' and following the initial tag octet by an octet containing the tag number represented as an eight-bit binary integer with bit 7 the most significant bit.

The encoding does not allow, nor does BACnet require, tag numbers larger than 254. The value B'11111111' of the subsequent octet is reserved by ASHRAE.

20.2.1.3 Length/Value/Type

The content of the length/value/type field of the initial octet distinguishes between primitive and constructed encodings and specifies the length or value of primitive data. A primitive encoding is one in which the data do not contain other tagged encodings. A constructed encoding is one in which the data do contain other tagged encodings.

20.2.1.3.1 Primitive Data

If the data being encoded are application class BOOLEAN data, then the Boolean value shall be encoded by setting the length/value/type field of the initial octet to B'000' if the Boolean value is FALSE or B'001' if the Boolean value is TRUE. In this case, the length/value/type field shall be interpreted as a value.

If the data being encoded are primitive (that is, not constructed) and not application class BOOLEAN data, then the value of the data shall be encoded according to 20.2.2 through 20.2.14, and the length/value/type field of the initial tag octet shall specify the length of the primitive data in octets as follows:

Data length in octets ranging from zero to four (inclusive) shall be encoded in the length/value/type field of the initial octet as a three-bit binary integer with bit 2 the most significant bit.

Data length in octets ranging from 5 to 253 (inclusive) shall be encoded by setting the length/value/type field of the initial octet to B'101' and following the initial tag octet or, if the Tag Number has been extended, following the Tag Number extension octet by an octet containing the data length represented as an eight-bit binary integer with bit 7 the most significant bit.

Data length in octets ranging from 254 to 65535 (inclusive) shall be encoded by setting the length/value/type field of the initial octet to B'101' and following the initial tag octet or, if the Tag Number has been extended, following the Tag Number extension octet by an octet containing D'254' and two additional octets whose value contains the data length represented as a 16-bit binary integer with the most significant octet first.

Data length in octets ranging from 65536 to 2^{32}-1 (inclusive) shall be encoded by setting the length/value/type field of the initial octet to B'101' and following the initial tag octet or, if the Tag Number has been extended, following the Tag Number extension octet by an octet containing D'255' and four additional octets whose value contains the data length represented as a 32-bit binary integer with the most significant octet first.

Data lengths larger than 2^{32}-1 are not encodable using primitive tags.

Note that with the exception of 8-octet IEEE-754 double precision floating point values and certain bit, character, and octet strings, the length of BACnet application-tagged primitives will fit in the tag octet without extension.

20.2.1.3.2 Constructed Data

If the production being encoded contains tagged elements, then the encoding is called "constructed" and shall consist of

(a) an "opening" tag whose Tag Number field shall contain the value of the tag number, whose Class field shall indicate "context specific," and whose length/value/type field shall have the value B'110';

(b) the complete encoding, with tags, of the zero, one, or more elements that comprise the data;

(c) a "closing" tag, whose Class and Tag Number fields shall contain the same values as the "opening" tag and whose length/value/type field shall have the value B'111'.

In this case, the length/value/type fields of the "opening" and "closing" tags shall be interpreted as types.

Note that a contained tagged element may itself be a constructed element. This recursion does not result in ambiguous encoding, as each "opening" tag must have a corresponding "closing" tag that will be contained within any outer "opening" and "closing" tags.

20.2.1.4 Application Tags

The Tag Number field of an encoded BACnet application tag shall specify the application datatype as follows:

Tag Number: 0 = Null
1 = Boolean
2 = Unsigned Integer

3 = Signed Integer (2's complement notation)
4 = Real (ANSI/IEEE-754 floating point)
5 = Double (ANSI/IEEE-754 double precision floating point)
6 = Octet String
7 = Character String
8 = Bit String
9 = Enumerated
10 = Date
11 = Time
12 = BACnetObjectIdentifier
13, 14, 15 = Reserved for ASHRAE

Note that all currently defined BACnet Application datatypes are primitively encoded.

20.2.1.5 Context-Specific Tags

The Tag Number field of an encoded BACnet context-specific tag shall contain the value of the context-specific tag number.

The data delimited by a context-specific tag may be either primitive or constructed.

20.2.2 Encoding of a Null Value

The encoding of a Null value shall be primitive, with no contents octet.

Example: Application-tagged null value
ASN.1 = NULL
Application Tag = Null (Tag Number = 0)
Encoded Tag = X'00'

20.2.3 Encoding of a Boolean Value

Application-tagged Boolean values shall be encoded within a single octet by setting the length/value/type field to B'000' if the value to be encoded is FALSE or B'001' if the value to be encoded is TRUE.

Example: Application-tagged Boolean value
ASN.1 = BOOLEAN
Value = FALSE
Application Tag = Boolean (Tag Number = 1)
Encoded Tag = X'10'

Context-tagged Boolean primitive data shall contain one contents octet. The value of this octet shall be B'00000000' if the value to be encoded is FALSE or B'00000001' if the value to be encoded is TRUE.

Example: Context-tagged Boolean value
ASN.1 = [2] BOOLEAN
Value = TRUE
Context Tag = 2
Encoded Tag = X'29'
Encoded Data = X'01'

NOTE: The Boolean datatype differs from the other datatypes in that the encoding of a context-tagged Boolean value is not the same as the encoding of an application-tagged Boolean value. This is done so that the application-tagged value may be encoded in a single octet, without a contents octet. While this same encoding could have been used for the context-tagged case, doing so would require that the context be known in order to distinguish between a length or a value in the length/value/type field. This was considered to be undesirable. See 20.2.20.

20.2.4 Encoding of an Unsigned Integer Value

The encoding of an unsigned integer value shall be primitive, with at least one contents octet.

Unsigned integers shall be encoded in the contents octet(s) as binary numbers in the range 0 to $(2^{8*L} - 1)$ where L is the number of octets used to encode the value and L is at least one. Values encoded into more than one octet shall be conveyed with the most significant octet first. All unsigned integers shall be encoded in the smallest number of octets possible. That is, the first octet of any multi-octet encoded value shall not be X'00'.

Example: Application-tagged unsigned integer
ASN.1 = Unsigned
Value = 72
Application Tag = Unsigned Integer (Tag Number = 2)
Encoded Tag = X'21'
Encoded Data = X'48'

20.2.5 Encoding of a Signed Integer Value

The encoding of a signed integer value shall be primitive, with at least one contents octet.

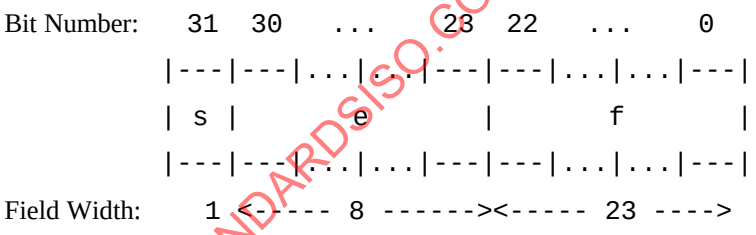
Signed integers shall be encoded in the contents octet(s) as binary numbers using 2's complement notation in the range $-2^{(8*L-1)}$ to $(2^{(8*L-1)} - 1)$ where L is the number of octets used to encode the value and L is at least one. Values encoded into more than one octet shall be conveyed most significant octet first. All signed integers shall be encoded in the smallest number of octets possible. That is, the first octet of any multi-octet encoded value shall not be X'00' if the most significant bit (bit 7) of the second octet is 0, and the first octet shall not be X'FF' if the most significant bit of the second octet is 1.

Example: Application-tagged signed integer
ASN.1 = INTEGER
Value = 72
Application Tag = Signed Integer (Tag Number = 3)
Encoded Tag = X'31'
Encoded Data = X'48'

20.2.6 Encoding of a Real Number Value

The encoding of a real number value shall be primitive, with four contents octets. Real numbers shall be encoded using the method specified in ANSI/IEEE Standard 754-1985, "IEEE Standard for Binary Floating-Point Arithmetic." This standard should be consulted for details. The multi-octet value shall be conveyed with the most significant (sign and exponent) octet first.

For the case of single precision real numbers, the encoding format is:



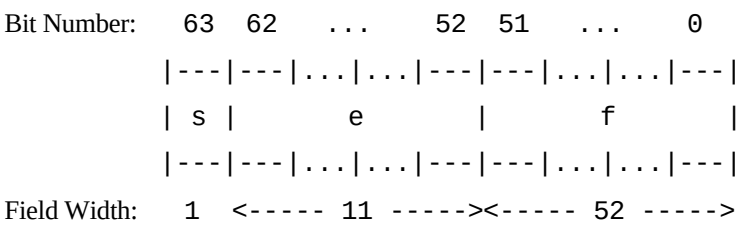
where the numbers indicate the field widths in bits. Non-zero values shall be represented by the equation $v = (-1)^s 2^{e-127}(1 \bullet f)$ where the symbol "•" signifies the binary point. Zero shall be indicated by setting s, e, and f to zero.

Example: Application-tagged single precision real
ASN.1 = REAL
Value = 72.0
Application Tag = Real (Tag Number = 4)
Encoded Tag = X'44'
Encoded Data = X'42900000'

20.2.7 Encoding of a Double Precision Real Number Value

The encoding of a double precision real number value shall be primitive with eight contents octets. Double precision real numbers shall be encoded using the method specified in ANSI/IEEE Standard 754-1985, "IEEE Standard for Binary Floating-Point Arithmetic." This standard should be consulted for details. The multi-octet value shall be conveyed most significant (sign and exponent) octet first.

For the case of double precision real numbers, the encoding format is:



where the numbers indicate the field widths in bits. Non-zero values shall be represented by the equation $v = (-1)^s 2^{e-1023}(1 \bullet f)$ where the symbol "•" signifies the binary point. Zero shall be indicated by setting s, e, and f to zero.

Example: Application-tagged double precision real

ASN.1 =	Double
Value =	72.0
Application Tag =	Double (Tag Number = 5)
Encoded Tag =	X'55'
Extended Length =	X'08'
Encoded Data =	X'4052000000000000'

ANSI/IEEE-754 Extended Precision format is not supported by BACnet.

20.2.8 Encoding of an Octet String Value

The encoding of an octet string value shall be primitive.

The encoding shall contain zero, one, or more contents octets equal in value to the octets in the data value, in the order in which they appear in the data value, and with the most significant bit of an octet of the data value aligned with the most significant bit of an octet of the contents octets.

Example: Application-tagged octet string

ASN.1 =	OCTET STRING
Value =	X'1234FF'
Application Tag =	Octet String (Tag Number = 6)
Encoded Tag =	X'63'
Encoded Data =	X'1234FF'

20.2.9 Encoding of a Character String Value

The encoding of a character string value shall be primitive.

The encoding shall contain an initial contents octet, and zero, one, or more additional contents octets equal in value to the octets in the data value, in the order in which they appear in the data value, i.e., most significant octet first, and with the most significant bit of an octet of the data value aligned with the most significant bit of an octet of the contents octets.

The initial octet shall specify the character set with the following encoding:

X'00'	ANSI X3.4
X'01'	IBM™/Microsoft™ DBCS
X'02'	JIS C 6226
X'03'	ISO 10646 (UCS-4)

X'04' ISO 10646 (UCS-2)
X'05' ISO 8859-1

Other values of the initial octet are reserved by ASHRAE.
Example: Application-tagged character string

ASN.1 =	CharacterString
Value =	"This is a BACnet string!" (ANSI X3.4)
Application Tag =	Character String (Tag Number = 7)
Encoded Tag =	X'75'
Length Extension =	X'19'
Character Set =	X'00' (ANSI X3.4)
Encoded Data =	X'546869732069732061204241 436E657420737472696E6721'

In the case of IBM/Microsoft DBCS (X'01'), the initial octet shall be followed by two additional octets whose value shall represent an unsigned integer, with the most significant octet first, that shall indicate the Code Page to be presumed for the characters that follow.

Example: Application-tagged character string (DBCS)

ASN.1 =	CharacterString
Value =	"This is a BACnet String!" (IBM/Microsoft DBCS, code page 850)
Application Tag =	Character String (Tag Number = 7)
Encoded Tag =	X'75'
Length Extension =	X'1B'
Encoded Data =	X'0103525468697320697320612042 41 436E657420737472696E6721'

In the case of ISO 10646 UCS-2 (X'04') and UCS4 (X'03'), each character of the string shall be represented by two or four octets, respectively. The octet order for UCS-2 shall be Row-Cell. The octet order for UCS-4 shall be Group-Plane-Row-Cell.

Example: Application-tagged character string (UCS-2)

ASN.1 =	CharacterString
Value =	"This is a BACnet String!" (ISO 10646 UCS-2)
Application Tag =	Character String (Tag Number = 7)
Encoded Tag =	X'75'
Length Extension =	X'31'
Encoded Data =	X'04005400680069007300200069007300200061002000420041 0043006E006500740020007300740072006900 6E00670021'

20.2.10 Encoding of a Bit String Value

The encoding of a bit string value shall be primitive.

The contents octets for the primitive encoding shall contain an initial octet and zero or more subsequent octets containing the bit string. The initial octet shall encode, as an unsigned binary integer, the number of unused bits in the final subsequent octet. The number of unused bits shall be in the range zero to seven, inclusive.

Bit strings defined in this standard, e.g., the Status_Flags property, shall be encoded in the order of definition, with the first defined Boolean value in the most significant bit, i.e. bit 7, of the first subsequent octet. The bits in the bitstring shall be placed in bits 7 to 0 of the first subsequent octet, followed by bits 7 to 0 of the second subsequent octet, followed by bits 7 to 0 of each octet in turn, followed by as many bits as are needed of the final subsequent octet, commencing with bit 7.

If the bit string is empty, there shall be no subsequent octets, and the initial octet shall be zero.

Example: Application-tagged bit string

ASN.1 =	BIT STRING
Value =	B'10101'

Application Tag = Bit String (Tag Number = 8)
Encoded Tag = X'82'
Encoded Data = X'03A8'

20.2.11 Encoding of an Enumerated Value

The encoding of an enumerated value shall be primitive, with at least one contents octet.

Enumerated values shall be encoded in the contents octet(s) as binary numbers in the range 0 to (2^{8*L} - 1) where L is the number of octets used to encode the value and L is at least one. Values encoded into more than one octet shall be conveyed most significant octet first. All enumerated values shall be encoded in the smallest number of octets possible. That is, the first octet of any multi-octet encoded value shall not be X'00'.

Example: Application-tagged enumeration
ASN.1 = BACnetObjectType
Value = ANALOG-INPUT (0)
Application Tag = Enumerated (Tag Number = 9)
Encoded Tag = X '91'
Encoded Data = X '00'

20.2.12 Encoding of a Date Value

The encoding of a date value shall be primitive, with four contents octets.

Date values shall be encoded in the contents octets as four binary integers. The first contents octet shall represent the year minus 1900; the second octet shall represent the month, with January = 1; the third octet shall represent the day of the month; and the fourth octet shall represent the day of the week, with Monday = 1. A value of X'FF' = D'255' in any of the four octets shall indicate that the corresponding value is unspecified. If all four octets = X'FF', the corresponding date may be interpreted as "any" or "don't care."

Example: Application-tagged date value
ASN.1 = Date
Value = January 24, 1991 (Day of week = Thursday)
Application Tag = Date (Tag Number = 10)
Encoded Tag = X'A4'
Encoded Data = X'5B011804'

20.2.13 Encoding of a Time Value

The encoding of a time value shall be primitive, with four contents octets.

Time values shall be encoded in the contents octets as four binary integers. The first contents octet shall represent the hour, in the 24-hour system (1 P.M. = D'13'); the second octet shall represent the minute of the hour; the third octet shall represent the second of the minute; and the fourth octet shall represent the fractional part of the second in hundredths of a second. A value of X'FF' = D'255' in any of the four octets shall indicate that the corresponding value is unspecified. If all four octets = X'FF', the corresponding time may be interpreted as "any" or "don't care."

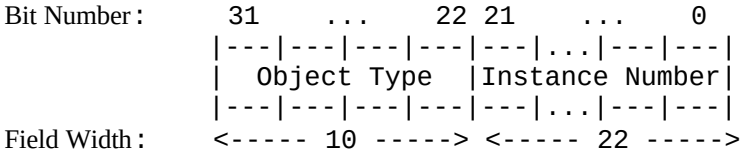
Example: Application-tagged time value
ASN.1 = Time
Value = 5:35:45.17 P.M. = 17:35:45.17
Application Tag = Time (Tag Number = 11)
Encoded Tag = X'B4'
Encoded Data = X'11232D11'

20.2.14 Encoding of an Object Identifier Value

A BACnet Object Identifier value shall consist of two components:

(1) A 10-bit object type, representing the BACnetObjectType of the object, with bit 9 the most significant bit and bit 0 the least significant. For objects defined in this standard, the value for this field shall be determined by the BACnetObjectType enumeration in Clause 21.

(2) A 22-bit object instance number, with bit 21 the most significant bit and bit 0 the least significant.



The encoding of an object identifier value shall be primitive, with four contents octets as follows:

Bits 9 through 2 of the object type shall be encoded in bits 7 through 0 of the first contents octet. Bits 1 through 0 of the object type shall be encoded in bits 7 through 6 of the second contents octet.

Bits 21 through 16 of the object instance shall be encoded in bits 5 through 0 of the second contents octet. Bits 15 through 8 of the object instance shall be encoded in bits 7 through 0 of the third contents octet. Bits 7 through 0 of the object instance shall be encoded in bits 7 through 0 of the fourth contents octet.

Example: Application-tagged object identifier value

ASN.1 =	ObjectIdentifier
Value =	(Binary Input, 15)
Application Tag =	ObjectIdentifier (Tag Number = 12)
Encoded Tag =	X'C4'
Encoded Data =	X'00C0000F'

20.2.15 Encoding of a Tagged Value

The encoding of a tagged value shall be derived from the complete encoding of the corresponding data value.

ISO 8824 defines the keywords "IMPLICIT" and "EXPLICIT," with "EXPLICIT" the default. Clause 21 begins with a "DEFINITION IMPLICIT TAGS," which changes the default to IMPLICIT. BACnet ASN.1 definitions are in terms of this default and use EXPLICIT only as an override.

If the "EXPLICIT" keyword is used in the production for the type, the encoding shall be constructed, and the contents octets shall be the complete base encoding, including tags.

If the "EXPLICIT" keyword is not used in the definition of the type, then

a) the encoding shall be constructed if the base encoding is constructed and shall be primitive otherwise, and either

b) the contents octets shall be the same as the contents octets of the base encoding if the base encoding is not primitively tagged application class Boolean or

c) the contents octet shall contain the value B'00000000' to denote a Boolean value of FALSE or B'00000001' to denote a Boolean value of TRUE if the base encoding is primitively tagged application class Boolean.

The context tag numbers shown in the following examples are for illustrative purposes only.

Example: Context-tagged null value

ASN.1 =	[3] NULL
Context Tag =	3
Encoded Tag =	X'38'

Example: Context-tagged Boolean value

ASN.1 =	[6] BOOLEAN
Value =	FALSE
Context Tag =	6
Encoded Tag =	X'69'
Encoded Data =	X'00'

Example: Context-tagged unsigned integer

ASN.1 =	[0] Unsigned
Value =	256
Context Tag =	0
Encoded Tag =	X'0A'
Encoded Data =	X'0100'

Example: Context-tagged signed integer

ASN.1 =	[5] INTEGER
Value =	-72
Context Tag =	5
Encoded Tag =	X'59'
Encoded Data =	X'B8'

Example: Context-tagged single precision real

ASN.1 =	[0] REAL
Value =	-33.3
Context Tag =	0
Encoded Tag =	X'0C'
Encoded Data =	X'C2053333'

Example: Context-tagged double precision real

ASN.1 =	[1] Double
Value =	-33.3
Context Tag =	1
Encoded Tag =	X'1D'
Extended Length =	X'08'
Encoded Data =	X'C040A66666666666'

Example: Context-tagged octet string

ASN.1 =	[1] OctetString
Value =	X'4321'
Context Tag =	1
Encoded Tag =	X'1A'
Encoded Data =	X'4321'

Example: Context-tagged character string

ASN.1 =	[5] CharacterString
Value =	"This is a BACnet string!" (ANSI X3.4)
Context Tag =	5
Encoded Tag =	X'5D'
Length Extension =	X'19'
Character Set =	X'00' (ANSI X3.4)
Encoded Data =	X'546869732069732061204241 436E657420737472696E6721'

Example: Context-tagged bit string

ASN.1 =	[0] BIT STRING
Value =	B'10101'
Context Tag =	0
Encoded Tag =	X'0A'
Unused Bits in Last Octet =	X'03'
Encoded Data =	X'A8'

Example: Context-tagged enumeration

ASN.1 =	[9] BACnetObjectType
Value =	ANALOG-INPUT (0)
Context Tag =	9
Encoded Tag =	X'99'
Encoded Data =	X'00'

Example: Context-tagged date value

ASN.1 =	[9] Date
Value =	January 24, 1991 (Day of week = Thursday)
Context Tag =	9
Encoded Tag =	X'9C'
Encoded Data =	X'5B011805'

Example: Context-tagged time value

ASN.1 =	[4] Time
Value =	5:35:45.17 P.M. = 17:35:45.17
Context Tag =	4
Encoded Tag =	X'4C'
Encoded Data =	X'11232D11'

Example: Context-tagged object identifier value

ASN.1 =	[4] ObjectIdentifier
Value =	(Binary Input, 15)
Context Tag =	4
Encoded Tag =	X'4C'
Encoded Data =	X'00C0000F'

20.2.16 Encoding of a Sequence Value

The encoding of a sequence value shall consist of the complete encoding, including tags, of one data value from each of the types listed in the ASN.1 production for the sequence type, in the order of their appearance in the definition, unless the type was referenced with the keyword "OPTIONAL".

The encoding of a data value may, but need not, be present for a type that was referenced with the keyword "OPTIONAL". If present, it shall appear in the encoding at the point corresponding to the appearance of the type in the ASN.1 definition.

Example: SEQUENCE value

ASN.1 =	BACnetDateTime
Value =	January 24, 1991, 5:35:45.17 P.M.
Application Tag =	Date (Tag Number = 10)
Encoded Tag =	X'A4'
Encoded Data =	X'5B011805'
Application Tag =	Time (Tag Number = 11)
Encoded Tag =	X'B4'
Encoded Data =	X'11232D11'

Example: Context-tagged SEQUENCE value

```
ASN.1 =          [0] BACnetDateTime
Value =          January 24, 1991, 5:35:45.17 P.M.
Context Tag =    0
Encoded Tag =    X'0E' (opening tag)
        Application Tag =Date (Tag Number = 10)
        Encoded Tag =    X'A4'
        Encoded Data =   X'5B011805'
        Application Tag = Time (Tag Number = 11)
        Encoded Tag =    X'B4'
        Encoded Data =   X'11232D11'
Encoded Tag =    X'0F' (closing tag)
```

All ASN.1 productions of sequences that contain structured elements shall have distinct tags as necessary to permit unambiguous encoding and decoding of values. The follow example illustrates this requirement.

```
(incorrect usage)
Var1 ::= SEQUENCE {
        varn1 SEQUENCE {
                varn2    [1] INTEGER,
                varn3    [2] INTEGER OPTIONAL
                },
        varn4 SEQUENCE {
                varn5    [1] INTEGER OPTIONAL,
                varn6    [2] INTEGER
                }
        }

(correct usage)
Var1 ::= SEQUENCE {
        varn1 SEQUENCE {
                varn2    [1] INTEGER,
                varn3    [2] INTEGER OPTIONAL
                },
        varn4 SEQUENCE {
                varn5    [3] INTEGER OPTIONAL,
                varn6    [4] INTEGER
                }
        }
```

20.2.17 Encoding of a Sequence-Of Value

The encoding of a sequence-of value shall consist of zero, one, or more complete encodings, including tags, of data values from the types listed in the ASN.1 definition.

The order of the encodings of the data values shall be the same as the order of the data values in the sequence-of value to be encoded.

Example: SEQUENCE OF primitive data
ASN.1 = SEQUENCE OF INTEGER
Value = 1,2,4
Application Tag = Unsigned Integer (Tag Number = 2)
Encoded Tag = X'21'
Encoded Data = X'01'
Application Tag = Unsigned Integer (Tag Number = 2)
Encoded Tag = X'21'
Encoded Data = X'02'
Application Tag = Unsigned Integer (Tag Number = 2)
Encoded Tag = X'21'
Encoded Data = X'04'

Example: Context-tagged SEQUENCE OF primitive data
ASN.1 = [1] SEQUENCE OF INTEGER
Value = 1,2,4
Encoded Tag = X'1E' (Opening Tag)
Application Tag = Unsigned Integer (Tag Number = 2)
Encoded Tag = X'21'
Encoded Data = X'01'
Application Tag = Unsigned Integer (Tag Number = 2)
Encoded Tag = X'21'
Encoded Data = X'02'
Application Tag = Unsigned Integer (Tag Number = 2)
Encoded Tag = X'21'
Encoded Data = X'04'
Encoded Tag = X'1F' (Closing Tag)

Example: SEQUENCE OF constructed data
ASN.1 = SEQUENCE OF BACnetDateTime
Value = (January 24, 1991, 5:00 P.M.),
(January 24, 1991, 6:45 P.M.)
Application Tag = Date (Tag Number = 10)
Encoded Tag = X'A4'
Encoded Data = X'5B011804'
Application Tag = Time (Tag Number = 11)
Encoded Tag = X'B4'
Encoded Data = X'11000000'
Application Tag = Date (Tag Number = 10)
Encoded Tag = X'A4'
Encoded Data = X'5B011804'
Application Tag = Time (Tag Number = 11)
Encoded Tag = X'B4'
Encoded Data = X'122D0000'

20.2.18 Encoding of a Choice Value

The encoding of a CHOICE value shall be the same as the encoding of a value of the chosen type. The encoding may be primitive or constructed depending on the chosen type.

Example: CHOICE of primitive data
ASN.1 = BACnetTimeStamp
Value = 5:35:45.17 P.M. = 17:35:45.17
Context Tag = 0 (Choice for 'time' in BACnetTimeStamp)
Encoded Tag = X'0C'
Encoded Data = X'11232D11'

Example: CHOICE of constructed data

ASN.1 =	BACnetTimeStamp
Value =	January 24, 1991, 5:45.17 P.M.
Context Tag =	2 (Choice for 'dateTime' in BACnetTimeStamp)
Encoded Tag =	X'2E' (Opening Tag)
	Application Tag = Date (Tag Number = 10)
	Encoded Tag = X'A4'
	Encoded Data = X'5B011804'
	Application Tag = Time (Tag Number = 11)
	Encoded Tag = X'B4'
	Encoded Data = X'11232D11'
Encoded Tag =	X'2F' (Closing Tag)

20.2.19 Encoding of a Value of the ANY Type

The encoding of an ANY type shall be the complete encoding specified in this standard for the type of the value substituted for the placeholder ANY. This is represented in ASN.1 by ABSTRACT-SYNTAX.&Type.

20.2.20 Summary of the Tagging Rules

While the tagged portion of a BACnet PDU cannot be interpreted without knowledge of the context, the tagging rules described in 20.2 result in a tagged stream that can be unambiguously parsed even without a priori knowledge of the context.

(a) The first octet in a stream shall be a tag, either context specific or application class.

(b) If a tag is application class, then the format and extent of its data are known according to the definitions in 20.2. In particular, the data, if any, may be bypassed and the next tag in the stream found.

(c) If a tag is context specific and primitive, then it contains primitive (untagged) data of some type. The length/value/type field of the tag specifies the length of this data. Thus, while the datatype and format of the data may be unknown, its length is known exactly. This allows the data, if any, to be bypassed and the next tag in the stream found.

(d) If a tag is constructed (length/value/type = 6), then it is the opening tag of a pair. Following this tag shall be a sequence of zero or more tagged elements, followed by the closing tag of the pair with length/value/type = 7. The tagged stream between opening and closing tags may be parsed according to these same four rules via a process of recursive descent until only primitive tags are encountered or until no tags are encountered.

21 FORMAL DESCRIPTION OF APPLICATION PROTOCOL DATA UNITS

This clause consists of an ASN.1 module that defines the BACnet APDUs and all necessary underlying datatypes. Clauses 13 through 17 contain many service parameters that are defined as conditional (C) or user optional (U). Both of these parameter types are designated as OPTIONAL in the ASN.1 productions to indicate that they may or may not be present in the PDU. The use of OPTIONAL in the ASN.1 production shall not supersede the conditional requirements defined in the service specification.

```
BACnetModule DEFINITIONS IMPLICIT TAGS ::=
BEGIN

****************** APDU Definitions ******************

BACnetPDU     ::= CHOICE {
        confirmed-request-PDU           [0] BACnet-Confirmed-Request-PDU,
        unconfirmed-request-PDU         [1] BACnet-Unconfirmed-Request-PDU,
        simpleACK-PDU                   [2] BACnet-SimpleACK-PDU,
        complexACK-PDU                  [3] BACnet-ComplexACK-PDU,
        segmentAck-PDU                  [4] BACnet-SegmentACK-PDU,
        error-PDU                       [5] BACnet-Error-PDU,
        reject-PDU                      [6] BACnet-Reject-PDU,
        abort-PDU                       [7] BACnet-Abort-PDU
        }

BACnet-Confirmed-Request-PDU ::= SEQUENCE {
        pdu-type                        [0] Unsigned (0..15), -- 0 for this PDU type
        segmented-message               [1] BOOLEAN,
        more-follows                    [2] BOOLEAN,
        segmented-response-accepted     [3] BOOLEAN,
        reserved                        [4] Unsigned (0..3), -- must be set to zero
        max-segments-accepted           [5] Unsigned (0..7),   -- as per 20.1.2.4
        max-APDU-length-accepted        [6] Unsigned (0..15), -- as per 20.1.2.5
        invokeID                        [7] Unsigned (0..255),
        sequence-number                 [8] Unsigned (0..255) OPTIONAL, -- only if segmented msg
        proposed-window-size            [9] Unsigned (1..127) OPTIONAL, -- only if segmented msg
        service-choice                  [10] BACnetConfirmedServiceChoice,
        service-request                 [11] BACnet-Confirmed-Service-Request OPTIONAL
-- Context-specific tags 0..11 are NOT used in header encoding
        }

BACnet-Unconfirmed-Request-PDU ::= SEQUENCE {
        pdu-type                [0] Unsigned (0..15), -- 1 for this PDU type
        reserved                [1] Unsigned (0..15), -- must be set to zero
        service-choice          [2] BACnetUnconfirmedServiceChoice,
        service-request         [3] BACnet-Unconfirmed-Service-Request
-- Context-specific tags 0..3 are NOT used in header encoding
        }

BACnet-SimpleACK-PDU ::= SEQUENCE {
        pdu-type                [0] Unsigned (0..15), -- 2 for this PDU type
        reserved                [1] Unsigned (0..15), -- must be set to zero
        invokeID                [2] Unsigned (0..255),
        service-ACK-choice      [3] BACnetConfirmedServiceChoice
-- Context-specific tags 0..3 are NOT used in header encoding
        }
```

```
BACnet-ComplexACK-PDU ::= SEQUENCE {
        pdu-type                [0] Unsigned (0..15), -- 3 for this PDU type
        segmented-message       [1] BOOLEAN,
        more-follows            [2] BOOLEAN,
        reserved                [3] Unsigned (0..3), -- must be set to zero
        invokeID                [4] Unsigned (0..255),
        sequence-number         [5] Unsigned (0..255) OPTIONAL, --only if segment
        proposed-window-size    [6] Unsigned (1..127) OPTIONAL, -- only if segment
        service-ACK-choice      [7] BACnetConfirmedServiceChoice,
        service-ACK             [8] BACnet-Confirmed-Service-ACK
-- Context-specific tags 0..8 are NOT used in header encoding
        }

BACnet-SegmentACK-PDU ::= SEQUENCE {
        pdu-type                [0] Unsigned (0..15), -- 4 for this PDU type
        reserved                [1] Unsigned (0..3), -- must be set to zero
        negative-ACK            [2] BOOLEAN,
        server                  [3] BOOLEAN,
        original-invokeID       [4] Unsigned (0..255),
        sequence-number         [5] Unsigned (0..255),
        actual-window-size      [6] Unsigned (1..127)
-- Context-specific tags 0..6 are NOT used in header encoding
        }

BACnet-Error-PDU ::= SEQUENCE {
        pdu-type                [0] Unsigned (0..15), -- 5 for this PDU type
        reserved                [1] Unsigned (0..15), -- must be set to zero
        original-invokeID       [2] Unsigned (0..255),
        error-choice            [3] BACnetConfirmedServiceChoice,
        error                   [4] BACnet-Error
-- Context-specific tags 0..4 are NOT used in header encoding
        }

BACnet-Reject-PDU ::= SEQUENCE {
        pdu-type                [0] Unsigned (0..15), -- 6 for this PDU type
        reserved                [1] Unsigned (0..15), -- must be set to zero
        original-invokeID       [2] Unsigned (0..255),
        reject-reason           [3] BACnetRejectReason
-- Context-specific tags 0..3 are NOT used in the header encoding
        }

BACnet-Abort-PDU ::= SEQUENCE {
        pdu-type                [0] Unsigned (0..15), -- 7 for this PDU type
        reserved                [1] Unsigned (0..7), -- must be set to zero
        server                  [2] BOOLEAN,
        original-invokeID       [3] Unsigned (0..255),
        abort-reason            [4] BACnetAbortReason
-- Context-specific tags 0..4 are NOT used in header encoding
        }
```

********** Confirmed Service Productions **********

BACnetConfirmedServiceChoice ::= ENUMERATED {
-- Alarm and Event Services
    acknowledgeAlarm (0),
    confirmedCOVNotification (1),
    confirmedEventNotification (2),
    getAlarmSummary (3),
    getEnrollmentSummary (4),
    getEventInformation (29),
    subscribeCOV (5),
    subscribeCOVProperty (28),
    lifeSafetyOperation (27),

-- File Access Services
    atomicReadFile (6),
    atomicWriteFile (7),

-- Object Access Services
    addListElement (8),
    removeListElement (9),
    createObject (10),
    deleteObject (11),
    readProperty (12),
    readPropertyConditional (13),
    readPropertyMultiple (14),
    readRange (26),
    writeProperty (15),
    writePropertyMultiple (16),

-- Remote Device Management Services
    deviceCommunicationControl (17),
    confirmedPrivateTransfer (18),
    confirmedTextMessage (19),
    reinitializeDevice (20),

-- Virtual Terminal Services
    vtOpen (21),
    vtClose (22),
    vtData (23),

-- Security Services
    authenticate (24),
    requestKey (25)

-- Services added after 1995
    -- readRange (26) see Object Access Services
    -- lifeSafetyOperation (27) see Alarm and Event Services
    -- subscribeCOVProperty (28) see Alarm and Event Services
    -- getEventInformation (29) see Alarm and Event Services
    }
-- Other services to be added as they are defined. All enumeration values in this production are reserved for definition by
-- ASHRAE. Proprietary extensions are made by using the ConfirmedPrivateTransfer or UnconfirmedPrivateTransfer
-- services. See Clause 23.

```
BACnet-Confirmed-Service-Request ::= CHOICE {
-- Alarm and Event Services
        acknowledgeAlarm            [0]   AcknowledgeAlarm-Request,
        confirmedCOVNotification    [1]   ConfirmedCOVNotification-Request,
        confirmedEventNotification  [2]   ConfirmedEventNotification-Request,
        -- getAlarmSummary conveys no parameters
        getEnrollmentSummary        [4]   GetEnrollmentSummary-Request,
        getEventInformation         [29]  GetEventInformation-Request,
        subscribeCOV                [5]   SubscribeCOV-Request,
        subscribeCOVProperty        [28]  SubscribeCOVProperty-Request,
        lifeSafetyOperation         [27]  LifeSafetyOperation-Request,

-- File Access Services
        atomicReadFile              [6]   AtomicReadFile-Request,
        atomicWriteFile             [7]   AtomicWriteFile-Request,

-- Object Access Services
        addListElement              [8]   AddListElement-Request,
        removeListElement           [9]   RemoveListElement-Request,
        createObject                [10]  CreateObject-Request,
        deleteObject                [11]  DeleteObject-Request,
        readProperty                [12]  ReadProperty-Request,
        readPropertyConditional     [13]  ReadPropertyConditional-Request,
        readPropertyMultiple        [14]  ReadPropertyMultiple-Request,
        readRange                   [26]  ReadRange-Request,
        writeProperty               [15]  WriteProperty-Request,
        writePropertyMultiple       [16]  WritePropertyMultiple-Request,

-- Remote Device Management Services
        deviceCommunicationControl  [17]  DeviceCommunicationControl-Request,
        confirmedPrivateTransfer    [18]  ConfirmedPrivateTransfer-Request,
        confirmedTextMessage        [19]  ConfirmedTextMessage-Request,
        reinitializeDevice          [20]  ReinitializeDevice-Request,

-- Virtual Terminal Services
        vtOpen                      [21]  VT-Open-Request,
        vtClose                     [22]  VT-Close-Request,
        vtData                      [23]  VT-Data-Request,

-- Security Services
        authenticate                [24]  Authenticate-Request,
        requestKey                  [25]  RequestKey-Request

-- Services added after 1995
        -- readRange                [26]  see Object Access Services
        -- lifeSafetyOperation      [27]  see Alarm and Event Services
        -- subscribeCOVProperty     [28]  see Alarm and Event Services
        -- getEventInformation      [29]  see Alarm and Event Services
        }
-- Context-specific tags 0..29 are NOT used in the encoding. The tag number is transferred as the service-choice parameter
-- in the BACnet-Confirmed-Request-PDU.
--
-- Other services will be added as they are defined. All choice values in this production are reserved for definition by
-- ASHRAE. Proprietary extensions are made by using the ConfirmedPrivateTransfer service. See Clause 23.
```

```
BACnet-Confirmed-Service-ACK  ::= CHOICE {
-- This production represents the 'Result(+)' parameters defined for each confirmed service that returns one or more
-- parameters with 'Result(+)'.

-- Alarm and Event Services
        getAlarmSummary                 [3]   GetAlarmSummary-ACK,
        getEnrollmentSummary            [4]   GetEnrollmentSummary-ACK,
        getEventInformation             [29]  GetEventInformation-ACK,

-- File Access Services
        atomicReadFile                  [6]   AtomicReadFile-ACK,
        atomicWriteFile                 [7]   AtomicWriteFile-ACK,

-- Object Access Services
        createObject                    [10]  CreateObject-ACK,
        readProperty                    [12]  ReadProperty-ACK,
        readPropertyConditional         [13]  ReadPropertyConditional-ACK,
        readPropertyMultiple            [14]  ReadPropertyMultiple-ACK,
        readRange                       [26]  ReadRange-ACK,

-- Remote Device Management Services
        confirmedPrivateTransfer        [18]  ConfirmedPrivateTransfer-ACK,

-- Virtual Terminal Services
        vtOpen                          [21]  VT-Open-ACK,
        vtData                          [23]  VT-Data-ACK,

-- Security Services
        authenticate                    [24]  Authenticate-ACK

-- Context-specific tags 3..29 are NOT used in the encoding. The tag number is transferred as the service-ACK-choice
-- parameter in the BACnet-ComplexACK-PDU.
--
-- Other services to be added as they are defined. All choice values in this production are reserved for definition by
-- ASHRAE. Proprietary extensions are made by using the ConfirmedPrivateTransfer service.
-- See Clause 23.
        }

********************** Confirmed Alarm and Event Services ******************

AcknowledgeAlarm-Request ::= SEQUENCE {
        acknowledgingProcessIdentifier  [0] Unsigned32,
        eventObjectIdentifier           [1] BACnetObjectIdentifier,
        eventStateAcknowledged          [2] BACnetEventState,
        timeStamp                       [3] BACnetTimeStamp,
        acknowledgmentSource            [4] CharacterString,
        timeOfAcknowledgment            [5] BACnetTimeStamp
        }

ConfirmedCOVNotification-Request ::= SEQUENCE {
        subscriberProcessIdentifier     [0] Unsigned32,
        initiatingDeviceIdentifier      [1] BACnetObjectIdentifier,
        monitoredObjectIdentifier       [2] BACnetObjectIdentifier,
        timeRemaining                   [3] Unsigned,
        listOfValues                    [4] SEQUENCE OF BACnetPropertyValue
        }
```

```
ConfirmedEventNotification-Request ::= SEQUENCE {
        processIdentifier               [0] Unsigned32,
        initiatingDeviceIdentifier      [1] BACnetObjectIdentifier,
        eventObjectIdentifier           [2] BACnetObjectIdentifier,
        timeStamp                       [3] BACnetTimeStamp,
        notificationClass               [4] Unsigned,
        priority                        [5] Unsigned8,
        eventType                       [6] BACnetEventType,
        messageText                     [7] CharacterString OPTIONAL,
        notifyType                      [8] BACnetNotifyType,
        ackRequired                     [9] BOOLEAN OPTIONAL,
        fromState                       [10] BACnetEventState OPTIONAL,
        toState                         [11] BACnetEventState,
        eventValues                     [12] BACnetNotificationParameters OPTIONAL
        }

GetAlarmSummary-ACK ::= SEQUENCE OF SEQUENCE {
        objectIdentifier            BACnetObjectIdentifier,
        alarmState                  BACnetEventState,
        acknowledgedTransitions     BACnetEventTransitionBits
        }

GetEnrollmentSummary-Request ::= SEQUENCE {
        acknowledgmentFilter            [0] ENUMERATED {
                                                all             (0),
                                                acked           (1),
                                                not-acked       (2)
                                                },
        enrollmentFilter                [1] BACnetRecipientProcess OPTIONAL,
        eventStateFilter                [2] ENUMERATED {
                                                offnormal       (0),
                                                fault           (1),
                                                normal          (2),
                                                all             (3),
                                                active          (4)
                                                } OPTIONAL,
        eventTypeFilter                 [3] BACnetEventType OPTIONAL,
        priorityFilter                  [4] SEQUENCE {
                                                minPriority     [0] Unsigned8,
                                                maxPriority     [1] Unsigned8
                                                } OPTIONAL,
        notificationClassFilter         [5] Unsigned OPTIONAL
        }

GetEnrollmentSummary-ACK ::= SEQUENCE OF SEQUENCE {
        objectIdentifier            BACnetObjectIdentifier,
        eventType                   BACnetEventType,
        eventState                  BACnetEventState,
        priority                    Unsigned8,
        notificationClass           Unsigned OPTIONAL
        }

GetEventInformation-Request ::= SEQUENCE {
        lastReceivedObjectIdentifier    [0]  BACnetObjectIdentifier OPTIONAL
        }
```

```
GetEventInformation-ACK ::= SEQUENCE {
        listOfEventSummaries    [0]  SEQUENCE OF SEQUENCE {
                objectIdentifier            [0]  BACnetObjectIdentifier,
                eventState                  [1]  BACnetEventState,
                acknowledgedTransitions     [2]  BACnetEventTransitionBits,
                eventTimeStamps             [3]  SEQUENCE SIZE (3) OF BACnetTimeStamp,
                notifyType                  [4]  BACnetNotifyType,
                eventEnable                 [5]  BACnetEventTransitionBits,
                eventPriorities             [6]  SEQUENCE SIZE (3) OF Unsigned
                },
        moreEvents              [1]  BOOLEAN
        }

LifeSafetyOperation-Request ::= SEQUENCE {
        requestingProcessIdentifier     [0]  Unsigned32,
        requestingSource                [1]  CharacterString,
        request                         [2]  BACnetLifeSafetyOperation,
        objectIdentifier                [3]  BACnetObjectIdentifier OPTIONAL
        }

SubscribeCOV-Request ::= SEQUENCE {
        subscriberProcessIdentifier     [0]  Unsigned32,
        monitoredObjectIdentifier       [1]  BACnetObjectIdentifier,
        issueConfirmedNotifications     [2]  BOOLEAN OPTIONAL,
        lifetime                        [3]  Unsigned OPTIONAL
        }

SubscribeCOVProperty-Request ::= SEQUENCE {
        subscriberProcessIdentifier     [0]  Unsigned32,
        monitoredObjectIdentifier       [1]  BACnetObjectIdentifier,
        issueConfirmedNotifications     [2]  BOOLEAN OPTIONAL,
        lifetime                        [3]  Unsigned OPTIONAL,
        monitoredPropertyIdentifier     [4]  BACnetPropertyReference,
        covIncrement                    [5]  REAL OPTIONAL
        }

************************** Confirmed File Access Services **************************

AtomicReadFile-Request ::= SEQUENCE {
        fileIdentifier      BACnetObjectIdentifier,
        accessMethod        CHOICE {
                                streamAccess    [0] SEQUENCE {
                                                        fileStartPosition       INTEGER,
                                                        requestedOctetCount     Unsigned
                                                        },
                                recordAccess    [1] SEQUENCE {
                                                        fileStartRecord         INTEGER,
                                                        requestedRecordCount    Unsigned
                                                        }
                                }
        }
```

```
AtomicReadFile-ACK ::= SEQUENCE {
        endOfFile       BOOLEAN,
        accessMethod    CHOICE {
                                streamAccess    [0] SEQUENCE {
                                                        fileStartPosition       INTEGER,
                                                        fileData                OCTET STRING
                                                        },
                                recordAccess    [1] SEQUENCE {
                                                        fileStartRecord         INTEGER,
                                                        returnedRecordCount     Unsigned,
                                                        fileRecordData          SEQUENCE OF OCTET STRING
                                                        }
                                }
        }

AtomicWriteFile-Request ::= SEQUENCE {
        fileIdentifier  BACnetObjectIdentifier,
        accessMethod    CHOICE {
                                streamAccess    [0] SEQUENCE {
                                                        fileStartPosition   INTEGER,
                                                        fileData            OCTET STRING
                                                        },
                                recordAccess    [1] SEQUENCE {
                                                        fileStartRecord     INTEGER,
                                                        recordCount         Unsigned,
                                                        fileRecordData      SEQUENCE OF OCTET STRING
                                                        }
                                }
        }

AtomicWriteFile-ACK ::= CHOICE {
        fileStartPosition   [0] INTEGER,
        fileStartRecord     [1] INTEGER
        }

************************** Confirmed Object Access Services ***********************

AddListElement-Request ::= SEQUENCE {
        objectIdentifier        [0] BACnetObjectIdentifier,
        propertyIdentifier      [1] BACnetPropertyIdentifier,
        propertyArrayIndex      [2] Unsigned OPTIONAL, -- used only with array datatype
        listOfElements          [3] ABSTRACT-SYNTAX.&Type
        }

CreateObject-Request ::= SEQUENCE {
        objectSpecifier         [0] CHOICE {
                                        objectType      [0] BACnetObjectType,
                                        objectIdentifier [1] BACnetObjectIdentifier
                                        },
        listOfInitialValues     [1] SEQUENCE OF BACnetPropertyValue OPTIONAL
        }

CreateObject-ACK ::= BACnetObjectIdentifier
```

```
DeleteObject-Request ::= SEQUENCE {
        objectIdentifier            BACnetObjectIdentifier
        }

ReadProperty-Request ::= SEQUENCE {
        objectIdentifier            [0] BACnetObjectIdentifier,
        propertyIdentifier          [1] BACnetPropertyIdentifier,
        propertyArrayIndex          [2] Unsigned OPTIONAL --used only with array datatype
                                                          -- if omitted with an array the entire array is referenced
        }

ReadProperty-ACK ::= SEQUENCE {
        objectIdentifier            [0] BACnetObjectIdentifier,
        propertyIdentifier          [1] BACnetPropertyIdentifier,
        propertyArrayIndex          [2] Unsigned OPTIONAL,--used only with array datatype
                                                          -- if omitted with an array the entire array is referenced
        propertyValue               [3] ABSTRACT-SYNTAX.&Type
        }

ReadPropertyConditional-Request ::= SEQUENCE {
        objectSelectionCriteria                 [0] SEQUENCE {
                        selectionLogic          [0] ENUMERATED {
                                                        and (0),
                                                        or (1),
                                                        all (2)
                                                        },
                        listOfSelectionCriteria [1] SEQUENCE OF SEQUENCE{
                                                        propertyIdentifier[0] BACnetPropertyIdentifier,
                                                        propertyArrayIndex      [1] Unsigned OPTIONAL,
                                                        relationSpecifier       [2] ENUMERATED {
                                                                equal                   (0),
                                                                not-equal               (1),
                                                                less-than               (2),
                                                                greater-than            (3),
                                                                less-than-or-equal      (4),
                                                                greater-than-or-equal   (5)
                                                                },
                                                        comparisonValue[3] ABSTRACT-SYNTAX.&Type
                                                        } OPTIONAL
                        },
        listOfPropertyReferences    [1] SEQUENCE OF BACnetPropertyReference OPTIONAL
        }

ReadPropertyConditional-ACK ::= SEQUENCE {
        listOfReadAccessResults     SEQUENCE OF ReadAccessResult OPTIONAL
        }

ReadPropertyMultiple-Request    ::= SEQUENCE {
        listOfReadAccessSpecs       SEQUENCE OF ReadAccessSpecification
        }

ReadPropertyMultiple-ACK ::= SEQUENCE {
        listOfReadAccessResults     SEQUENCE OF ReadAccessResult
        }
```

```
ReadRange-Request ::= SEQUENCE {
        objectIdentifier            [0]  BACnetObjectIdentifier,
        propertyIdentifier          [1]  BACnetPropertyIdentifier,
        propertyArrayIndex          [2]  Unsigned OPTIONAL          -- used only with array datatype
        range                        CHOICE {
                                        byPosition              [3]  SEQUENCE {
                                                                        referenceIndex  Unsigned,
                                                                        count           INTEGER
                                                                        },
                                        -- context tag 4 is deprecated
                                        -- context tag 5 is deprecated
                                        bySequenceNumber        [6]  SEQUENCE {
                                                                        referenceIndex  Unsigned,
                                                                        count           INTEGER
                                                                        },
                                        byTime                  [7]  SEQUENCE {
                                                                        referenceTime   BACnetDateTime,
                                                                        count           INTEGER
                                                                        }
                                        } OPTIONAL
        }

ReadRange-ACK ::= SEQUENCE {
        objectIdentifier            [0]  BACnetObjectIdentifier,
        propertyIdentifier          [1]  BACnetPropertyIdentifier,
        propertyArrayIndex          [2]  Unsigned OPTIONAL,  -- used only with array datatype
        resultFlags                 [3]  BACnetResultFlags,
        itemCount                   [4]  Unsigned,
        itemData                    [5]  SEQUENCE OF ABSTRACT-SYNTAX.&TYPE,
        firstSequenceNumber         [6] Unsigned32 OPTIONAL -- used only if 'Item Count' > 0 and the request was either of
                                                            -- type 'By Sequence Number' or 'By Time'
        }

RemoveListElement-Request ::= SEQUENCE {
        objectIdentifier            [0] BACnetObjectIdentifier,
        propertyIdentifier          [1] BACnetPropertyIdentifier,
        propertyArrayIndex          [2] Unsigned OPTIONAL, --used only with array datatypes
        listOfElements              [3] ABSTRACT-SYNTAX.&Type
        }

WriteProperty-Request ::= SEQUENCE {
                objectIdentifier            [0] BACnetObjectIdentifier,
                propertyIdentifier          [1] BACnetPropertyIdentifier,
                propertyArrayIndex          [2] Unsigned OPTIONAL, --used only with array datatype
                                                                   -- if omitted with an array the entire
                                                                   -- array is referenced
                propertyValue               [3] ABSTRACT-SYNTAX.&Type,
                priority                    [4] Unsigned8 (1..16) OPTIONAL --used only when property is commandable
        }

WritePropertyMultiple-Request ::= SEQUENCE {
        listOfwriteAccessSpecifications     SEQUENCE OF WriteAccessSpecification
        }
```

```
************************** Confirmed Remote Device Management Services ******************

DeviceCommunicationControl-Request ::= SEQUENCE {
        timeDuration        [0] Unsigned16 OPTIONAL,
        enable-disable      [1] ENUMERATED {
                                    enable              (0),
                                    disable             (1),
                                    disable-initiation  (2)
                                    },
        password            [2] CharacterString (SIZE(1..20)) OPTIONAL
        }

ConfirmedPrivateTransfer-Request ::= SEQUENCE {
        vendorID                [0] Unsigned,
        serviceNumber           [1] Unsigned,
        serviceParameters       [2] ABSTRACT-SYNTAX.&Type OPTIONAL
        }

ConfirmedPrivateTransfer-ACK ::= SEQUENCE {
        vendorID                [0] Unsigned,
        serviceNumber           [1] Unsigned,
        resultBlock             [2] ABSTRACT-SYNTAX.&Type OPTIONAL
        }

ConfirmedTextMessage-Request ::= SEQUENCE {
        textMessageSourceDevice     [0] BACnetObjectIdentifier,
        messageClass                [1] CHOICE {
                                            numeric     [0] Unsigned,
                                            character   [1] CharacterString
                                            } OPTIONAL,
        messagePriority             [2] ENUMERATED {
                                            normal      (0),
                                            urgent      (1)
                                            },
        message                     [3] CharacterString
        }

ReinitializeDevice-Request ::= SEQUENCE {
        reinitializedStateOfDevice      [0] ENUMERATED {
                                                coldstart       (0),
                                                warmstart       (1),
                                                startbackup     (2),
                                                endbackup       (3),
                                                startrestore    (4),
                                                endrestore      (5),
                                                abortrestore    (6)
                                                },
        password                        [1] CharacterString (SIZE (1..20)) OPTIONAL
        }

********************** Confirmed Virtual Terminal Services **********************

VT-Open-Request ::= SEQUENCE {
        vtClass                 BACnetVTClass,
        localVTSessionIdentifier    Unsigned8
        }
```

```
VT-Open-ACK ::= SEQUENCE {
        remoteVTSessionIdentifier       Unsigned8
        }

VT-Close-Request ::= SEQUENCE {
        listOfRemoteVTSessionIdentifiers  SEQUENCE OF Unsigned8
        }

VT-Data-Request ::= SEQUENCE {
        vtSessionIdentifier     Unsigned8,
        vtNewData               OCTET STRING,
        vtDataFlag              Unsigned (0..1)
        }

VT-Data-ACK ::= SEQUENCE {
        allNewDataAccepted      [0] BOOLEAN,
        acceptedOctetCount      [1] Unsigned OPTIONAL --present only if allNewDataAccepted = FALSE
        }

********** Confirmed Security Services **********

Authenticate-Request ::= SEQUENCE {
        pseudoRandomNumber      [0] Unsigned32,
        expectedInvokeID        [1] Unsigned8 OPTIONAL,
        operatorName            [2] CharacterString OPTIONAL,
        operatorPassword        [3] CharacterString (SIZE (1..20)) OPTIONAL,
        startEncipheredSession  [4] BOOLEAN OPTIONAL
        }

Authenticate-ACK ::= SEQUENCE {
        modifiedRandomNumber    Unsigned32
        }

RequestKey-Request ::= SEQUENCE {
        requestingDeviceIdentifier      BACnetObjectIdentifier,
        requestingDeviceAddress         BACnetAddress,
        remoteDeviceIdentifier          BACnetObjectIdentifier,
        remoteDeviceAddress             BACnetAddress
        }

********** Unconfirmed Request Productions **********

BACnetUnconfirmedServiceChoice ::= ENUMERATED {
        i-Am                            (0),
        i-Have                          (1),
        unconfirmedCOVNotification      (2),
        unconfirmedEventNotification    (3),
        unconfirmedPrivateTransfer      (4),
        unconfirmedTextMessage          (5),
        timeSynchronization             (6),
        who-Has                         (7),
        who-Is                          (8),
        utcTimeSynchronization          (9)
        }
-- Other services to be added as they are defined. All choice values in this production are reserved for definition by
```

```
-- ASHRAE. Proprietary extensions are made by using the UnconfirmedPrivateTransfer service. See Clause 23.
BACnet-Unconfirmed-Service-Request ::= CHOICE {
        i-Am                            [0] I-Am-Request,
        i-Have                          [1] I-Have-Request,
        unconfirmedCOVNotification      [2] UnconfirmedCOVNotification-Request,
        unconfirmedEventNotification    [3] UnconfirmedEventNotification-Request,
        unconfirmedPrivateTransfer      [4] UnconfirmedPrivateTransfer-Request,
        unconfirmedTextMessage          [5] UnconfirmedTextMessage-Request,
        timeSynchronization             [6] TimeSynchronization-Request,
        who-Has                         [7] Who-Has-Request,
        who-Is                          [8] Who-Is-Request,
        utcTimeSynchronization          [9] UTCTimeSynchronization-Request
        }
-- Context-specific tags 0..8 are NOT used in the encoding. The tag number is transferred as the service-choice parameter
-- in the BACnet-Unconfirmed-Request-PDU.
--
-- Other services to be added as they are defined. All choice values in this production are reserved for definition by
-- ASHRAE. Proprietary extensions are made by using the UnconfirmedPrivateTransfer service.
-- See Clause 23.

******************** Unconfirmed Alarm and Event Services **********************

UnconfirmedCOVNotification-Request ::= SEQUENCE {
        subscriberProcessIdentifier     [0] Unsigned32,
        initiatingDeviceIdentifier      [1] BACnetObjectIdentifier,
        monitoredObjectIdentifier       [2] BACnetObjectIdentifier,
        timeRemaining                   [3] Unsigned,
        listOfValues                    [4] SEQUENCE OF BACnetPropertyValue
        }

UnconfirmedEventNotification-Request ::= SEQUENCE {
        processIdentifier               [0] Unsigned32,
        initiatingDeviceIdentifier      [1] BACnetObjectIdentifier,
        eventObjectIdentifier           [2] BACnetObjectIdentifier,
        timeStamp                       [3] BACnetTimeStamp,
        notificationClass               [4] Unsigned,
        priority                        [5] Unsigned8,
        eventType                       [6] BACnetEventType,
        messageText                     [7] CharacterString OPTIONAL,
        notifyType                      [8] BACnetNotifyType,
        ackRequired                     [9] BOOLEAN OPTIONAL,
        fromState                       [10] BACnetEventState OPTIONAL,
        toState                         [11] BACnetEventState,
        eventValues                     [12] BACnetNotificationParameters OPTIONAL
        }

******************** Unconfirmed Remote Device Management Services **********************

I-Am-Request ::= SEQUENCE {
        iAmDeviceIdentifier             BACnetObjectIdentifier,
        maxAPDULengthAccepted           Unsigned,
        segmentationSupported           BACnetSegmentation,
        vendorID                        Unsigned
        }
```

```
I-Have-Request ::= SEQUENCE {
        deviceIdentifier    BACnetObjectIdentifier,
        objectIdentifier    BACnetObjectIdentifier,
        objectName          CharacterString
        }

UnconfirmedPrivateTransfer-Request ::= SEQUENCE {
        vendorID                [0] Unsigned,
        serviceNumber           [1] Unsigned,
        serviceParameters       [2] ABSTRACT-SYNTAX.&Type OPTIONAL
        }

UnconfirmedTextMessage-Request ::= SEQUENCE {
        textMessageSourceDevice     [0] BACnetObjectIdentifier,
        messageClass                [1] CHOICE {
                                            numeric     [0] Unsigned,
                                            character   [1] CharacterString
                                            } OPTIONAL,
        messagePriority             [2] ENUMERATED {
                                            normal      (0),
                                            urgent      (1)
                                            },
        message                     [3] CharacterString
        }

TimeSynchronization-Request ::= SEQUENCE {
        time    BACnetDateTime
        }

UTCTimeSynchronization-Request ::= SEQUENCE {
        time    BACnetDateTime
        }

Who-Has-Request ::= SEQUENCE {
        limits  SEQUENCE {
                    deviceInstanceRangeLowLimit     [0] Unsigned (0..4194303),
                    deviceInstanceRangeHighLimit    [1] Unsigned (0..4194303)
                    } OPTIONAL,
        object  CHOICE {
                    objectIdentifier                [2] BACnetObjectIdentifier,
                    objectName                      [3] CharacterString
                    }
        }

Who-Is-Request ::= SEQUENCE {
        deviceInstanceRangeLowLimit     [0] Unsigned (0..4194303) OPTIONAL, -- must be used as a pair, see 16.10
        deviceInstanceRangeHighLimit    [1] Unsigned (0..4194303) OPTIONAL  -- must be used as a pair, see 16.10
        }
```

```
****************************** Error Productions ************************

BACnetAbortReason ::= ENUMERATED {
        other                                   (0),
        buffer-overflow                         (1),
        invalid-apdu-in-this-state              (2),
        preempted-by-higher-priority-task       (3),
        segmentation-not-supported              (4),
        ...
        }
        -- Enumerated values 0-63 are reserved for definition by ASHRAE. Enumerated values 64-255
        -- may be used by others subject to the procedures and constraints described in Clause 23.

BACnet-Error  ::= CHOICE {
        other                                   [127] Error,
-- The remaining choices in this production represent the Result(-) parameters
-- defined for each confirmed service.

-- Alarm and Event Services
        acknowledgeAlarm                        [0]   Error,
        confirmedCOVNotification                [1]   Error,
        confirmedEventNotification              [2]   Error,
        getAlarmSummary                         [3]   Error,
        getEnrollmentSummary                    [4]   Error,
        getEventInformation                     [29]  Error,
        subscribeCOV                            [5]   Error,
        subscribeCOVProperty                    [28]  Error,
        lifeSafetyOperation                     [27]  Error,

-- File Access Services
        atomicReadFile                          [6]   Error,
        atomicWriteFile                         [7]   Error,

-- Object Access Services
        addListElement                          [8]   ChangeList-Error,
        removeListElement                       [9]   ChangeList-Error,
        createObject                            [10]  CreateObject-Error,
        deleteObject                            [11]  Error,
        readProperty                            [12]  Error,
        readPropertyConditional                 [13]  Error,
        readPropertyMultiple                    [14]  Error,
        readRange                               [26]  Error,
        writeProperty                           [15]  Error,
        writePropertyMultiple                   [16]  WritePropertyMultiple-Error,

-- Remote Device Management Services
        deviceCommunicationControl              [17] Error,
        confirmedPrivateTransfer                [18] ConfirmedPrivateTransfer-Error,
        confirmedTextMessage                    [19] Error,
        reinitializeDevice                      [20] Error,

-- Virtual Terminal Services
        vtOpen                                  [21] Error,
        vtClose                                 [22] VTClose-Error,
        vtData                                  [23] Error,
```

```
-- Security Services
        authenticate                    [24] Error,
        requestKey                      [25] Error

-- Services added after 1995
        -- readRange                    [26]  see Object Access  Services
        -- lifeSafetyOperation          [27]  see Alarm and Event Services
        -- subscribeCOVProperty         [28]  see Alarm and Event Services
        -- getEventInformation          [29]  see Alarm and Event Services
        }
-- Context-specific tags 0..29 and 127 are NOT used in the encoding. The tag number is transferred as the error-choice
-- parameter in the BACnet-Error-PDU.
--
-- Other services to be added as they are defined. All choice values in this production are reserved for definition by
-- ASHRAE. See Clause 23.

BACnetRejectReason ::= ENUMERATED {
        other                           (0),
        buffer-overflow                 (1),
        inconsistent-parameters         (2),
        invalid-parameter-data-type     (3),
        invalid-tag                     (4),
        missing-required-parameter      (5),
        parameter-out-of-range          (6),
        too-many-arguments              (7),
        undefined-enumeration           (8),
        unrecognized-service            (9),
        ...
        }
-- Enumerated values 0-63 are reserved for definition by ASHRAE. Enumerated values 64-65535
-- may be used by others subject to the procedures and constraints described in Clause 23.

ChangeList-Error ::= SEQUENCE {
        errorType                       [0] Error,
        firstFailedElementNumber        [1] Unsigned
        }

CreateObject-Error ::= SEQUENCE {
        errorType                       [0] Error,
        firstFailedElementNumber        [1] Unsigned
        }

ConfirmedPrivateTransfer-Error ::= SEQUENCE {
        errorType           [0] Error,
        vendorID            [1] Unsigned,
        serviceNumber       [2] Unsigned,
        errorParameters     [3] ABSTRACT-SYNTAX.&Type OPTIONAL
        }
```

```
Error ::= SEQUENCE {
-- NOTE: The valid combinations of error-class and error-code are defined in Clause 18.
        error-class         ENUMERATED {
                            device              (0),
                            object              (1),
                            property            (2),
                            resources           (3),
                            security            (4),
                            services            (5),
                            vt                  (6),
                            ...
                            },
        -- Enumerated values 0-63 are reserved for definition by ASHRAE. Enumerated values
        -- 64-65535 may be used by others subject to the procedures and constraints described
        -- in Clause 23.

        error-code          ENUMERATED {
                            other                               (0),
                            authentication-failed               (1),
                            character-set-not-supported         (41),
                            configuration-in-progress           (2),
                            datatype-not-supported              (47),
                            device-busy                         (3),
                            duplicate-name                      (48),
                            duplicate-object-id                 (49),
                            dynamic-creation-not-supported      (4),
                            file-access-denied                  (5),
                            incompatible-security-levels        (6),
                            inconsistent-parameters             (7),
                            inconsistent-selection-criterion    (8),
                            invalid-array-index                 (42),
                            invalid-configuration-data          (46),
                            invalid-data-type                   (9),
                            invalid-file-access-method          (10),
                            invalid-file-start-position         (11),
                            invalid-operator-name               (12),
                            invalid-parameter-data-type         (13),
                            invalid-time-stamp                  (14),
                            key-generation-error                (15),
                            missing-required-parameter          (16),
                            no-objects-of-specified-type        (17),
                            no-space-for-object                 (18),
                            no-space-to-add-list-element        (19),
                            no-space-to-write-property          (20),
                            no-vt-sessions-available            (21),
                            object-deletion-not-permitted       (23),
                            object-identifier-already-exists    (24),
                            operational-problem                 (25),
                            optional-functionality-not-supported, (45),
                            password-failure                    (26),
                            property-is-not-a-list              (22),
                            property-is-not-an-array            (50),
                            read-access-denied                  (27),
                            security-not-supported              (28),
                            service-request-denied              (29),
                            timeout                             (30),
```

```
                unknown-object                          (31),
                unknown-property                        (32),
            -- this enumeration was removed              (33),
                unknown-vt-class                        (34),
                unknown-vt-session                      (35),
                unsupported-object-type                 (36),
                value-out-of-range                      (37),
                vt-session-already-closed               (38),
                vt-session-termination-failure          (39),
                write-access-denied                     (40),
            -- see  character-set-not-supported         (41),
            -- see  invalid-array-index                 (42),
                cov-subscription-failed                 (43),
                not-cov-property                        (44),
            -- see  optional-functionality-not-supported, (45),
            -- see  invalid-configuration-data          (46),
            -- see datatype-not-supported               (47),
            -- see duplicate-name                       (48),
            -- see duplicate-object-id                  (49),
            -- see property-is-not-an-array             (50),
                ...
        }
        -- Enumerated values 0-255 are reserved for definition by ASHRAE. Enumerated values
        -- 256-65535 may be used by others subject to the procedures and constraints described
        -- in Clause 23. The last enumeration used in this version is 47.
    }

WritePropertyMultiple-Error ::= SEQUENCE {
        errorType                       [0] Error,
        firstFailedWriteAttempt         [1] BACnetObjectPropertyReference
        }

VTClose-Error ::= SEQUENCE {
        errorType                       [0] Error,
        listOfVTSessionIdentifiers      [1] SEQUENCE OF Unsigned8 OPTIONAL
}

********************************** Application Types **********************************

-- The following productions are the definitions of the Application datatypes.
-- See 20.2.1.4.

-- NULL                 [APPLICATION 0], equivalent to [UNIVERSAL 5]

-- BOOLEAN              [APPLICATION 1], equivalent to [UNIVERSAL 1]

  Unsigned ::=          [APPLICATION 2] INTEGER (0..MAX)

  Unsigned8  ::=        Unsigned (0..255)

  Unsigned16  ::=       Unsigned (0..65535)

  Unsigned32  ::=       Unsigned (0..4294967295)

-- INTEGER              [APPLICATION 3], equivalent to [UNIVERSAL 2]
```

```
-- REAL             [APPLICATION 4], equivalent to [UNIVERSAL 9] ANSI/IEEE-754 single precision floating point

 Double   ::=       [APPLICATION 5] OCTET STRING (SIZE(8)) -- ANSI/IEEE-754 double precision floating point

-- OCTET STRING     [APPLICATION 6], equivalent to [UNIVERSAL 4]

 CharacterString ::= [APPLICATION 7] OCTET STRING -- see 20.2.9 for supported types

-- BIT STRING       [APPLICATION 8], equivalent to [UNIVERSAL 3]

-- ENUMERATED       [APPLICATION 9], equivalent to [UNIVERSAL 10]

Date ::= [APPLICATION 10] OCTET STRING (SIZE(4)) -- see 20.2.12
--       first octet     year minus 1900          X'FF' = unspecified
--       second octet    month (1.. 14)           1   = January
--                                                13 = odd months
--                                                14 = even months
--                                                X'FF' = unspecified
--       third octet     day of month    (1..32), 32 = last day of month
--                                                X'FF' = unspecified
--       fourth octet    day of week     (1..7)   1 = Monday
--                                                7 = Sunday
--                                                X'FF' = unspecified

 Time  ::= [APPLICATION 11] OCTET STRING (SIZE(4)) -- see 20.2.13
--       first octet     hour            (0..23), (X'FF') = unspecified
--       second octet    minute          (0..59), (X'FF') = unspecified
--       third octet     second          (0..59), (X'FF') = unspecified
--       fourth octet    hundredths      (0..99), (X'FF') = unspecified

BACnetObjectIdentifier := [APPLICATION 12] OCTET STRING (SIZE(4)) -- see 20.2.14


****************************** Base Types *********************************

BACnetAccumulatorRecord ::= SEQUENCE {
        timestamp               [0] BACnetDateTime,
        presentValue            [1] Unsigned,
        accumulatedValue        [2] Unsigned,
        accumulatorStatus       [3] ENUMERATED {
                                        normal          (0),
                                        starting        (1),
                                        recovered       (2),
                                        abnormal        (3),
                                        failed          (4)
                                        }
        }

BACnetAction ::= ENUMERATED {
        direct          (0),
        reverse         (1)
        }

BACnetActionCommand ::= SEQUENCE {
        deviceIdentifier        [0] BACnetObjectIdentifier OPTIONAL,
        objectIdentifier        [1] BACnetObjectIdentifier,
```

```
        propertyIdentifier          [2] BACnetPropertyIdentifier,
        propertyArrayIndex          [3] Unsigned OPTIONAL,   --used only with array datatype
        propertyValue               [4] ABSTRACT-SYNTAX.&Type,
        priority                    [5] Unsigned (1..16) OPTIONAL, --used only when property is commandable
        postDelay                   [6] Unsigned OPTIONAL,
        quitOnFailure               [7] BOOLEAN,
        writeSuccessful             [8] BOOLEAN
        }

BACnetActionList ::= SEQUENCE{
        action    [0] SEQUENCE OF BACnetActionCommand
        }

BACnetAddress ::= SEQUENCE {
        network-number  Unsigned16, -- A value of 0 indicates the local network
        mac-address     OCTET STRING -- A string of length 0 indicates a broadcast
        }

BACnetAddressBinding ::= SEQUENCE {
        deviceObjectIdentifier      BACnetObjectIdentifier,
        deviceAddress               BACnetAddress
        }

BACnetBinaryPV ::= ENUMERATED {
        inactive            (0),
        active              (1)
        }

BACnetCalendarEntry ::= CHOICE {
        date                [0] Date,
        dateRange           [1] BACnetDateRange,
        weekNDay            [2] BACnetWeekNDay
        }

BACnetClientCOV ::= CHOICE {
        real-increment      REAL,
        default-increment   NULL
        }

BACnetCOVSubscription ::= SEQUENCE {
        Recipient                   [0] BACnetRecipientProcess,
        MonitoredPropertyReference  [1] BACnetObjectPropertyReference,
        IssueConfirmedNotifications [2] BOOLEAN,
        TimeRemaining               [3] Unsigned,
        COVIncrement                [4] REAL OPTIONAL       -- used only with monitored
                                                            -- properties with a datatype of REAL
        }

BACnetDailySchedule ::= SEQUENCE {
        day-schedule        [0] SEQUENCE OF BACnetTimeValue
        }

BACnetDateRange ::= SEQUENCE {
        StartDate           Date,
        endDate             Date
        }
```

```
BACnetDateTime ::= SEQUENCE {
        date            Date,
        time            Time
        }

BACnetDaysOfWeek ::= BIT STRING {
        monday          (0),
        tuesday         (1),
        wednesday       (2),
        thursday        (3),
        friday          (4),
        saturday        (5),
        sunday          (6)
        }

BACnetDestination ::= SEQUENCE {
        validDays                       BACnetDaysOfWeek,
        fromTime                        Time,
        toTime                          Time,
        recipient                       BACnetRecipient,
        processIdentifier               Unsigned32,
        issueConfirmedNotifications     BOOLEAN,
        transitions                     BACnetEventTransitionBits
        }

BACnetDeviceObjectPropertyReference ::= SEQUENCE {
        objectIdentifier        [0]     BACnetObjectIdentifier,
        propertyIdentifier      [1]     BACnetPropertyIdentifier,
        propertyArrayIndex      [2]     Unsigned OPTIONAL,      -- used only with array datatype
                                                                -- if omitted with an array then
                                                                -- the entire array is referenced
        deviceIdentifier        [3]     BACnetObjectIdentifier OPTIONAL
        }

BACnetDeviceObjectPropertyValue ::= SEQUENCE {
        deviceIdentifier        [0]     BACnetObjectIdentifier,
        objectIdentifier        [1]     BACnetObjectIdentifier,
        propertyIdentifier      [2]     BACnetPropertyIdentifier,
        arrayIndex              [3]     Unsigned – OPTIONAL,
        value                   [4]     ABSTRACT-SYNTAX.&Type
        }

BACnetDeviceObjectReference ::= SEQUENCE {
        deviceIdentifier        [0]     BACnetObjectIdentifier OPTIONAL,
        objectIdentifier        [1]     BACnetObjectIdentifier
        }
```

```
BACnetDeviceStatus ::= ENUMERATED {
        operational             (0),
        operational-read-only   (1),
        download-required       (2),
        download-in-progress    (3),
        non-operational         (4),
        backup-in-progress      (5),
        ...
        }
-- Enumerated values 0-63 are reserved for definition by ASHRAE. Enumerated values
-- 64-65535 may be used by others subject to the procedures and constraints described
-- in Clause 23.

BACnetEngineeringUnits ::= ENUMERATED {
-- Acceleration
        meters-per-second-per-second    (166),
--Area
        square-meters                   (0),
        square-centimeters              (116),
        square-feet                     (1),
        square-inches                   (115),

--Currency
        currency1                       (105),
        currency2                       (106),
        currency3                       (107),
        currency4                       (108),
        currency5                       (109),
        currency6                       (110),
        currency7                       (111),
        currency8                       (112),
        currency9                       (113),
        currency10                      (114),

--Electrical
        milliamperes                    (2),
        amperes                         (3),
        amperes-per-meter               (167),
        amperes-per-square-meter        (168),
        ampere-square-meters            (169),
        farads                          (170),
        henrys                          (171),
        ohms                            (4),
        ohm-meters                      (172),
        milliohms                       (145),
        kilohms                         (122),
        megohms                         (123),
        siemens                         (173),  -- 1 mho equals 1 siemens
        siemens-per-meter               (174),
        teslas                          (175),
        volts                           (5),
        millivolts                      (124),
        kilovolts                       (6),
        megavolts                       (7),
        volt-amperes                    (8),
        kilovolt-amperes                (9),
```

```
        megavolt-amperes                    (10),
        volt-amperes-reactive               (11),
        kilovolt-amperes-reactive           (12),
        megavolt-amperes-reactive           (13),
        volts-per-degree-Kelvin             (176),
        volts-per-meter                     (177),
        degrees-phase                       (14),
        power-factor                        (15),
        webers                              (178),

--Energy
        joules                              (16),
        kilojoules                          (17),
        kilojoules-per-kilogram             (125),
        megajoules                          (126),
        watt-hours                          (18),
        kilowatt-hours                      (19),
        megawatt-hours                      (146),
        btus                                (20),
        kilo-btus                           (147),
        mega-btus                           (148),
        therms                              (21),
        ton-hours                           (22),

--Enthalpy
        joules-per-kilogram-dry-air         (23),
        kilojoules-per-kilogram-dry-air     (149),
        megajoules-per-kilogram-dry-air     (150),
        btus-per-pound-dry-air              (24),
        btus-per-pound                      (117),

--Entropy
        joules-per-degree-Kelvin            (127),
        kilojoules-per-degree-Kelvin        (151),
        megajoules-per-degree-Kelvin        (152),
        joules-per-kilogram-degree-Kelvin   (128),

-- Force
        newton                              (153),

--Frequency
        cycles-per-hour                     (25),
        cycles-per-minute                   (26),
        hertz                               (27),
        kilohertz                           (129),
        megahertz                           (130),
        per-hour                            (131),

--Humidity
        grams-of-water-per-kilogram-dry-air (28),
        percent-relative-humidity           (29),

--Length
        millimeters                         (30),
        centimeters                         (118),
        meters                              (31),
```

```
        inches                          (32),
        feet                            (33),

--Light
        candelas                        (179),
        candelas-per-square-meter       (180),
        watts-per-square-foot           (34),
        watts-per-square-meter          (35),
        lumens                          (36),
        luxes                           (37),
        foot-candles                    (38),

--Mass
        kilograms                       (39),
        pounds-mass                     (40),
        tons                            (41),

--Mass Flow
        grams-per-second                (154),
        grams-per-minute                (155),
        kilograms-per-second            (42),
        kilograms-per-minute            (43),
        kilograms-per-hour              (44),
        pounds-mass-per-second          (119),
        pounds-mass-per-minute          (45),
        pounds-mass-per-hour            (46),
        tons-per-hour                   (156),

--Power
        milliwatts                      (132),
        watts                           (47),
        kilowatts                       (48),
        megawatts                       (49),
        btus-per-hour                   (50),
        kilo-btus-per-hour              (157),
        horsepower                      (51),
        tons-refrigeration              (52),

--Pressure
        pascals                         (53),
        hectopascals                    (133),
        kilopascals                     (54),
        millibars                       (134),
        bars                            (55),
        pounds-force-per-square-inch    (56),
        centimeters-of-water            (57),
        inches-of-water                 (58),
        millimeters-of-mercury          (59),
        centimeters-of-mercury          (60),
        inches-of-mercury               (61),

--Temperature
        degrees-Celsius                 (62),
        degrees-Kelvin                  (63),
        degrees-Kelvin-per-hour         (181),
        degrees-Kelvin-per-minute       (182),
```

```
            degrees-Fahrenheit              (64),
            degree-days-Celsius             (65),
            degree-days-Fahrenheit          (66),
            delta-degrees-Fahrenheit        (120),
            delta-degrees-Kelvin            (121),

--Time
            years                           (67),
            months                          (68),
            weeks                           (69),
            days                            (70),
            hours                           (71),
            minutes                         (72),
            seconds                         (73),
            hundredths-seconds              (158),
            milliseconds                    (159),

--Torque
            newton-meters                   (160),

--Velocity
            millimeters-per-second          (161),
            millimeters-per-minute          (162),
            meters-per-second               (74),
            meters-per-minute               (163),
            meters-per-hour                 (164),
            kilometers-per-hour             (75),
            feet-per-second                 (76),
            feet-per-minute                 (77),
            miles-per-hour                  (78),
--Volume
            cubic-feet                      (79),
            cubic-meters                    (80),
            imperial-gallons                (81),
            liters                          (82),
            us-gallons                      (83),

--Volumetric Flow
            cubic-feet-per-second           (142),
            cubic-feet-per-minute           (84),
            cubic-meters-per-second         (85),
            cubic-meters-per-minute         (165),
            cubic-meters-per-hour           (135),
            imperial-gallons-per-minute     (86),
            liters-per-second               (87),
            liters-per-minute               (88),
            liters-per-hour                 (136),
            us-gallons-per-minute           (89),

--Other
            degrees-angular                 (90),
            degrees-Celsius-per-hour        (91),
            degrees-Celsius-per-minute      (92),
            degrees-Fahrenheit-per-hour     (93),
            degrees-Fahrenheit-per-minute   (94),
            joule-seconds                   (183),
```

```
        kilograms-per-cubic-meter           (186),
        kilowatt-hours-per-square-meter     (137),
        kilowatt-hours-per-square-foot      (138),
        megajoules-per-square-meter         (139),
        megajoules-per-square-foot          (140),
        no-units                            (95),
        newton-seconds                      (187),
        newtons-per-meter                   (188),
        parts-per-million                   (96),
        parts-per-billion                   (97),
        percent                             (98),
        percent-obscuration-per-foot        (143),
        percent-obscuration-per-meter       (144),
        percent-per-second                  (99),
        per-minute                          (100),
        per-second                          (101),
        psi-per-degree-Fahrenheit           (102),
        radians                             (103),
        radians-per-second                  (184),
        revolutions-per-minute              (104),
        square-meters-per-Newton            (185),
        watts-per-meter-per-degree-Kelvin   (189),
        watts-per-square-meter-degree-kelvin (141),
        ...
        }
-- Enumerated values 0-255 are reserved for definition by ASHRAE. Enumerated values
-- 256-65535 may be used by others subject to the procedures and constraints described
-- in Clause 23. The last enumeration used in this version is 189.

BACnetEventParameter ::= CHOICE {

-- These choices have a one-to-one correspondence with the Event_Type enumeration with the  exception of the
-- complex-event-type, which is used for proprietary event types.

        change-of-bitstring     [0] SEQUENCE {
                                    time-delay              [0] Unsigned,
                                    bitmask                 [1] BIT STRING,
                                    list-of-bitstring-values [2] SEQUENCE OF BIT STRING
                                    },
        change-of-state         [1] SEQUENCE {
                                    time-delay      [0] Unsigned,
                                    list-of-values  [1] SEQUENCE OF BACnetPropertyStates
                                    },
        change-of-value         [2] SEQUENCE {
                                    time-delay      [0] Unsigned,
                                    cov-criteria    [1] CHOICE {
                                                        bitmask                         [0] BIT STRING,
                                                        referenced-property-increment   [1] REAL
                                                        }
                                    },
        command-failure         [3] SEQUENCE {
                                    time-delay                  [0] Unsigned,
                                    feedback-property-reference [1] BACnetDeviceObjectPropertyReference
                                    },
```

```
        floating-limit          [4] SEQUENCE {
                                    time-delay                  [0] Unsigned,
                                    setpoint-reference          [1] BACnetDeviceObjectPropertyReference,
                                    low-diff-limit              [2] REAL,
                                    high-diff-limit             [3] REAL,
                                    deadband                    [4] REAL
                                    },
        out-of-range            [5] SEQUENCE {
                                    time-delay          [0] Unsigned,
                                    low-limit           [1] REAL,
                                    high-limit          [2] REAL,
                                    deadband            [3] REAL
                                    },
        -- context tag 7 is deprecated
        change-of-life-safety   [8] SEQUENCE {
                                    time-delay                      [0] Unsigned,
                                    list-of-life-safety-alarm-values [1] SEQUENCE OF BACnetLifeSafetyState,
                                    list-of-alarm-values            [2] SEQUENCE OF BACnetLifeSafetyState,
                                    mode-property-reference         [3] BACnetDeviceObjectPropertyReference
                                    },
        extended                [9] SEQUENCE {
                                    vendorId            [0] Unsigned,
                                    extendedEventType   [1] Unsigned,
                                    parameters          [2] SEQUENCE OF CHOICE {
                                                            null        NULL,
                                                            real        REAL,
                                                            integer     Unsigned,
                                                            boolean     BOOLEAN,
                                                            double      DOUBLE,
                                                            octet       OCTET STRING,
                                                            bitstring   BIT STRING,
                                                            enum        ENUMERATED,
                                                            reference   [0] BACnetDeviceObjectProperty
                                                            }
                                    },
        buffer-ready            [10] SEQUENCE {
                                    notification-threshold          [0] Unsigned,
                                    previous-notification-count     [1] Unsigned32
                                    },
        unsigned-range          [11] SEQUENCE {
                                    time-delay          [0] Unsigned,
                                    low-limit           [1] Unsigned,
                                    high-limit          [2] Unsigned
                                    }
    }

-- CHOICE [6] has been intentionally omitted. It parallels the complex-event-type CHOICE [6] of the
-- BACnetNotificationParameters production which was introduced to allow the addition of proprietary event
-- algorithms whose event parameters are not necessarily network-visible.
```

```
BACnetEventState ::= ENUMERATED {
        normal          (0),
        fault           (1),
        offnormal       (2),
        high-limit      (3),
        low-limit       (4),
        life-safety-alarm (5),
        ...
        }
-- Enumerated values 0-63 are reserved for definition by ASHRAE. Enumerated values
-- 64-65535 may be used by others subject to the procedures and constraints described
-- in Clause 23. The last enumeration used in this version is 5.

BACnetEventTransitionBits ::= BIT STRING {
        to-offnormal    (0),
        to-fault        (1),
        to-normal       (2)
        }

BACnetEventType ::= ENUMERATED {
        change-of-bitstring     (0),
        change-of-state         (1),
        change-of-value         (2),
        command-failure         (3),
        floating-limit          (4),
        out-of-range            (5),
        -- complex-event-type   (6), -- see comment below
        -- context tag 7 is deprecated
        change-of-life-safety   (8),
        extended                (9),
        buffer-ready            (10),
        unsigned-range          (11),
        ...
        }
-- Enumerated values 0-63 are reserved for definition by ASHRAE. Enumerated values
-- 64-65535 may be used by others subject to the procedures and constraints described
-- in Clause 23. It is expected that these enumerated values will correspond to the use of the
-- complex-event-type CHOICE [6] of the BACnetNotificationParameters production.
-- The last enumeration used in this version is 11.

BACnetFileAccessMethod ::= ENUMERATED {
        record-access           (0),
        stream-access           (1)
        }

BACnetLifeSafetyMode ::= ENUMERATED {
        off                     (0),
        on                      (1),
        test                    (2),
        manned                  (3),
        unmanned                (4),
        armed                   (5),
        disarmed                (6),
        prearmed                (7),
        slow                    (8),
        fast                    (9),
```

```
        disconnected                (10),
        enabled                     (11),
        disabled                    (12),
        automatic-release-disabled  (13),
        default                     (14),
        ...
        }
-- Enumerated values 0-255 are reserved for definition by ASHRAE. Enumerated values
-- 256-65535 may be used by others subject to procedures and constraints described in Clause 23.

BACnetLifeSafetyOperation ::= ENUMERATED {
        none                (0),
        silence             (1),
        silence-audible     (2),
        silence-visual      (3),
        reset               (4),
        reset-alarm         (5),
        reset-fault         (6),
        unsilence           (7),
        unsilence-audible   (8),
        unsilence-visual    (9),
        ...
        }
-- Enumerated values 0-63 are reserved for definition by ASHRAE. Enumerated values
-- 64-65535 may be used by others subject to procedures and constraints described in
-- Clause 23.

BACnetLifeSafetyState ::= ENUMERATED {
        quiet               (0),
        pre-alarm           (1),
        alarm               (2),
        fault               (3),
        fault-pre-alarm     (4),
        fault-alarm         (5),
        not-ready           (6),
        active              (7),
        tamper              (8),
        test-alarm          (9),
        test-active         (10),
        test-fault          (11),
        test-fault-alarm    (12),
        holdup              (13),
        duress              (14),
        tamper-alarm        (15),
        abnormal            (16),
        emergency-power     (17),
        delayed             (18),
        blocked             (19),
        local-alarm         (20),
        general-alarm       (21),
        supervisory         (22),
        test-supervisory    (23),
        ...
        }
-- Enumerated values 0-255 are reserved for definition by ASHRAE. Enumerated values
-- 256-65535 may be used by others subject to procedures and constraints described in Clause 23.
```

```
BACnetLimitEnable ::= BIT STRING {
        lowLimitEnable   (0),
        highLimitEnable  (1)
        }

BACnetLogRecord ::=  SEQUENCE {
        timestamp           [0] BACnetDateTime,
        logDatum            [1] CHOICE {
                                log-status          [0] BACnetLogStatus,
                                boolean-value       [1] BOOLEAN,
                                real-value          [2] REAL,
                                enum-value          [3] ENUMERATED,       -- Optionally limited to 32 bits
                                unsigned-value      [4] Unsigned,     -- Optionally limited to 32 bits
                                signed-value        [5] INTEGER,    -- Optionally limited to 32 bits
                                bitstring-value     [6] BIT STRING,-- Optionally limited to 32 bits
                                null-value          [7] NULL,
                                failure             [8] Error,
                                time-change         [9] REAL,
                                any-value           [10] ABSTRACT-SYNTAX.&Type  -- Optional
                                }
        statusFlags         [2] BACnetStatusFlags OPTIONAL
        }

BACnetLogStatus ::= BIT STRING {
        log-disabled        (0),
        buffer-purged       (1)
        }

BACnetMaintenance ::= ENUMERATED {
        none                        (0),
        periodic-test               (1),
        need-service-operational    (2),
        need-service-inoperative    (3),
        ...
        }
-- Enumerated values 0-255 are reserved for definition by ASHRAE. Enumerated values
-- 256-65535 may be used by others subject to procedures and constraints described in
-- Clause 23.

BACnetNotificationParameters ::= CHOICE {
-- These choices have a one-to-one correspondence with the Event_Type enumeration with the exception of the
-- complex-event-type, which is used for proprietary event types.

        change-of-bitstring     [0] SEQUENCE {
                                    referenced-bitstring    [0] BIT STRING,
                                    status-flags            [1] BACnetStatusFlags
                                    },
        change-of-state         [1] SEQUENCE {
                                    new-state               [0] BACnetPropertyStates,
                                    status-flags            [1] BACnetStatusFlags
                                    },
        change-of-value         [2] SEQUENCE {
                                    new-value               [0] CHOICE {
                                                                changed-bits    [0] BIT STRING,
                                                                changed-value   [1] REAL
```

```
                                        },
                    status-flags        [1] BACnetStatusFlags
                    },
command-failure     [3] SEQUENCE {
                    command-value       [0] ABSTRACT-SYNTAX.&Type, -- depends on ref property
                    status-flags        [1] BACnetStatusFlags,
                    feedback-value      [2] ABSTRACT-SYNTAX.&Type -- depends on ref property
                    },
floating-limit      [4] SEQUENCE {
                    reference-value     [0] REAL,
                    status-flags        [1] BACnetStatusFlags,
                    setpoint-value      [2] REAL,
                    error-limit         [3] REAL
                    },
out-of-range        [5] SEQUENCE {
                    exceeding-value     [0] REAL,
                    status-flags        [1] BACnetStatusFlags,
                    deadband            [2] REAL,
                    exceeded-limit      [3] REAL
                    },
complex-event-type  [6] SEQUENCE OF BACnetPropertyValue,
                    -- complex tag 7 is deprecated
change-of-life-safety [8] SEQUENCE {
                    new-state           [0] BACnetLifeSafetyState,
                    new-mode            [1] BACnetLifeSafetyMode,
                    status-flags        [2] BACnetStatusFlags,
                    operation-expected  [3] BACnetLifeSafetyOperation
                    },
extended            [9] SEQUENCE {
                    vendorId            [0] Unsigned,
                    extendedEventType   [1] Unsigned,
                    parameters          [2] SEQUENCE OF CHOICE {
                                            null            NULL,
                                            real            REAL,
                                            integer         Unsigned,
                                            boolean         BOOLEAN,
                                            double          DOUBLE,
                                            octet           OCTET STRING,
                                            bitstring       BIT STRING,
                                            enum            ENUMERATED,
                                            propertyValue [0] BACnetDeviceObjectPropertyValue
                                            }
                    },
buffer-ready        [10] SEQUENCE {
                    buffer-property     [0] BACnetDeviceObjectPropertyReference,
                    previous-notification [1] Unsigned32,
                    current-notification [2] Unsigned32
                    },
unsigned-range      [11] SEQUENCE {
                    exceeding-value     [0] Unsigned,
                    status-flags        [1] BACnetStatusFlags,
                    exceeded-limit      [2] Unsigned
                    }
}
```

```
BACnetNotifyType ::= ENUMERATED {
        alarm           (0),
        event           (1),
        ack-notification (2)
        }

BACnetObjectPropertyReference ::= SEQUENCE {
        objectIdentifier        [0] BACnetObjectIdentifier,
        propertyIdentifier      [1] BACnetPropertyIdentifier,
        propertyArrayIndex      [2] Unsigned OPTIONAL  -- used only with array datatype
                                                       -- if omitted with an array the entire array is referenced
        }

BACnetObjectPropertyValue ::= SEQUENCE {
        objectIdentifier        [0] BACnetObjectIdentifier,
        propertyIdentifier      [1] BACnetPropertyIdentifier,
        propertyArrayIndex      [2] Unsigned OPTIONAL,          -- used only with array datatype
                                                                -- if omitted with an array the entire array is referenced
        value                   [3] ABSTRACT-SYNTAX.&Type, --any datatype appropriate for the specified property
        priority                [4] Unsigned (1..16) OPTIONAL
        }

BACnetObjectType ::= ENUMERATED {
        accumulator             (23),
        analog-input            (0),
        analog-output           (1),
        analog-value            (2),
        averaging               (18),
        binary-input            (3),
        binary-output           (4),
        binary-value            (5),
        calendar                (6),
        command                 (7),
        device                  (8),
        event-enrollment        (9),
        file                    (10),
        group                   (11),
        life-safety-point       (21),
        life-safety-zone        (22),
        loop                    (12),
        multi-state-input       (13),
        multi-state-output      (14),
        multi-state-value       (19),
        notification-class      (15),
        program                 (16),
        pulse-converter         (24),
        schedule                (17),
        -- see averaging        (18),
        -- see multi-state-value (19),
        trend-log               (20),
        -- see life-safety-point (21),
        -- see life-safety-zone (22),
        -- see accumulator      (23),
        -- see pulse-converter  (24),
        ...
        }
```

```
-- Enumerated values 0-127 are reserved for definition by ASHRAE. Enumerated values
-- 128-1023 may be used by others subject to the procedures and constraints described
-- in Clause 23.

BACnetObjectTypesSupported ::= BIT STRING {
        -- accumulator          (23),
        analog-input            (0),
        analog-output           (1),
        analog-value            (2),
        -- averaging            (18),
        binary-input            (3),
        binary-output           (4),
        binary-value            (5),
        calendar                (6),
        command                 (7),
        device                  (8),
        event-enrollment        (9),
        file                    (10),
        group                   (11),

        -- life-safety-point    (21),
        -- life-safety-zone     (22),
        loop                    (12),
        multi-state-input       (13),
        multi-state-output      (14),
        -- multi-state-value    (19),
        notification-class      (15),
        program                 (16),
        -- pulse-converter      (24),
        schedule                (17),
        -- trend-log            (20),
-- Objects added after 1995
        averaging               (18),
        multi-state-value       (19),
        trend-log               (20),
        life-safety-point       (21),
        life-safety-zone        (22),
-- Objects added after 2001
        accumulator             (23),
        pulse-converter         (24)
        }

BACnetPolarity ::= ENUMERATED {
        normal          (0),
        reverse         (1)
        }

BACnetPrescale ::= SEQUENCE {
        multiplier      [0] Unsigned,
        moduloDivide    [1] Unsigned
        }

BACnetPriorityArray ::= SEQUENCE SIZE (16) OF BACnetPriorityValue
        -- accessed as a BACnetARRAY

BACnetPriorityValue ::= CHOICE {
```

```
        null                NULL,
        real                REAL,
        binary              BACnetBinaryPV,
        integer             Unsigned,
        constructedValue    [0] ABSTRACT-SYNTAX.&Type
        }

BACnetProgramError ::= ENUMERATED {
        normal          (0),
        load-failed     (1),
        internal        (2),
        program         (3),
        other           (4),
        ...
        }
-- Enumerated values 0-63 are reserved for definition by ASHRAE. Enumerated values
-- 64-65535 may be used by others subject to the procedures and constraints described
-- in Clause 23.

BACnetProgramRequest ::= ENUMERATED {
        ready           (0),
        load            (1),
        run             (2),
        halt            (3),
        restart         (4),
        unload          (5)
        }

BACnetProgramState ::= ENUMERATED {
        idle            (0),
        loading         (1),
        running         (2),
        waiting         (3),
        halted          (4),
        unloading       (5)
        }

BACnetPropertyIdentifier ::= ENUMERATED {
        accepted-modes                  (175),
        acked-transitions               (0),
        ack-required                    (1),
        action                          (2),
        action-text                     (3),
        active-text                     (4),
        active-vt-sessions              (5),
        active-cov-subscriptions        (152),
        adjust-value                    (176),
        alarm-value                     (6),
        alarm-values                    (7),
        all                             (8),
        all-writes-successful           (9),
        apdu-segment-timeout            (10),
        apdu-timeout                    (11),
        application-software-version    (12),
        archive                         (13),
        attempted-samples               (124),
```

```
    auto-slave-discovery                    (169),
    average-value                           (125),
    backup-failure-timeout                  (153),
    bias                                    (14),
    buffer-size                             (126),
    change-of-state-count                   (15),
    change-of-state-time                    (16),
    -- see notification-class               (17),
    -- the property in this place was deleted (18),
    client-cov-increment                    (127),
    configuration-files                     (154),
    controlled-variable-reference           (19),
    controlled-variable-units               (20),
    controlled-variable-value               (21),
    count                                   (177),
    count-before-change                     (178),
    count-change-time                       (179),
    cov-increment                           (22),
    cov-period                              (180),
    cov-resubscription-interval             (128),
    --  current-notify-time                 (129),    This property was deleted in version 1 revision 3.
    database-revision                       (155),
    date-list                               (23),
    daylight-savings-status                 (24),
    deadband                                (25),
    derivative-constant                     (26),
    derivative-constant-units               (27),
    description                             (28),
    description-of-halt                     (29),
    device-address-binding                  (30),
    device-type                             (31),
    direct-reading                          (156),
    effective-period                        (32),
    elapsed-active-time                     (33),
    error-limit                             (34),
    event-enable                            (35),
    event-state                             (36),
    event-time-stamps                       (130),
    event-type                              (37),
    event-parameters                        (83), -- renamed from previous version
    exception-schedule                      (38),
    fault-values                            (39),
    feedback-value                          (40),
    file-access-method                      (41),
    file-size                               (42),
    file-type                               (43),
    firmware-revision                       (44),
    high-limit                              (45),
    inactive-text                           (46),
    in-process                              (47),
    input-reference                         (181),
    instance-of                             (48),
    integral-constant                       (49),
    integral-constant-units                 (50),
    --  issue-confirmed-notifications       (51),     This property was deleted in version 1 revision 4.
    last-notify-record                      (173),
```

```
last-restore-time                   (157),
life-safety-alarm-values            (166),
limit-enable                        (52),
limit-monitoring-interval           (182),
list-of-group-members               (53),
list-of-object-property-references  (54),
list-of-session-keys                (55),
local-date                          (56),
local-time                          (57),
location                            (58),
log-buffer                          (131),
log-device-object-property          (132),
log-enable                          (133),
log-interval                        (134),
logging-object                      (183),
logging-record                      (184),
low-limit                           (59),
maintenance-required                (158),
manipulated-variable-reference      (60),
manual-slave-address-binding        (170),
maximum-output                      (61),
maximum-value                       (135),
maximum-value-timestamp             (149),
max-apdu-length-accepted            (62),
max-info-frames                     (63),
max-master                          (64),
max-pres-value                      (65),
max-segments-accepted               (167),
member-of                           (159),
minimum-off-time                    (66),
minimum-on-time                     (67),
minimum-output                      (68),
minimum-value                       (136),
minimum-value-timestamp             (150),
min-pres-value                      (69),
mode                                (160),
model-name                          (70),
modification-date                   (71),
notification-class                  (17), -- renamed from previous version
notification-threshold              (137),
notify-type                         (72),
number-of-APDU-retries              (73),
number-of-states                    (74),
object-identifier                   (75),
object-list                         (76),
object-name                         (77),
object-property-reference           (78),
object-type                         (79),
operation-expected                  (161),
optional                            (80),
out-of-service                      (81),
output-units                        (82),
-- see event-parameters             (83),
polarity                            (84),
prescale                            (185),
present-value                       (85),
```

-- previous-notify-time	(138),	This property was deleted in version 1 revision 3.
priority	(86),	
pulse-rate	(186),	
priority-array	(87),	
priority-for-writing	(88),	
process-identifier	(89),	
profile-name	(168),	
program-change	(90),	
program-location	(91),	
program-state	(92),	
proportional-constant	(93),	
proportional-constant-units	(94),	
-- protocol-conformance-class	(95),	This property was deleted in version 1 revision 2.
protocol-object-types-supported	(96),	
protocol-revision	(139),	
protocol-services-supported	(97),	
protocol-version	(98),	
read-only	(99),	
reason-for-halt	(100),	
-- recipient	(101),	This property was deleted in version 1 revision 4.
recipient-list	(102),	
records-since-notification	(140),	
record-count	(141),	
reliability	(103),	
relinquish-default	(104),	
required	(105),	
resolution	(106),	
scale	(187),	
scale-factor	(188),	
schedule-default	(174),	
segmentation-supported	(107),	
setpoint	(108),	
setpoint-reference	(109),	
slave-address-binding	(171),	
setting	(162),	
silenced	(163),	
start-time	(142),	
state-text	(110),	
status-flags	(111),	
stop-time	(143),	
stop-when-full	(144),	
system-status	(112),	
time-delay	(113),	
time-of-active-time-reset	(114),	
time-of-state-count-reset	(115),	
time-synchronization-recipients	(116),	
total-record-count	(145),	
tracking-value	(164),	
units	(117),	
update-interval	(118),	
update-time	(189),	
utc-offset	(119),	
valid-samples	(146),	
value-before-change	(190),	
value-set	(191),	
value-change-time	(192),	

```
    variance-value                      (151),
    vendor-identifier                   (120),
    vendor-name                         (121),
    vt-classes-supported                (122),
    weekly-schedule                     (123),
    -- see attempted-samples            (124),
    -- see average-value                (125),
    -- see buffer-size                  (126),
    -- see client-cov-increment         (127),
    -- see cov-resubscription-interval  (128),
    --  unused                          (129),   current-notify-time was deleted in version 1 revision 3.
    -- see event-time-stamps            (130),
    -- see log-buffer                   (131),
    -- see log-device-object-property   (132),
    -- see log-enable                   (133),
    -- see log-interval                 (134),
    -- see maximum-value                (135),
    -- see minimum-value                (136),
    -- see notification-threshold       (137),
    -- unused                           (138),   previous-notify-time was deleted in version 1 revision 3.
    -- see protocol-revision            (139),
    -- see records-since-notification   (140),
    -- see record-count                 (141),
    -- see start-time                   (142),
    -- see stop-time                    (143),
    -- see stop-when-full               (144),
    -- see total-record-count           (145),
    -- see valid-samples                (146),
    window-interval                     (147),
    window-samples                      (148),
    zone-members                        (165),
    -- see maximum-value-timestamp      (149),
    -- see minimum-value-timestamp      (150),
    -- see variance-value               (151),
    -- see active-cov-subscriptions     (152),
    -- see backup-failure-timeout       (153),
    -- see configuration-files          (154),
    -- see database-revision            (155),
    -- see direct-reading               (156),
    -- see last-restore-time            (157),
    -- see maintenance-required         (158),
    -- see member-of                    (159),
    -- see mode                         (160),
    -- see operation-expected           (161),
    -- see setting                      (162),
    -- see silenced                     (163),
    -- see tracking-value               (164),
    -- see zone-members                 (165),
    -- see life-safety-alarm-values     (166),
    -- see max-segments-accepted        (167),
    -- see profile-name                 (168),
    -- see auto-slave-discovery         (169),
    -- see manual-slave-address-binding (170),
    -- see slave-address-binding        (171),
    -- see slave-proxy-enable           (172),
    -- see last-notify-time             (173),
```

```
        -- see schedule-default                 (174),
        -- see accepted-modes                   (175),
        -- see adjust-value                     (176),
        -- see count                            (177),
        -- see count-before-change              (178),
        -- see count-change-time                (179),
        -- see cov-period                       (180),
        -- see input-reference                  (181),
        -- see limit-monitoring-interval        (182),
        -- see logging-device                   (183),
        -- see logging-record                   (184),
        -- see prescale                         (185),
        -- see pulse-rate                       (186),
        -- see scale                            (187),
        -- see scale-factor                     (188),
        -- see update-time                      (189),
        -- see value-before-change              (190),
        -- see value-set                        (191),
        -- see value-change-time                (192),
        ...
        }

-- The special property identifiers all, optional, and required are reserved for use in the ReadPropertyConditional and
-- ReadPropertyMultiple services or services not defined in this standard.
--
-- Enumerated values 0-511 are reserved for definition by ASHRAE. Enumerated values 512-4194303 may be used by
-- others subject to the procedures and constraints described in Clause 23. The highest enumeration used in this version is 192.

BACnetPropertyReference ::= SEQUENCE {
        propertyIdentifier          [0] BACnetPropertyIdentifier,
        propertyArrayIndex          [1] Unsigned OPTIONAL  --used only with array datatype
                                                           -- if omitted with an array the entire array is referenced
        }

BACnetPropertyStates ::= CHOICE {
-- This production represents the possible datatypes for properties that
-- have discrete or enumerated values. The choice must be consistent with the
-- datatype of the property referenced in the Event Enrollment Object.

        boolean-value               [0] BOOLEAN,
        binary-value                [1] BACnetBinaryPV,
        event-type                  [2] BACnetEventType,
        polarity                    [3] BACnetPolarity,
        program-change              [4] BACnetProgramRequest,
        program-state               [5] BACnetProgramState,
        reason-for-halt             [6] BACnetProgramError,
        reliability                 [7] BACnetReliability,
        state                       [8] BACnetEventState,
        system-status               [9] BACnetDeviceStatus,
        units                       [10] BACnetEngineeringUnits,
        unsigned-value              [11] Unsigned,
        life-safety-mode            [12] BACnetLifeSafetyMode,
        life-safety-state           [13] BACnetLifeSafetyState,
        ...
        }
```

```
-- Tag values 0-63 are reserved for definition by ASHRAE. Tag values of 64-254 may be used by others to
-- accommodate vendor specific properties that have discrete or enumerated values, subject to the constraints described
-- in Clause 23.

BACnetPropertyValue ::= SEQUENCE {
        PropertyIdentifier          [0] BACnetPropertyIdentifier,
        propertyArrayIndex          [1] Unsigned OPTIONAL,  -- used only with array datatypes
                                                            -- if omitted with an array the entire array is referenced
        value                       [2] ABSTRACT-SYNTAX.&Type, -- any datatype appropriate for the specified property
        priority                    [3] Unsigned (1..16) OPTIONAL  -- used only when property is commandable
        }

BACnetRecipient ::= CHOICE {
        device    [0] BACnetObjectIdentifier,
        address   [1] BACnetAddress
        }

BACnetRecipientProcess ::= SEQUENCE {
        recipient [0] BACnetRecipient,
        processIdentifier   [1] Unsigned32
        }

BACnetReliability ::= ENUMERATED {
        no-fault-detected       (0),
        no-sensor               (1),
        over-range              (2),
        under-range             (3),
        open-loop               (4),
        shorted-loop            (5),
        no-output               (6),
        unreliable-other        (7),
        process-error           (8),
        multi-state-fault       (9),
        configuration-error     (10),
        ...
        }
-- Enumerated values 0-63 are reserved for definition by ASHRAE. Enumerated values
-- 64-65535 may be used by others subject to the procedures and constraints described
-- in Clause 23.

BACnetResultFlags ::= BIT STRING {
        first-item              (0),
        last-item               (1),
        more-items              (2)
        }

BACnetScale ::= CHOICE {
        floatScale          [0] REAL,
        integerScale        [1] INTEGER
        }
```

```
BACnetSegmentation ::= ENUMERATED {
        segmented-both          (0),
        segmented-transmit      (1),
        segmented-receive       (2),
        no-segmentation         (3)
        }

BACnetServicesSupported ::= BIT STRING {
-- Alarm and Event Services
        acknowledgeAlarm                (0),
        confirmedCOVNotification        (1),
        confirmedEventNotification      (2),
        getAlarmSummary                 (3),
        getEnrollmentSummary            (4),
        -- getEventInformation          (39),
        subscribeCOV                    (5),
        -- subscribeCOVProperty         (38),
        -- lifeSafetyOperation          (37),

-- File Access Services
        atomicReadFile                  (6),
        atomicWriteFile                 (7),

-- Object Access Services
        addListElement                  (8),
        removeListElement               (9),
        createObject                    (10),
        deleteObject                    (11),
        readProperty                    (12),
        readPropertyConditional         (13),
        readPropertyMultiple            (14),
        -- readRange                    (35),
        writeProperty                   (15),
        writePropertyMultiple           (16),

-- Remote Device Management Services
        deviceCommunicationControl      (17),
        confirmedPrivateTransfer        (18),
        confirmedTextMessage            (19),
        reinitializeDevice              (20),

-- Virtual Terminal Services
        vtOpen                          (21),
        vtClose                         (22),
        vtData                          (23),

-- Security Services
        authenticate                    (24),
        requestKey                      (25),

-- Unconfirmed Services
        i-Am                            (26),
        i-Have                          (27),
        unconfirmedCOVNotification      (28),
        unconfirmedEventNotification    (29),
        unconfirmedPrivateTransfer      (30),
```

```
        unconfirmedTextMessage          (31),
        timeSynchronization             (32),
        -- utcTimeSynchronization       (36),
        who-Has                         (33),
        who-Is                          (34),

-- Services added after 1995
        readRange                       (35), -- Object Access Service
        utcTimeSynchronization          (36), -- Remote Device Management Service
        lifeSafetyOperation             (37), -- Alarm and Event Service
        subscribeCOVProperty            (38), -- Alarm and Event Service
        getEventInformation             (39)  -- Alarm and Event Service
        }

BACnetSessionKey ::= SEQUENCE {
        sessionKey      OCTET STRING (SIZE(8)), -- 56 bits for key, 8 bits for checksum
        peerAddress     BACnetAddress
        }

BACnetSetpointReference ::= SEQUENCE {
        setpointReference [0] BACnetObjectPropertyReference OPTIONAL
                }

BACnetSilencedState ::= ENUMERATED {
        unsilenced      (0),
        audible-silenced (1),
        visible-silenced (2),
        all-silenced    (3),
        ...
        }
-- Enumerated values 0-63 are reserved for definition by ASHRAE. Enumerated values
-- 64-65535 may be used by others subject to procedures and constraints described in
-- Clause 23.

BACnetSpecialEvent ::= SEQUENCE {
        period                  CHOICE {
                                        calendarEntry           [0] BACnetCalendarEntry,
                                        calendarReference       [1] BACnetObjectIdentifier
                                                                },
        listOfTimeValues        [2] SEQUENCE OF BACnetTimeValue,
        eventPriority           [3] Unsigned (1..16)
        }

BACnetStatusFlags ::= BIT STRING {
        in-alarm        (0),
        fault           (1),
        overridden      (2),
        out-of-service  (3)
        }

BACnetTimeStamp ::= CHOICE {
        time                    [0]     Time,
        sequenceNumber          [1]     Unsigned (0..65535),
        dateTime                [2]     BACnetDateTime
        }
```

```
BACnetTimeValue ::= SEQUENCE {
        time            Time,
        value           ABSTRACT-SYNTAX.&Type  -- any primitive datatype, complex types cannot be decoded
        }

BACnetVTClass ::= ENUMERATED {
        default-terminal  (0),
        ansi-x3-64        (1),
        dec-vt52          (2),
        dec-vt100         (3),
        dec-vt220         (4),
        hp-700-94         (5),
        ibm-3130          (6),
        ...
        }
-- Enumerated values 0-63 are reserved for definition by ASHRAE. Enumerated values
-- 64-65535 may be used by others subject to the procedures and constraints described
-- in Clause 23.

BACnetVTSession ::= SEQUENCE {
        local-vtSessionID       Unsigned8,
        remote-vtSessionID      Unsigned8,
        remote-vtAddress        BACnetAddress
        }

BACnetWeekNDay ::= OCTET STRING (SIZE (3))
--      first octet     month (1..14)           1 =January
--                                              13 = odd months
--                                              14 = even months
--                                              X'FF' = any month
--      second octet    weekOfMonth where:      1 = days numbered 1-7
--                                              2 = days numbered 8-14
--                                              3 = days numbered 15-21
--                                              4 = days numbered 22-28
--                                              5 = days numbered 29-31
--                                              6 = last 7 days of this month
--                                              X'FF' = any week of this month
--      third octet     dayOfWeek (1..7) where  1 = Monday
--                                              7 = Sunday
--                                              X'FF' = any day of week

ReadAccessResult ::= SEQUENCE {
        objectIdentifier        [0] BACnetObjectIdentifier,
        listOfResults           [1] SEQUENCE OF SEQUENCE {
                propertyIdentifier      [2] BACnetPropertyIdentifier,
                propertyArrayIndex      [3] Unsigned OPTIONAL, -- used only with array datatype
                                                               -- if omitted with an array the entire
                                                               -- array is referenced
                readResult      CHOICE  {
                                        propertyValue           [4] ABSTRACT-SYNTAX.&Type,
                                        propertyAccessError     [5] Error
                                        }
                } OPTIONAL
        }
```

```
ReadAccessSpecification ::= SEQUENCE {
        objectIdentifier            [0] BACnetObjectIdentifier,
        listOfPropertyReferences    [1] SEQUENCE OF BACnetPropertyReference
        }

WriteAccessSpecification ::= SEQUENCE {
        objectIdentifier                    [0] BACnetObjectIdentifier,
        listOfProperties                    [1] SEQUENCE OF BACnetPropertyValue
        }

END
```

22 CONFORMANCE AND INTEROPERABILITY

BACnet defines a comprehensive set of object types and application services in the sense that communication requirements among all levels of control in a distributed, hierarchical building automation system are addressed. There is a need to account for the reality that not all devices in a building automation system need to support the full functionality of BACnet in order to perform their tasks.

To reach the overarching goal of this standard – communication between disparate building automation and control devices, possibly from different manufacturers – two distinct conditions must be met: 1) each implemented BACnet capability must precisely conform to the requirements of this standard; and 2) devices that seek to interoperate must implement precisely complementary BACnet capabilities appropriate to the desired form of interoperation. This clause defines how these conditions are to be met and what it means to conform to BACnet.

22.1 Conformance to BACnet

This subclause specifies the requirements that shall be met in order to conform with BACnet.

22.1.1 Protocol Implementation Conformance Statement (PICS)

All devices conforming to the BACnet protocol shall have a Protocol Implementation Conformance Statement (PICS) that identifies all of the portions of BACnet that are implemented. This PICS shall contain all of the information described in 22.1.1.1 and shall be in the format found in Annex A.

22.1.1.1 PICS Contents

A PICS is a written document, created by the manufacturer of a device, that identifies the particular options specified by BACnet that are implemented in the device. A BACnet PICS is considered a public document that is available for use by any interested party. At a minimum, a BACnet PICS shall convey the following information.

(a) Basic information identifying the vendor and describing the BACnet device.

(b) The BACnet Interoperability Building Blocks supported by the device (see Annex K).

(c) The standardized BACnet device profile to which the device conforms, if any (see Annex L).

(d) All non-standard application services that are supported along with an indication for each service of whether the device can initiate the service request, respond to a service request, or both.

(e) A list of all standard and proprietary object types that are supported.

(f) For each object type supported,

1. any optional properties that are supported,
2. which properties can be written-to using BACnet services,
3. if the objects can be dynamically created or deleted using BACnet services,
4. any restrictions on the range of data values for properties.

(g) The data link layer option options, both real and virtual, supported. (See Annexes H and J).

(h) Whether segmented requests are supported.

(i) Whether segmented responses are supported.

22.1.2 Conformance Test

In order to conform to the BACnet protocol, all devices shall pass a conformance test that verifies the correct implementation of the standard object types and services indicated in the PICS. This conformance test shall consist of a collection of test cases drawn from a standard test suite in such a way as to test each object type and service for which support is claimed (positive test) and to test for an appropriate response to errors and standard services and objects that are not implemented to

ensure the absence of detrimental behavior (negative test). The details of these tests are prescribed in the companion standard, "Testing Conformance to BACnet," ASHRAE 135.1.

22.1.3 Data Link and Physical Layers

To conform to the BACnet protocol, all devices shall support one of the five data link layer options, defined in Clauses 7 through 11, and one of the physical layers compatible with that data link layer, except as indicated in 22.1.4.

22.1.4 Conformance with Non-Standard Data Link Layer

Special circumstances may require that a device support a data link and physical layer technology that is not one of the BACnet options in order to interoperate with other networked devices in a particular situation. Such a device may be said to conform to BACnet with a non-standard data link layer, provided that the criteria in 22.1.1 through 22.1.2 are met.

A device conforming to the BACnet protocol under the provisions of this subclause may use non-standard protocol layers other than the data link and physical layers, provided that the non-standard protocol is used to convey a standard BACnet LSDU that contains application and network layer information defined by this standard and encoded according to the rules of Clause 20 and Clause 6. Segmentation of the BACnet LSDU is permitted. Annex H provides examples of this for the Department of Defense Internet protocols and the Novell Internetwork Datagram Protocol.

22.2 BACnet Interoperability

BACnet is intended to provide a single, uniform standard for building control systems, the ultimate goal of which is "interoperability." Interoperability means the ability of disparate control system devices to work together toward a common objective through the digital exchange of relevant information. Although interoperability is often thought of in terms of interconnecting equipment from multiple manufacturers, it is also possible to envision interoperating systems from a single vendor, possibly equipment of different vintages. Thus, while BACnet enables multi-vendor interoperability, it in no way requires it.

22.2.1 Interoperability Areas

"Interoperability areas" (IAs) are intended to describe the functionality that is important in practical automation and control systems to achieve specific operational objectives. The five IAs delineated in this standard are data sharing, alarm and event management, scheduling, trending, and device and network management. Each IA implies a set of capabilities. Each capability, in turn, requires that specific elements of BACnet be implemented in a particular device to enable interoperability in a known and predictable manner with a minimum of field engineering. The selection of which BACnet elements are required for a particular type of device is indicated in the device profiles presented in Annex L. This section describes the specific capabilities associated with each IA.

22.2.1.1 Data Sharing

"Data sharing" is the exchange of information between BACnet devices. It may be uni-directional or bi-directional. Interoperability in this area permits the collection of data for archival storage, graphics, and reports, the sharing of common sensor or calculated values between devices, carrying out interlocked control strategies, and the modification of setpoints or other operational parameters of BACnet objects.

22.2.1.2 Alarm and Event Management

"Alarm and event management" is the exchange of data between BACnet devices related to the occurrence of a predefined condition that meets specific criteria. Such conditions are called "events" and may be the basis for the initiation of a particular control action in response or the simple logging of the event's occurrence. The event may also be deemed to represent a condition that constitutes an "alarm", requiring human acknowledgment and intervention. Interoperability in this area permits the annunciation and acknowledgment of alarms; the display of data indicating the basis for the alarm annunciation; the sharing of events for the purpose of logging or distributed control applications; modification of alarm limits and routing; and the production of summaries of the occurrence of such alarms and events.

BACnet defines two different mechanisms for generating alarms and events. One is called "intrinsic reporting" because it relies on the use of properties that are part of or "intrinsic" to the object that is being monitored for alarms or events. The other mechanism is called "algorithmic change reporting." Algorithmic change reporting is more general but it also requires the overhead of an additional object called the Event Enrollment object. The intrinsic reporting method is preferred under circumstances where it meets the objectives of the intended application. See Clause 13.

22.2.1.3 Scheduling

"Scheduling" is the exchange of data between BACnet devices related to the establishment and maintenance of dates and times at which specified output actions are to be taken. Interoperability in this area permits the use of date and time schedules for starting and stopping equipment and changing control setpoints as well as other analog or binary parameters.

22.2.1.4 Trending

"Trending" is the accumulation of (time, value) pairs at specified rates for a specified duration. The values are those of a specific property of a specific object. "Trending" is distinguished from the real-time plotting of data in that the data are usually destined for long-term storage and the sampling intervals are usually longer. Interoperability in this area permits the establishment of trending parameters and the subsequent retrieval and storage of trend data.

22.2.1.5 Device and Network Management

"Device and network management" is the exchange of data between BACnet devices concerning the operation and status of the devices comprising the BACnet internetwork. Interoperability in this area permits determining which devices are present on a given network and some of their operational capabilities, including which objects they maintain; the ability to start up and shut down communication from a particular device; the ability to synchronize the time in those devices that maintain clocks; the ability to reinitialize the operation of a device's computer; the ability to establish connections as needed; and the ability to change the connection configuration.

23 EXTENDING BACnet TO ACCOMMODATE VENDOR PROPRIETARY INFORMATION

The objective of BACnet is to provide the mechanisms by which building automation equipment may exchange information. To aid in interoperability, BACnet defines a standardized set of data structures, called objects, which contain information that is common to most building systems. BACnet may also be used to exchange non-standardized information between devices that understand this information. There are four independent areas where BACnet may be extended to exchange non-standard information:

(a) A vendor may define proprietary extended values for enumerations used in BACnet.

(b) A vendor may invoke a proprietary service using the PrivateTransfer services.

(c) A vendor may add new proprietary properties to a standard object.

(d) A vendor may define new proprietary object types.

In each of these cases, the BACnet messages implicitly reference a vendor identification code that serves to unambiguously specify which vendor's proprietary enumerations, services, properties, or objects were intended. Vendor identification codes are administrated by ASHRAE and are assigned one per vendor. The special Vendor_Identifier of zero is permanently assigned to ASHRAE. The Vendor_Identifier for a given device may be determined by reading the Vendor_Identifier property of the Device object. A list of vendor identification codes may be obtained from the ASHRAE Manager of Standards.

23.1 Extending Enumeration Values

There may be instances when it is necessary for a vendor to extend BACnet by including additional possible values to an enumeration. This is accomplished by using enumeration values that are greater than the range reserved for BACnet for a given enumeration type. Table 23-1 defines those enumerations that may be extended and the range of enumerated values reserved for BACnet use. All other enumerations, which do not appear in Table 23-1, may not be extended.

Table 23-1. Extensible Enumerations

Enumeration Name	Reserved Range	Maximum Value
error-class	0-63	65535
error-code	0-255	65535
BACnetAbortReason	0-63	255
BACnetDeviceStatus	0-63	65535
BACnetEngineeringUnits	0-255	65535
BACnetEventState	0-63	65535
BACnetEventType	0-63	65535
BACnetLifeSafetyMode	0-255	65535
BACnetLifeSafetyState	0-255	65535
BACnetLifeSafetyOperation	0-63	65535
BACnetMaintenance	0-255	65535
BACnetObjectType	0-127	1023
BACnetProgramError	0-63	65535
BACnetPropertyIdentifier	0-511	4194303
BACnetPropertyStates	0-63	254
BACnetReliability	0-63	65535
BACnetRejectReason	0-63	255
BACnetSilencedState	0-63	65535
BACnetVTClass	0-63	65535

23.2 Using the PrivateTransfer Services to Invoke Non-Standardized Services

BACnet defines a set of application layer services that are specifically tailored to integrating building control systems. While this standard prescribes a set of application layer services that is intended to be comprehensive, vendors are free to create additional services. Standard services shall be used when possible.

Vendors may add proprietary services to BACnet using the PrivateTransfer services to invoke them. The service types and arguments are not restricted by BACnet, but they shall be conveyed using the Confirmed or Unconfirmed PrivateTransfer services. The protocol mechanisms used in the handling of these APDUs shall perform as specified in this standard.

The format of proprietary application layer services, invoked using PrivateTransfer, shall follow the encoding rules of this standard.

When using the PrivateTransfer service, it is important to note that segmentation is not permitted for Error APDUs. The implementor shall ensure that the parameters in the Error APDU do not expand to the point where segmentation is required.

23.3 Adding Proprietary Properties to a Standardized Object

BACnet defines a set of standard objects, each with a set of standard properties that can be accessed and manipulated with BACnet services. BACnet allows a vendor to add proprietary properties to extend the capabilities of a standard object. Proprietary properties receive the same support from BACnet services as standard properties and therefore can be accessed and manipulated in a manner identical to standard properties.

Objects may indicate conformance to an object profile by use of the Profile_Name property.

If a proprietary property is to be a commandable property, additional properties that fulfill the role of the standard Priority_Array and Relinquish_Default properties shall be provided for each commandable property. The priority arbitration mechanism described in Clause 19 shall apply.

Vendors may add proprietary properties to a standard object by modifying the object definition within a device. Proprietary properties are enumerated with Property_Identifier values of 512 and above. These property identifiers can be used in any BACnet service that uses a Property_Identifier as a parameter.

Proprietary property identifiers implicitly reference the Vendor_Identifier property of the Device object in the device where they reside. It is entirely possible, and expected, that different vendors will use the same enumeration values to represent completely different properties.

23.4 Adding Proprietary Object Types to BACnet

To accommodate building applications where the defined set of standardized objects is not adequate, BACnet allows a vendor to add proprietary object types. Standard object types shall be used when possible. To enhance extensibility, BACnet provides the same support for proprietary objects as for standard objects.

Objects may indicate conformance to an object profile by use of the Profile_Name property.

23.4.1 Proprietary Object_Type Enumerations

Vendors may add proprietary object types to BACnet by extending the BACnetObjectType enumeration. Proprietary object types are enumerated with Object_Type values of 128 and above. These Object_Type values may be used in any BACnet service that uses an Object_Type as a parameter.

23.4.2 Proprietary Property Datatypes

The properties of vendor proprietary objects may include both standard and proprietary datatypes. Proprietary datatypes may only be constructed from application datatypes defined in 20.2.1.4.

23.4.3 Required Properties in Proprietary Object Types

Non-standard object types shall support the following properties:

Object_Identifier
Object_Name
Object_Type

These properties shall be implemented to behave as they would in standard BACnet objects. This means that the Object_Identifier and Object_Name properties shall be unique within the BACnet device that maintains them. The Object_Name string shall be at least one character in length and shall consist of only printable characters.

23.5 Restrictions on Extending BACnet

The following restrictions to extending BACnet apply:

(a) APDU types 8 through 15 are reserved for future ASHRAE use.

(b) Services may be added only via the Confirmed- and UnconfirmedPrivateTransfer services. That is, the enumerations BACnetConfirmedServiceChoice and BACnetUnconfirmedServiceChoice may not be extended.

24 NETWORK SECURITY

This clause defines a limited security architecture for BACnet. Network security in BACnet is optional. The intent of this architecture is to provide peer entity, data origin, and operator authentication, as well as data confidentiality and integrity. Other types of communications security, such as access control and non-repudiation, are not provided in the BACnet standard. Systems that require these services may add them to BACnet in a proprietary manner.

24.1 Security Architecture

24.1.1 Overview

Peer and data origin authentication is performed in BACnet using handshaking. Handshaking will ensure that communication has been established between two known nodes within a communications network. Handshaking will be performed for one of two reasons:

(a) The Client device needs to authenticate that a confirmed service request was indeed executed by the claimed device. With this mechanism, unconfirmed service requests cannot be authenticated.

(b) The Server device needs to authenticate that a confirmed service request was indeed initiated by the claimed device. With this mechanism, unconfirmed service requests cannot be authenticated.

This is accomplished by testing that bound enciphered data can be communicated between the two nodes using a common Session Key (SK). SKs are obtained from a device known as a key server.

Operator authentication is performed in BACnet by authenticating a password provided by a user. The password can be authenticated locally by the device or over the network. If the password is authenticated over the network, the key server shall contain the master list of passwords.

Data confidentiality and integrity are performed in BACnet using encipherment. Data are enciphered using a common SK that is obtained from a key server. With this mechanism, broadcast or multicast messages cannot be enciphered.

SKs are distributed to the secure BACnet devices by the key server. A device requests a key between itself and another device from the key server using the RequestKey service. Alternatively, keys may be delivered by the key server to devices upon network initialization or at a timed interval. The key server shall deliver the SK to the devices by writing to the List_Of_Session_Keys property of the Device object using the AddListElement service. The SKs being sent over the BACnet network shall be enciphered using the unique Private Key (PK) of each device. Once in the device, each SK can be deciphered for use. The lifetime for SKs is a local matter.

24.1.2 Private Cryptographic Keys

Any BACnet device X that needs to authenticate peers or encipher data needs to hold a unique Private Key (PK_x). PK_x is a 56-bit Data Encryption Standard (DES) cryptographic key. PK_x is used to decipher SKs that are used in secure BACnet transactions.

Generation and distribution of the PKs in BACnet are considered local matters.

24.1.3 Session Cryptographic Keys

The key server generates and distributes all the required SKs. To do this, it must hold a copy of all the PKs used on the BACnet network. An SK is a 56-bit DES cryptographic key. The method for generating an SK is a local matter. SKs are distributed, protected by a PK, using the AddListElement service.

24.1.4 Cryptographic Algorithm

Enciphering and deciphering of data shall be performed using the DES algorithm. DES is specified in Federal Information Processing Standard 46-2 (also known as ANSI X3.92). To prevent compromise of the data or the cryptographic keys, all enciphered APDUs shall be padded, before encipherment, with random data up to the maximum length supported by the recipient device.

24.2 Authentication Mechanisms

24.2.1 Procedure for Obtaining a Session Key

The procedure for obtaining an SK is as follows:

(a) Device A, which requires authentication, sends a RequestKey service request to the key server indicating that remote client B will be authenticated.

(b) The key server authenticates that the RequestKey service request was initiated by device A, using PK_A in the message initiation authentication procedure in 24.2.4.

(c) The key server generates a Session Key for devices A and B (SK_{AB}).

(d) The key server enciphers SK_{AB} and the BACnetAddress of device A with the PK_B of device B. This information is then sent to device B using an AddListElement request.

(e) Device B accepts the AddListElement request and authenticates that the key was indeed sent by the key server, using PK_B in the message initiation authentication procedure in 24.2.4.

(f) Once the key server has been authenticated, device B shall decipher SK_{AB} and place it in its Device object. A 'Result(+)' is then returned to the key server for the AddListElement request.

(g) The key server enciphers SK_{AB} and the BACnetAddress of client B with the PK_A of client A. This information is then sent to client A using a AddListElement request.

(h) Device A accepts the AddListElement request and authenticates that the key was indeed sent by the key server, using PK_A in the message initiation authentication procedure in 24.2.4.

(i) Once the key server has been authenticated, client A shall decipher SK_{AB} and place it in its Device object. A 'Result(+)' is then returned to the key server for the AddListElement request.

24.2.2 Peer Authentication Mechanism

The procedure of peer authentication is summarized as follows:

(a) Once device A receives SK_{AB}, authentication of device B can begin.

(b) Device A generates an Authenticate service request and sends it to device B. The Service Request portion of the BACnet-Confirmed-Request-PDU, i.e., the fully formed Authenticate-Request production, is enciphered with SK_{AB}. The APDU header and all lower layer PCI are left unenciphered.

(c) If device B can decipher the Authenticate service request, it shall modify the 'Pseudo Random Number' as specified in the Authenticate service procedure, encipher it as the 'Modified Random Number' using SK_{AB}, and return it in the Authenticate-ACK portion of a ComplexACK to device A.

(d) If device A receives a ComplexACK with the correct 'Modified Random Number', then device B is authenticated.

24.2.3 Message Execution Authentication

The method used by a BACnet client to authenticate that a message is being executed by the correct BACnet server is performed by enhancing peer authentication to include a binding to a Transaction State Machine (TSM). This implies that message execution authentication can be performed only on confirmed service requests. The process of message execution authentication consists of two parts, obtaining an SK and authentication of message receipt. The process of obtaining an SK is described in 24.2.1. The procedure for Client A to authenticate that Server B is executing a certain message is as follows:

(a) Once Client A receives SK_{AB}, message execution authentication of Server B can begin.

(b) Client A generates an Authenticate service request and sends it to Server B. The 'Expected Invoke ID' parameter of this message shall contain the invokeID of the Confirmed Request APDU being authenticated. The Service Request

portion of the BACnet-Confirmed-Request-PDU, i.e., the fully formed Authenticate-Request production, is enciphered with SK_{AB}. The APDU header and all lower layer PCI are left unenciphered.

(c) After sending the Authenticate service request, Client A shall immediately send the Confirmed Request APDU that is being authenticated to Server B. The InvokeId of the Confirmed Request APDU shall match the 'Expected Invoke ID' of the Authenticate service request.

(d) If Server B can decipher the Authenticate service request, it shall wait up to 30 seconds for the Confirmed Request APDU with the 'Expected Invoke ID'. If the APDU is received, Server B shall modify the 'Pseudo Random Number' as specified in the Authenticate service procedure, encipher it as the 'Modified Random Number' using SK_{AB}, and return it in the Authenticate-ACK portion of a ComplexACK-PDU to Client A.

(e) If Client A receives a ComplexACK for the Authenticate service request with the correct 'Modified Random Number', then the message is being executed by Server B.

NOTE: This procedure may also be used by a server to authenticate the receipt of simple or complex acknowledgment by the client.

24.2.4 Message Initiation Authentication

The method used by a BACnet server to authenticate that a message was initiated by the claimed BACnet client is performed by enhancing peer authentication to include a binding to a Transaction State Machine (TSM). This implies that message initiation authentication can be performed only on confirmed service requests. When the message initiation authentication involves the key server, a PK is used to encipher and decipher the information. When neither device involved in the authentication is a key server, an SK is used to encipher and decipher information. The procedure for Server B to authenticate that Client A has initiated a certain message is as follows:

(a) If neither Server B or Client A is a key server, an SK_{AB} must be obtained before authentication can begin.

(b) Server B generates a Authenticate service request and sends it to Client A. The 'Expected Invoke ID' parameter of this message shall contain the InvokeId of the Confirmed Request APDU being authenticated. The Service Request portion of the BACnet-Confirmed-Request-PDU, i.e., the fully formed Authenticate-Request production, is enciphered with either SK_{AB}, or the appropriate PK if either A or B is a key server. The APDU header and all lower layer PCI are left unenciphered.

(c) If Client A can decipher the Authenticate service request, it shall search its TSMs for the Confirmed Request APDU with the 'Expected Invoke ID'. If the TSM is found, Client A shall modify the 'Pseudo Random Number' as specified in the Authenticate service procedure, encipher it as the 'Modified Random Number' using the appropriate key, and return it in the Authenticate-ACK portion of a ComplexACK-PDU to Server B.

(d) If Server B receives a ComplexACK for the Authenticate service request with the correct 'Modified Random Number', then the message was initiated by Client A.

24.2.5 Operator Authentication

BACnet provides a method that allows an operator to log onto a networked device. The procedure of operator authentication is summarized as follows:

(a) The operator logs onto Client A, giving an operator name and password. The only restriction on passwords and names is that they be printable characters. If Client A has the capability, it shall authenticate the password locally. Otherwise, it shall proceed on to the next step.

(b) Client A shall generate an Authenticate service request and transmit it to the key server. The 'Operator Name' and 'Operator Password' parameters shall be included in the Authenticate request. The Service Request portion of the BACnet-Confirmed-Request-PDU, i.e., the fully formed Authenticate-Request production, is enciphered with Client A's PK. The APDU header and all lower layer PCI are left unenciphered.

(c) If the key server can decipher the Authenticate service request, it shall check the 'Operator Name' against the 'Operator Password' to see if they correlate. If the operator is authentic, the key server shall modify the 'Pseudo Random Number' as specified in the Authenticate service procedure, encipher it as the 'Modified Random Number'

using Client A's PK, and return it in the Authenticate-ACK portion of a ComplexACK-PDU to Client A. If not, the key server shall return a 'Result(-)' with the Error portion of the BACnet-Error-PDU enciphered with Client A's PK.

(d) If Client A receives a ComplexACK with the correct 'Modified Random Number', then the operator is authenticated.

24.3 Data Confidentiality Mechanism

The process of ensuring data confidentiality and integrity consists of two parts: obtaining an SK and enciphering data. The process of obtaining an SK is described in 24.2.1. The procedure of encipherment is as follows.

(a) Once client A receives SK_{AB}, encipherment of data between A and B can begin.

(b) Client A requests that an Enciphered Session be started with server B, using the Authenticate service. If the Authenticate service request is successful, the process proceeds to step 3. Otherwise, an Enciphered Session cannot be established.

(c) When A needs to send data to B, A enciphers the data portion of the APDU using the key SK_{AB} and sends the APDU to B. The fixed part of the APDU as well as all lower layer PCI are left unenciphered.

(d) When B receives the data from A, B deciphers the data portion of the APDU using the key SK_{AB}.

(e) When B replies to A, the data portion of the APDU is enciphered using SK_{AB} and the fixed part of the APDU header and all lower layer PCI are left unenciphered.

(f) To terminate the Enciphered Session, either A or B may send an Authenticate service request with 'Start Enciphered Session' set to FALSE.

24.3.1 Requesting an Enciphered Session

The procedure to request an enciphered session is as follows:

(a) Client A generates an Authenticate service request and sends it to server B. The 'Start Enciphered Session' parameter shall be sent and set to TRUE. The Service Request portion of the BACnet-Confirmed-Request-PDU, i.e., the fully formed Authenticate-Request production, is enciphered with SK_{AB}. The APDU header and all lower layer PCI are left unenciphered.

(b) Before processing this message, B shall use the Message Initiation Authentication procedure in 24.2.4 to ensure that A sent the message. If B can decipher the Authenticate request and will accept the enciphered session, it shall modify the 'Pseudo Random Number' as specified in the Authenticate service procedure, encipher it as the 'Modified Random number' using SK_{AB}, and return it in the Authenticate-ACK portion of a ComplexACK-PDU to A. If not, B shall return a 'Result(-)' with the Error portion of the BACnet-Error-PDU enciphered using SK_{AB}.

(c) If A receives a ComplexACK-PDU with the correct 'Modified Random Number', then an enciphered session is started.

24.3.2 Ending an Enciphered Session

The procedure to terminate an enciphered session is as follows:

(a) Device A generates an Authenticate confirmed request and sends it to device B. The 'Start Enciphered Session' parameter shall be included and set to FALSE. The Service Request portion of the BACnet-Confirmed-Request-PDU, i.e., the fully formed Authenticate-Request production, is enciphered with SK_{AB}. The APDU header and all lower layer PCI are left unenciphered.

(b) Before processing this message, B shall use the Message Initiation Authentication procedure in 24.2.4 to ensure that A sent the message. If B can decipher the Authenticate request and will accept the termination of the enciphered session, it shall modify the 'Pseudo Random Number' as specified in the Authenticate service procedure, encipher it as the 'Modified Random Number' using SK_{AB}, and return a ComplexACK-PDU to A. If not, B shall return a 'Result(-)' with the Error portion of the BACnet-Error-PDU enciphered using SK_{AB}.

(c) If Client A receives a ComplexACK-PDU with the the correct 'Modified Random Number', then the enciphered session is terminated.

24.4 RequestKey Service

The RequestKey service is used by a client BACnet protocol user to request that an SK be set up between itself and another BACnet protocol user. To ensure data integrity, the Service Request portion of the BACnet-Confirmed-Request-PDU, i.e., the fully formed RequestKey-Request production, shall be enciphered using the requesting client's PK. The APDU header and all lower layer PCI are left unenciphered. Note that the use of this service may require extending the originating device's APDU_Timeout value to accommodate the time required to deliver an authenticated session key to both key server clients.

24.4.1 Structure

The structure of the RequestKey service primitives is shown in Table 24-1.

Table 24-1. Structure of RequestKey Service Primitives

Parameter Name	Req	Ind	Rsp	Cnf
Argument	M	M(=)		
Requesting Device Identifier	M	M(=)		
Requesting Device Address	M	M(=)		
Remote Device Identifier	M	M(=)		
Remote Device Address	M	M(=)		
Result(+)			S	S(=)
Result(-)			S	S(=)
Error Type			M	M(=)

24.4.1.1 Argument

This parameter shall convey the parameters for the RequestKey confirmed service request.

24.4.1.1.1 Requesting Device Identifier

This parameter, of type BACnetObjectIdentifier, shall convey the identity of the client requesting that an SK be generated. Since this argument is protected by encipherment, its integrity is assured.

24.4.1.1.2 Requesting Device Address

This parameter, of type BACnetAddress, shall convey the address of the client requesting that an SK be generated. Since this argument is protected by encipherment, its integrity is assured.

24.4.1.1.3 Remote Device Identifier

This parameter, of type BACnetObjectIdentifier, shall convey the identity of the associated remote device that is receiving an SK. Since this argument is protected by encipherment, its integrity is assured.

24.4.1.1.4 Remote Device Address

This parameter, of type BACnetAddress, shall convey the address of the associated remote device that is receiving an SK. Since this argument is protected by encipherment, its integrity is assured.

24.4.1.2 Result(+)

The 'Result(+)' shall indicate that the service request succeeded. The security policy of authenticating that the requester does receive the 'Result(+)' is a local matter. If authentication of this message is desired, Message Execution Authentication shall be used.

24.4.1.3 Result(-)

The 'Result(-)' parameter shall indicate that the service request has failed in its entirety. The reason for failure shall be specified by the 'Error Type' parameter. To ensure data integrity, the 'Error Type' shall be protected by encipherment using the client's PK.

24.4.1.3.1 Error Type

This parameter consists of two component parameters: (1) an 'Error Class' and (2) an 'Error Code'. The Error Class shall be SECURITY. Error Codes are defined in 18.5.

24.4.2 Service Procedure

The service provider (the key server) shall authenticate that this request was initiated by the requesting device using the Message Initiation Authentication procedure in 24.2.4 before proceeding. After deciphering and verifying the validity of the request, the key server shall generate the requested SK. Once this SK is generated, the key server shall deliver it to the Remote Device and then to the Requesting Device by writing to the List_Of_Session_Keys property of the Device object using the AddListElement service. The List_Of_Session_Keys property element shall be enciphered using the PK of the recipient. If this succeeds, the key server shall return a 'Result(+)' for this service. If this fails, the key server shall return a 'Result(-)' for this service. The arguments of the 'Result(-)' shall be enciphered with the requesting device's PK.

24.5 Authenticate Service

The Authenticate service is used by a client protocol user to authenticate a peer device, initiating device, executing device, or human operator. To ensure data entegrity, the Service Request portion of the BACnet-Confirmed-Request-PDU, i.e., the fully formed Authenticate-Request production, shall be enciphered. If the transaction is occurring between the client and the key server, the client's PK shall be used. If the transaction is between non-key server devices, the corresponding SK shall be used to encipher and decipher the parameters.

24.5.1 Structure

The structure of the Authenticate service primitives is shown in Table 24-2.

Table 24-2. Structure of Authenticate Primitives

Parameter Name	Req	Ind	Rsp	Cnf
Argument	M	M(=)		
Pseudo Random Number	M	M(=)		
Expected Invoke ID	U	U(=)		
Operator Name	U	U(=)		
Operator Password	C	C(=)		
Start Enciphered Session	U	U(=)		
Result(+)			S	S(=)
Modified Random Number			M	M(=)
Result(-)			S	S(=)
Error Type			M	M(=)

24.5.1.1 Argument

This parameter shall convey the parameters for the Authenticate service request.

24.5.1.1.1 Pseudo Random Number

This parameter, of type Unsigned32, is a pseudo random number generated by the service requester. It is used as the seed for the 'Result(+)' argument called Modified Random Number.

24.5.1.1.2 Expected Invoke ID

This parameter, of type Unsigned8, is an optional parameter used to perform Message Initiator Authentication and Message Execution Authentication. It shall be the 'Invoke Id' of the message being authenticated.

24.5.1.1.3 Operator Name

This parameter, of type CharacterString, is an optional parameter used to perform Operator Authentication. If provided, it shall indicate the name of the operator requesting access.

24.5.1.1.4 Operator Password

This parameter, of type CharacterString, is a conditional parameter used to perform Operator Authentication. It shall be provided when the 'Operator Name' parameter is present. It shall indicate the password of the operator requesting access. The 'Operator Password' shall have a length of no more than 20 characters.

24.5.1.1.5 Start Enciphered Session

This optional parameter, of type BOOLEAN, indicates that the requesting BACnet-user is attempting to either start (TRUE) or terminate (FALSE) an enciphered session. If this parameter is absent, an enciphered session is not being established or terminated.

24.5.1.2 Result(+)

The 'Result(+)' shall indicate that the service request succeeded. The security policy of authenticating that the requesting BACnet-user does receive the 'Result(+)' is a local matter. If authentication of this message is desired, Message Execution Authentication shall be used. The 'Result(+)' primitive returns a Modified Random Number as defined in 24.5.2.

24.5.1.3 Result(-)

The 'Result(-)' parameter shall indicate that the service request has failed in its entirety. The reason for failure shall be specified by the 'Error Type' parameter. To ensure data integrity, the 'Error Type' shall be protected by encipherment using the client's PK.

24.5.1.3.1 Error Type

This parameter consists of two component parameters: (1) an 'Error Class' and (2) an 'Error Code'. The Error Class shall be SECURITY. Error codes are defined in 18.5.

24.5.2 Service Procedure

After deciphering and verifying the validity of the request, the service provider shall perform either

(a) Message Execution Authentication;

(b) Message Initiation Authentication;

(c) Operator Authentication; or

(d) Enciphered Session Determination.

In the event that the authentication succeeds, the service provider shall generate the Modified Random Number by inverting the most significant and least significant two bits of every octet of the received Pseudo Random Number and return it in the 'Result(+)' primitive. Otherwise a 'Result(-)' shall be issued. In either case, the parameters of the APDU shall be enciphered. If the transaction is occ+urring between the client and the key server, the client's PK shall be used. If the transaction is between non-key-server clients, the corresponding SK shall be used to encipher and decipher the parameters.

24.5.2.1 Message Execution Authentication

The process of message execution authentication consists of the following:

(a) Decode the Authenticate APDU and start a transaction state machine (TSM) for the 'Expected Invoke ID'.

(b) Wait up to 30 seconds for the receipt of another APDU with the 'Expected Invoke ID' from the requester.

(c) If an APDU with the 'Expected Invoke ID' is received, then a 'Result(+)' shall be generated and returned to the requester. If not, then a 'Result(-)' shall be generated.

24.5.2.2 Message Initiation Authentication

The process of message initiation authentication consists of the following:

(a) Decode the Authenticate APDU and determine if a transaction state machine (TSM) for the 'Expected Invoke ID' exists to the requester.

(b) If an APDU with the 'Expected Invoke ID' is outstanding, then a 'Result(+)'shall be generated and returned to the requester. If not, then a 'Result(-)' shall be generated.

24.5.2.3 Operator Authentication

The process of operator authentication consists of the following:

(a) Decode the Authenticate APDU and determine if a valid operator can be associated with the 'Operator Name' parameter of the APDU. If not, a negative result shall be returned to the requester.

(b) If an operator exists, the 'Operator Password' shall be validated. If the password is valid, a 'Result(+)' shall be returned to the requester. If not, a 'Result(-)' shall be returned to the requester.

24.5.2.4 Enciphered Session Determination

The process of enciphered session determination consists of the following:

(a) Decode the Authenticate APDU and send a Message Initiation Authentication request to the requester.

(b) If the requester responds to the Message Initiation Authentication affirmatively, then the executing device shall determine if the request to either start or end an enciphered session can be accommodated. If the request can be accommodated, a 'Result(+)' shall be sent. If not, a 'Result(-)' shall be sent.

25 REFERENCES

ANSI/ATA 878.1 (1999), *ARCNET Local Area Network Standard*.

ANSI/EIA/CEA-709.1-B (2002), *Control Network Protocol Specification.*

ANSI/IEEE Standard 754 (1985), *IEEE Standard for Binary Floating-Point Arithmetic*.

ANSI/INCITS 92-1981 (R1998), (formerly ANSI X3.92-1981), *Data Encryption Algorithm.*

ANSI/INCITS X3.4-1986 (R1997), *Information Processing – Coded Character Sets – 7-Bit American National Standard Code for Information Interchange (7-bit ASCII).*

ANSI/TIA/EIA-232-F-1997 (R2002), *Interface Between Data Terminal Equipment and Data Communication Equipment Employing Serial Binary Data Interchange*.

ANSI/TIA/EIA-485-A-1998 (R2003), *Standard for Electrical Characteristics of Generators and Receivers for Use in Balanced Digital Multipoint Systems.*

ANSI X3.41-1974 (R1990), *American National Standard Code Extension Techniques for Use with the 7-bit Coded Character Set of American National Standard Code for Information Interchange.*

DDN Protocol Handbook, Volumes 1-3, NIC 50004, 50005, and 50006.

Echelon, *LonMark® Layer 1-6 Interoperability Guidelines Version 3.3.*

FIPS 46-2 (1993), *Federal Information Processing Standards - Data Encryption Standard*.

ISO 7498 (1984), *Information processing systems - Open Systems Interconnection - Basic Reference Model*.

ISO TR 8509 (1987), *Information processing systems - Open Systems Interconnection - service conventions.*

ISO 8649 (1988), *Information processing systems – Open Systems Interconnection - Service definition for the Association Control Service Element.*

ISO 8802-2 (1998), *Information processing systems - Local area networks - Part 2: Logical link control*.

ISO/IEC 8802-3 (2000), *Information processing systems - Local area networks - Part 3: Carrier sense multiple access with collision detection (CSMA/CD) access method and physical layer specifications.*

ISO 8822 (1994), *Information processing systems - Open Systems Interconnection - Connection-oriented presentation service definition.*

ISO/IEC 8824 (1990), *Information technology - Open Systems Interconnection - Specification of Abstract Syntax Notation One (ASN.1).*

ISO/IEC 8825 (1990), *Information technology - Open Systems Interconnection - Specification of Basic Encoding Rules for Abstract Syntax Notation One (ASN.1).*

ISO 9545 (1994), *Information processing systems - Open Systems Interconnection - Application Layer Structure (ALS).*

ISO/IEC 10646-1 (2000), *IT - Universal Multiple-Octet Coded Character Set (UCS) - Part 1: Architecture and Basic Multilingual Plane.*

JIS C 6226 (1983), *Code of the Japanese Graphic Character Set for Information Interchange*. Japan Institute for Standardization.

Konnex Association, *Konnex Handbook Volume 3: System Specifications.*

Konnex Association, *Konnex Handbook Volume 3: System Specifications, Part 7: Interworking, Chapter 2: Datapoint Types.*

Konnex Association, *Konnex Handbook Volume 3: System Specifications, Part 7: Interworking, Chapter 3: Standard Identifier Tables, Annex 1 – Property Identifiers.*

Konnex Association, *Konnex Handbook Volume 7: Applications Descriptions.*

Sources for Reference Material

ANSI: American National Standards Institute, 25 West 43rd St., 4th Floor, New York , NY 10036.

DDN: Available from the Defense Data Network Information Center, SRI International, 333 Ravenswood Ave., Room EJ291, Menlo Park, CA 94025.

Echelon: Echelon Corporation, 550 Meridian Ave., San Jose, CA 95126.

EIA: Electronics Industries Alliance, 2500 Wilson Blvd. Arington, VA 22201.

EIBA: EIB Association (EIBA) s.c.r.l., Neerveldstraat / Rue de Neerveld 105, B-1200 Brussels, Belgium

FIPS: National Institute of Standards and Technology, Gaithersburg, MD 20899.

IEEE: The Institute of Electrical and Electronics Engineers, Inc., 3 Park Ave., 17th Floor, New York, NY 10016.

INCITS: The International Committee for Information Technology Standards (INCITS) is sponsored by the Information Technology Industry Council (ITI), 1250 Eye St. NW, Suite 200, Washington, DC 20005.

ISO: Available from ANSI.

JIS: Available from ANSI.

Konnex Association: Neerveldstraat / Rue de Neerveld 105, B-1200 Brussels, Belgium

LonMark International, 550 Meridian Avenue, San Jose, CA 95126.

ANNEX A - PROTOCOL IMPLEMENTATION CONFORMANCE STATEMENT (NORMATIVE)

(This annex is part of this Standard and is required for its use.)

BACnet Protocol Implementation Conformance Statement

Date: _____
Vendor Name: _____
Product Name: _____
Product Model Number: _____
Applications Software Version: _____ **Firmware Revision:** _____ **BACnet Protocol Revision:** _____

Product Description:

_____
_____
_____
_____
_____
_____
_____
_____

BACnet Standardized Device Profile (Annex L):

☐ **BACnet Operator Workstation (B-OWS)**
☐ **BACnet Building Controller (B-BC)**
☐ **BACnet Advanced Application Controller (B-AAC)**
☐ **BACnet Application Specific Controller (B-ASC)**
☐ **BACnet Smart Sensor (B-SS)**
☐ **BACnet Smart Actuator (B-SA)**

List all BACnet Interoperability Building Blocks Supported (Annex K): _____

_____
_____

Segmentation Capability:

☐ Able to transmit segmented messages Window Size _____
☐ Able to receive segmented messages Window Size _____

Standard Object Types Supported:

An object type is supported if it may be present in the device. For each standard Object Type supported provide the following data:

1) Whether objects of this type are dynamically creatable using the CreateObject service
2) Whether objects of this type are dynamically deletable using the DeleteObject service
3) List of the optional properties supported
4) List of all properties that are writable where not otherwise required by this standard
5) List of proprietary properties and for each its property identifier, datatype, and meaning
6) List of any property range restrictions

Data Link Layer Options:

☐ BACnet IP, (Annex J)
☐ BACnet IP, (Annex J), Foreign Device
☐ ISO 8802-3, Ethernet (Clause 7)
☐ ANSI/ATA 878.1, 2.5 Mb. ARCNET (Clause 8)
☐ ANSI/ATA 878.1, EIA-485 ARCNET (Clause 8), baud rate(s) _____
☐ MS/TP master (Clause 9), baud rate(s): _____
☐ MS/TP slave (Clause 9), baud rate(s): _____
☐ Point-To-Point, EIA 232 (Clause 10), baud rate(s):_____
☐ Point-To-Point, modem, (Clause 10), baud rate(s): _____
☐ LonTalk, (Clause 11), medium: _____
☐ Other: _____

Device Address Binding:

Is static device binding supported? (This is currently necessary for two-way communication with MS/TP slaves and certain other devices.) ☐ Yes ☐ No

Networking Options:

☐ Router, Clause 6 - List all routing configurations, e.g., ARCNET-Ethernet, Ethernet-MS/TP, etc.
☐ Annex H, BACnet Tunneling Router over IP
☐ BACnet/IP Broadcast Management Device (BBMD)
Does the BBMD support registrations by Foreign Devices? ☐ Yes ☐ No

Character Sets Supported:

Indicating support for multiple character sets does not imply that they can all be supported simultaneously.

☐ ANSI X3.4	☐ IBM™/Microsoft™ DBCS	☐ ISO 8859-1
☐ ISO 10646 (UCS-2)	☐ ISO 10646 (UCS-4)	☐ JIS C 6226

If this product is a communication gateway, describe the types of non-BACnet equipment/networks(s) that the gateway supports:

_____
_____
_____

ANNEX B - GUIDE TO SPECIFYING BACnet DEVICES (INFORMATIVE)

(This annex is not part of this standard but is included for informative purposes only.)

The BIBBs (Annex K) and standardized BACnet device profiles (Annex L) are intended to be a useful tool for people who design, specify, or operate building automation systems that contain BACnet devices. This classification approach is a compromise between two conflicting goals. The first goal is to promote interoperability by limiting the various combinations of BACnet object types and services that can be supported and still conform to this standard. The other goal is to avoid unnecessarily restricting manufacturers of BACnet devices in the sense that they would be required to provide BACnet functionality that would never be used by a device except to meet a conformance requirement. Maximum interoperability would be achieved by requiring all BACnet devices to support exactly the same combination of standard object types and application services. On the other hand, complete flexibility for manufacturers would inevitably lead to such widespread variation in the particular object types and application services that are supported that many devices would only partially interoperate. Interoperability would be limited to the intersection of the application services and object types supported by the devices.

The idea behind the BIBB and device profile model is to combine the portions of the BACnet protocol that are needed to perform particular functions, and to identify those functions that a system designer would expect in a certain type of device. When designing or specifying BACnet devices for an automation system, it is appropriate to specify the device profile that best meets the needs of the application and any additional BIBBs that are also required. Devices can be expected to interoperate with respect to a given BIBB so long as one device implements the A-side functionality and the other device implements the B-side functionality.

A particular manufacturer may decide to build a product that supports more BIBBs than required by its device profile. This can be determined from the PICS.

ANNEX C - FORMAL DESCRIPTION OF OBJECT TYPE STRUCTURES (INFORMATIVE)

(This annex is not part of this standard but is included for informative purposes only.)

ACCUMULATOR ::= SEQUENCE {

object-identifier	[75]	BACnetObjectIdentifier,
object-name	[77]	CharacterString,
object-type	[79]	BACnetObjectType,
present-value	[85]	Unsigned,
description	[28]	CharacterString OPTIONAL,
device-type	[31]	CharacterString OPTIONAL,
status-flags	[111]	BACnetStatusFlags,
event-state	[36]	BACnetEventState,
reliability	[103]	BACnetReliability OPTIONAL,
out-of-service	[81]	BOOLEAN,
scale	[186]	BACnetScale,
units	[117]	BACnetEngineeringUnits,
prescale	[188]	BACnetPrescale OPTIONAL,
max-pres-value	[65]	Unsigned,
value-change-time	[192]	BACnetDateTime OPTIONAL,
value-before-change	[190]	Unsigned OPTIONAL,
value-set	[191]	Unsigned OPTIONAL,
logging-record	[184]	BACnetAccumulatorRecord OPTIONAL,
logging-object	[183]	BACnetObjectIdentifier OPTIONAL,
pulse-rate	[186]	Unsigned OPTIONAL,
high-limit	[45]	Unsigned OPTIONAL,
low-limit	[59]	Unsigned OPTIONAL,
limit-monitoring-interval	[182]	Unsigned OPTIONAL,
notification-class	[17]	Unsigned OPTIONAL,
time-delay	[113]	Unsigned OPTIONAL,
limit-enable	[52]	BACnetLimitEnable OPTIONAL,
event-enable	[35]	BACnetEventTransitionBits OPTIONAL,
acked-transitions	[0]	BACnetEventTransitionBits OPTIONAL,
notify-type	[72]	BACnetNotifyType OPTIONAL,
event-time-stamps	[130]	SEQUENCE OF BACnetTimeStamp OPTIONAL,
profile-name	[167]	CharacterString OPTIONAL

}

ANALOG-INPUT ::= SEQUENCE {

object-identifier	[75]	BACnetObjectIdentifier,
object-name	[77]	CharacterString,
object-type	[79]	BACnetObjectType,
present-value	[85]	REAL,
description	[28]	CharacterString OPTIONAL,
device-type	[31]	CharacterString OPTIONAL,
status-flags	[111]	BACnetStatusFlags,
event-state	[36]	BACnetEventState,
reliability	[103]	BACnetReliability OPTIONAL,
out-of-service	[81]	BOOLEAN,
update-interval	[118]	Unsigned OPTIONAL,
units	[117]	BACnetEngineeringUnits,
min-pres-value	[69]	REAL OPTIONAL,
max-pres-value	[65]	REAL OPTIONAL,
resolution	[106]	REAL OPTIONAL,
cov-increment	[22]	REAL OPTIONAL,
time-delay	[113]	Unsigned OPTIONAL,

notification-class	[17]	Unsigned OPTIONAL,
high-limit	[45]	REAL OPTIONAL,
low-limit	[59]	REAL OPTIONAL,
deadband	[25]	REAL OPTIONAL,
limit-enable	[52]	BACnetLimitEnable OPTIONAL,
event-enable	[35]	BACnetEventTransitionBits OPTIONAL,
acked-transitions	[0]	BACnetEventTransitionBits OPTIONAL,
notify-type	[72]	BACnetNotifyType OPTIONAL,
event-time-stamps	[130]	SEQUENCE OF BACnetTimeStamp OPTIONAL,
		-- accessed as a BACnetARRAY
profile-name	[168]	CharacterString OPTIONAL

}

ANALOG-OUTPUT ::= SEQUENCE {

object-identifier	[75]	BACnetObjectIdentifier,
object-name	[77]	CharacterString,
object-type	[79]	BACnetObjectType,
present-value	[85]	REAL,
description	[28]	CharacterString OPTIONAL,
device-type	[31]	CharacterString OPTIONAL,
status-flags	[111]	BACnetStatusFlags,
event-state	[36]	BACnetEventState,
reliability	[103]	BACnetReliability OPTIONAL,
out-of-service	[81]	BOOLEAN,
units	[117]	BACnetEngineeringUnits,
min-pres-value	[69]	REAL OPTIONAL,
max-pres-value	[65]	REAL OPTIONAL,
resolution	[106]	REAL OPTIONAL,
priority-array	[87]	BACnetPriorityArray,
relinquish-default	[104]	REAL,
cov-increment	[22]	REAL OPTIONAL,
time-delay	[113]	Unsigned OPTIONAL,
notification-class	[17]	Unsigned OPTIONAL,
high-limit	[45]	REAL OPTIONAL,
low-limit	[59]	REAL OPTIONAL,
deadband	[25]	REAL OPTIONAL,
limit-enable	[52]	BACnetLimitEnable OPTIONAL,
event-enable	[35]	BACnetEventTransitionBits OPTIONAL,
acked-transitions	[0]	BACnetEventTransitionBits OPTIONAL,
notify-type	[72]	BACnetNotifyType OPTIONAL,
event-time-stamps	[130]	SEQUENCE OF BACnetTimeStamp OPTIONAL,
		-- accessed as a BACnetARRAY
profile-name	[168]	CharacterString OPTIONAL

}

ANALOG-VALUE ::= SEQUENCE {

object-identifier	[75]	BACnetObjectIdentifier,
object-name	[77]	CharacterString,
object-type	[79]	BACnetObjectType,
present-value	[85]	REAL,
description	[28]	CharacterString OPTIONAL,
status-flags	[111]	BACnetStatusFlags,
event-state	[36]	BACnetEventState,
reliability	[103]	BACnetReliability OPTIONAL,
out-of-service	[81]	BOOLEAN,
units	[117]	BACnetEngineeringUnits,

priority-array	[87]	BACnetPriorityArray OPTIONAL,
relinquish-default	[104]	REAL OPTIONAL,
cov-increment	[22]	REAL OPTIONAL,
time-delay	[113]	Unsigned OPTIONAL,
notification-class	[17]	Unsigned OPTIONAL,
high-limit	[45]	REAL OPTIONAL,
low-limit	[59]	REAL OPTIONAL,
deadband	[25]	REAL OPTIONAL,
limit-enable	[52]	BACnetLimitEnable OPTIONAL,
event-enable	[35]	BACnetEventTransitionBits OPTIONAL,
acked-transitions	[0]	BACnetEventTransitionBits OPTIONAL,
notify-type	[72]	BACnetNotifyType OPTIONAL,
event-time-stamps	[130]	SEQUENCE OF BACnetTimeStamp OPTIONAL,
		-- accessed as a BACnetARRAY
profile-name	[168]	CharacterString OPTIONAL

}

AVERAGING ::= SEQUENCE {

object-identifier	[75]	BACnetObjectIdentifier,
object-name	[77]	CharacterString,
object-type	[79]	BACnetObjectType,
minimum-value	[136]	REAL,
minimum-value-timestamp	[150]	BACnetDateTime OPTIONAL,
average-value	[125]	REAL,
variance-value	[151]	REAL OPTIONAL,
maximum-value	[135]	REAL,
maximum-value-timestamp	[149]	BACnetDateTime OPTIONAL,
description	[28]	CharacterString OPTIONAL,
attempted-samples	[124]	Unsigned,
valid-samples	[146]	Unsigned,
object-property-reference	[78]	BACnetDeviceObjectPropertyReference,
window-interval	[147]	Unsigned,
window-samples	[148]	Unsigned,
profile-name	[168]	CharacterString OPTIONAL

}

BINARY-INPUT ::= SEQUENCE {

object-identifier	[75]	BACnetObjectIdentifier,
object-name	[77]	CharacterString,
object-type	[79]	BACnetObjectType,
present-value	[85]	BACnetBinaryPV,
description	[28]	CharacterString OPTIONAL,
device-type	[31]	CharacterString OPTIONAL,
status-flags	[111]	BACnetStatusFlags,
event-state	[36]	BACnetEventState,
reliability	[103]	BACnetReliability OPTIONAL,
out-of-service	[81]	BOOLEAN,
polarity	[84]	BACnetPolarity,
inactive-text	[46]	CharacterString OPTIONAL,
active-text	[4]	CharacterString OPTIONAL,
change-of-state-time	[16]	BACnetDateTime OPTIONAL,
change-of-state-count	[15]	Unsigned OPTIONAL,
time-of-state-count-reset	[115]	BACnetDateTime OPTIONAL,
elapsed-active-time	[33]	Unsigned32 OPTIONAL,
time-of-active-time-reset	[114]	BACnetDateTime OPTIONAL,
time-delay	[113]	Unsigned OPTIONAL,

```
        notification-class          [17]    Unsigned OPTIONAL,
        alarm-value                 [6]     BACnetBinaryPV OPTIONAL,
        event-enable                [35]    BACnetEventTransitionBits OPTIONAL,
        acked-transitions           [0]     BACnetEventTransitionBits OPTIONAL,
        notify-type                 [72]    BACnetNotifyType OPTIONAL,
        event-time-stamps           [130]   SEQUENCE OF BACnetTimeStamp OPTIONAL,
                                            -- accessed as a BACnetARRAY
        profile-name                [168]   CharacterString OPTIONAL
        }

BINARY-OUTPUT ::= SEQUENCE {
        object-identifier           [75]    BACnetObjectIdentifier,
        object-name                 [77]    CharacterString,
        object-type                 [79]    BACnetObjectType,
        present-value               [85]    BACnetBinaryPV,
        description                 [28]    CharacterString OPTIONAL,
        device-type                 [31]    CharacterString OPTIONAL,
        status-flags                [111]   BACnetStatusFlags,
        event-state                 [36]    BACnetEventState,
        reliability                 [103]   BACnetReliability OPTIONAL,
        out-of-service              [81]    BOOLEAN,
        polarity                    [84]    BACnetPolarity,
        inactive-text               [46]    CharacterString OPTIONAL,
        active-text                 [4]     CharacterString OPTIONAL,
        change-of-state-time        [16]    BACnetDateTime OPTIONAL,
        change-of-state-count       [15]    Unsigned OPTIONAL,
        time-of-state-count-reset   [115]   BACnetDateTime OPTIONAL,
        elapsed-active-time         [33]    Unsigned32 OPTIONAL,
        time-of-active-time-reset   [114]   BACnetDateTime OPTIONAL,
        minimum-off-time            [66]    Unsigned32 OPTIONAL,
        minimum-on-time             [67]    Unsigned32 OPTIONAL,
        priority-array              [87]    BACnetPriorityArray,
        relinquish-default          [104]   BACnetBinaryPV,
        time-delay                  [113]   Unsigned OPTIONAL,
        notification-class          [17]    Unsigned OPTIONAL,
        feedback-value              [40]    BACnetBinaryPV OPTIONAL,
        event-enable                [35]    BACnetEventTransitionBits OPTIONAL,
        acked-transitions           [0]     BACnetEventTransitionBits OPTIONAL,
        notify-type                 [72]    BACnetNotifyType OPTIONAL,
        event-time-stamps           [130]   SEQUENCE OF BACnetTimeStamp OPTIONAL,
                                            -- accessed as a BACnetARRAY
        profile-name                [168]   CharacterString OPTIONAL
        }

BINARY-VALUE ::= SEQUENCE {
        object-identifier           [75]    BACnetObjectIdentifier,
        object-name                 [77]    CharacterString,
        object-type                 [79]    BACnetObjectType,
        present-value               [85]    BACnetBinaryPV,
        description                 [28]    CharacterString OPTIONAL,
        status-flags                [111]   BACnetStatusFlags,
        event-state                 [36]    BACnetEventState,
        reliability                 [103]   BACnetReliability OPTIONAL,
        out-of-service              [81]    BOOLEAN,
        inactive-text               [46]    CharacterString OPTIONAL,
        active-text                 [4]     CharacterString OPTIONAL,
```

```
        change-of-state-time        [16]    BACnetDateTime OPTIONAL,
        change-of-state-count       [15]    Unsigned OPTIONAL,
        time-of-state-count-reset   [115]   BACnetDateTime OPTIONAL,
        elapsed-active-time         [33]    Unsigned32 OPTIONAL,
        time-of-active-time-reset   [114]   BACnetDateTime OPTIONAL,
        minimum-off-time            [66]    Unsigned32 OPTIONAL,
        minimum-on-time             [67]    Unsigned32 OPTIONAL,
        priority-array              [87]    BACnetPriorityArray OPTIONAL,
        relinquish-default          [104]   BACnetBinaryPV OPTIONAL,
        time-delay                  [113]   Unsigned OPTIONAL,
        notification-class          [17]    Unsigned OPTIONAL,
        alarm-value                 [6]     BACnetBinaryPV OPTIONAL,
        event-enable                [35]    BACnetEventTransitionBits OPTIONAL,
        acked-transitions           [0]     BACnetEventTransitionBits OPTIONAL,
        notify-type                 [72]    BACnetNotifyType OPTIONAL,
        event-time-stamps           [130]   SEQUENCE OF BACnetTimeStamp OPTIONAL,
                                            -- accessed as a BACnetARRAY
        profile-name                [168]   CharacterString OPTIONAL
        }

CALENDAR ::= SEQUENCE {
        object-identifier           [75]    BACnetObjectIdentifier,
        object-name                 [77]    CharacterString,
        object-type                 [79]    BACnetObjectType,
        description                 [28]    CharacterString OPTIONAL,
        present-value               [85]    BOOLEAN,
        date-list                   [23]    SEQUENCE OF BACnetCalendarEntry,
        profile-name                [168]   CharacterString OPTIONAL
        }

COMMAND ::= SEQUENCE {
        object-identifier           [75]    BACnetObjectIdentifier,
        object-name                 [77]    CharacterString,
        object-type                 [79]    BACnetObjectType,
        description                 [28]    CharacterString OPTIONAL,
        present-value               [85]    Unsigned,
        in-process                  [47]    BOOLEAN,
        all-writes-successful       [9]     BOOLEAN,
        action                      [2]     SEQUENCE OF BACnetActionList, -- accessed as a BACnetARRAY
        action-text                 [3]     SEQUENCE OF CharacterString OPTIONAL,
                                            -- accessed as a BACnetARRAY
        profile-name                [168]   CharacterString OPTIONAL
        }

DEVICE ::= SEQUENCE {
        object-identifier           [75]    BACnetObjectIdentifier,
        object-name                 [77]    CharacterString,
        object-type                 [79]    BACnetObjectType,
        system-status               [112]   BACnetDeviceStatus,
        vendor-name                 [121]   CharacterString,
        vendor-identifier           [120]   Unsigned16,
        model-name                  [70]    CharacterString,
        firmware-revision           [44]    CharacterString,
        application-software-version [12]   CharacterString,
        location                    [58]    CharacterString OPTIONAL,
        description                 [28]    CharacterString OPTIONAL,
```

```
        protocol-version                [98]  Unsigned,
        protocol-revision               [139] Unsigned,
        protocol-services-supported     [97]  BACnetServicesSupported,
        protocol-object-types-supported [96]  BACnetObjectTypesSupported,
        object-list                     [76]  SEQUENCE OF BACnetObjectIdentifier, -- accessed as a BACnetARRAY
        max-APDU-length-supported       [62]  Unsigned,
        segmentation-supported          [107] BACnetSegmentation,
        vt-classes-supported            [122] SEQUENCE OF BACnetVTClass OPTIONAL,
        active-vt-sessions              [5]   SEQUENCE OF BACnetVTSession OPTIONAL,
        local-time                      [57]  Time OPTIONAL,
        local-date                      [56]  Date OPTIONAL,
        utc-offset                      [119] INTEGER OPTIONAL,
        daylight-savings-status         [24]  BOOLEAN OPTIONAL,
        apdu-segment-timeout            [10]  Unsigned,
        apdu-timeout                    [11]  Unsigned,
        number-of-APDU-retries          [73]  Unsigned,
        list-of-session-keys            [55]  SEQUENCE OF BACnetSessionKey OPTIONAL,
        time-synchronization-recipients [116] SEQUENCE OF BACnetRecipient OPTIONAL, -- required for time master
        max-master                      [64]  Unsigned(1..127) OPTIONAL, -- required for MS/TP master, see 12.11
        max-info-frames                 [63]  Unsigned OPTIONAL, -- required for MS/TP master, see 12.11
        device-address-binding          [30]  SEQUENCE OF BACnetAddressBinding,
        database-revision               [155] Unsigned,
        configuration-files             [154] SEQUENCE OF BACnetObjectIdentifier,
        last-restore-time               [157] BACnetTimeStamp,
        backup-failure-timeout          [153] Unsigned16,
        active-cov-subscriptions        [152] SEQUENCE OF BACnetCOVSubscription,
        max-segments-accepted           [167] Unsigned,
        slave-proxy-enable              [172] SEQUENCE OF BOOLEAN OPTIONAL,
        auto-slave-discovery            [169] SEQUENCE OF BOOLEAN OPTIONAL,
        slave-address-binding           [171] SEQUENCE OF BACnetAddressBinding OPTIONAL,
        manual-slave-address-binding    [170] SEQUENCE OF BACnetAddressBinding OPTIONAL,
        profile-name                    [168] CharacterString OPTIONAL
        }

EVENT-ENROLLMENT ::= SEQUENCE {
        object-identifier               [75]  BACnetObjectIdentifier,
        object-name                     [77]  CharacterString,
        object-type                     [79]  BACnetObjectType,
        description                     [28]  CharacterString OPTIONAL,
        event-type                      [37]  BACnetEventType,
        notify-type                     [72]  BACnetNotifyType,
        event-parameters                [83]  BACnetEventParameter,
        object-property-reference       [78]  BACnetDeviceObjectPropertyReference,
        event-state                     [36]  BACnetEventState,
        event-enable                    [35]  BACnetEventTransitionBits,
        acked-transitions               [0]   BACnetEventTransitionBits,
        notification-class              [17]  Unsigned OPTIONAL,
        event-time-stamps               [130] SEQUENCE OF BACnetTimeStamp,  -- accessed as a BACnetARRAY
        profile-name                    [168] CharacterString OPTIONAL
        }

FILE ::= SEQUENCE {
        object-identifier               [75]  BACnetObjectIdentifier,
        object-name                     [77]  CharacterString,
        object-type                     [79]  BACnetObjectType,
        description                     [28]  CharacterString OPTIONAL,
```

```
        file-type                   [43]    CharacterString,
        file-size                   [42]    Unsigned,
        modification-date           [71]    BACnetDateTime,
        archive                     [13]    BOOLEAN,
        read-only                   [99]    BOOLEAN,
        file-access-method          [41]    BACnetFileAccessMethod,
        record-count                [141]   Unsigned OPTIONAL,
        profile-name                [168]   CharacterString OPTIONAL
        }

GROUP ::= SEQUENCE {
        object-identifier           [75]    BACnetObjectIdentifier,
        object-name                 [77]    CharacterString,
        object-type                 [79]    BACnetObjectType,
        description                 [28]    CharacterString OPTIONAL,
        list-of-group-members       [53]    SEQUENCE OF ReadAccessSpecification,
        present-value               [85]    SEQUENCE OF ReadAccessResult,
        profile-name                [168]   CharacterString OPTIONAL
        }

LIFE-SAFETY-POINT ::= SEQUENCE {
        object-identifier           [75]    BACnetObjectIdentifier,
        object-name                 [77]    CharacterString,
        object-type                 [79]    BACnetObjectType,
        present-value               [85]    BACnetLifeSafetyState,
        tracking-value              [164]   BACnetLifeSafetyState OPTIONAL,
        description                 [28]    CharacterString OPTIONAL,
        device-type                 [31]    CharacterString OPTIONAL,
        status-flags                [111]   BACnetStatusFlags,
        event-state                 [36]    BACnetEventState,
        reliability                 [103]   BACnetReliability,
        out-of-service              [81]    BOOLEAN,
        mode                        [160]   BACnetLifeSafetyMode,
        accepted-modes              [175]   SEQUENCE OF BACnetLifeSafetyMode,
        time-delay                  [113]   Unsigned OPTIONAL,
        notification-class          [17]    Unsigned OPTIONAL,
        life-safety-alarm-values    [166]   SEQUENCE OF BACnetLifeSafetyState OPTIONAL,
        alarm-values                [7]     SEQUENCE OF BACnetLifeSafetyState OPTIONAL,
        fault-values                [39]    SEQUENCE OF BACnetLifeSafetyState OPTIONAL,
        event-enable                [35]    BACnetEventTransitionBits OPTIONAL,
        acked-transitions           [0]     BACnetEventTransitionBits OPTIONAL,
        notify-type                 [72]    BACnetNotifyType OPTIONAL,
        event-time-stamps           [130]   SEQUENCE OF BACnetTimeStamp OPTIONAL,
                                            --accessed as a BACnetARRAY
        silenced                    [163]   BACnetSilencedState,
        operation-expected          [161]   BACnetLifeSafetyOperation,
        maintenance-required        [158]   BACnetMaintenance OPTIONAL,
        setting                     [162]   Unsigned8 OPTIONAL,
        direct-reading              [156]   REAL OPTIONAL,
        units                       [117]   BACnetEngineeringUnits OPTIONAL,
        member-of                   [159]   SEQUENCE OF BACnetDeviceObjectReference OPTIONAL,
        profile-name                [168]   CharacterString OPTIONAL
        }
```

LIFE-SAFETY-ZONE ::= SEQUENCE {

object-identifier	[75]	BACnetObjectIdentifier,
object-name	[77]	CharacterString,
object-type	[79]	BACnetObjectType,
present-value	[85]	BACnetLifeSafetyState,
tracking-value	[164]	BACnetLifeSafetyState OPTIONAL,
description	[28]	CharacterString OPTIONAL,
device-type	[31]	CharacterString OPTIONAL,
status-flags	[111]	BACnetStatusFlags,
event-state	[36]	BACnetEventState,
reliability	[103]	BACnetReliability,
out-of-service	[81]	BOOLEAN,
mode	[160]	BACnetLifeSafetyMode,
accepted-modes	[175]	SEQUENCE OF BACnetLifeSafetyMode,
time-delay	[113]	Unsigned OPTIONAL,
notification-class	[17]	Unsigned OPTIONAL,
life-safety-alarm-values	[166]	SEQUENCE OF BACnetLifeSafetyState OPTIONAL,
alarm-values	[7]	SEQUENCE OF BACnetLifeSafetyState OPTIONAL,
fault-values	[39]	SEQUENCE OF BACnetLifeSafetyState OPTIONAL,
event-enable	[35]	BACnetEventTransitionBits OPTIONAL,
acked-transitions	[0]	BACnetEventTransitionBits OPTIONAL,
notify-type	[72]	BACnetNotifyType OPTIONAL,
event-time-stamps	[130]	SEQUENCE OF BACnetTimeStamp OPTIONAL,
		-- accessed as a BACnetARRAY
silenced	[163]	BACnetSilencedState,
operation-expected	[161]	BACnetLifeSafetyOperation,
maintenance-required	[158]	BOOLEAN OPTIONAL,
zone-members	[165]	SEQUENCE OF BACnetDeviceObjectReference,
member-of	[159]	SEQUENCE OF BACnetDeviceObjectReference OPTIONAL,
profile-name	[168]	CharacterString OPTIONAL

}

LOOP ::= SEQUENCE {

object-identifier	[75]	BACnetObjectIdentifier,
object-name	[77]	CharacterString,
object-type	[79]	BACnetObjectType,
present-value	[85]	REAL,
description	[28]	CharacterString OPTIONAL,
status-flags	[111]	BACnetStatusFlags,
event-state	[36]	BACnetEventState,
reliability	[103]	BACnetReliability OPTIONAL,
out-of-service	[81]	BOOLEAN,
update-interval	[118]	Unsigned OPTIONAL,
output-units	[82]	BACnetEngineeringUnits,
manipulated-variable-reference	[60]	BACnetObjectPropertyReference,
controlled-variable-reference	[19]	BACnetObjectPropertyReference,
controlled-variable-value	[21]	REAL,
controlled-variable-units	[20]	BACnetEngineeringUnits,
setpoint-reference	[109]	BACnetSetpointReference,
setpoint	[108]	REAL,
action	[2]	BACnetAction,
proportional-constant	[93]	REAL OPTIONAL,
proportional-constant-units	[94]	BACnetEngineeringUnits OPTIONAL,
integral-constant	[49]	REAL OPTIONAL,
integral-constant-units	[50]	BACnetEngineeringUnits OPTIONAL,
derivative-constant	[26]	REAL OPTIONAL,

```
    derivative-constant-units   [27]  BACnetEngineeringUnits OPTIONAL,
    bias                        [14]  REAL OPTIONAL,
    maximum-output              [61]  REAL OPTIONAL,
    minimum-output              [68]  REAL OPTIONAL,
    priority-for-writing        [88]  Unsigned (1..16),
    cov-increment               [22]  REAL OPTIONAL,
    time-delay                  [113] Unsigned OPTIONAL,
    notification-class          [17]  Unsigned OPTIONAL,
    error-limit                 [34]  REAL OPTIONAL,
    event-enable                [35]  BACnetEventTransitionBits OPTIONAL,
    acked-transitions           [0]   BACnetEventTransitionBits OPTIONAL,
    notify-type                 [72]  BACnetNotifyType OPTIONAL,
    event-time-stamps           [130] SEQUENCE OF BACnetTimeStamp OPTIONAL,
                                      -- accessed as a BACnetARRAY
    profile-name                [168] CharacterString OPTIONAL
    }

MULTI-STATE-INPUT ::= SEQUENCE {
    object-identifier           [75]  BACnetObjectIdentifier,
    object-name                 [77]  CharacterString,
    object-type                 [79]  BACnetObjectType,
    present-value               [85]  Unsigned, -- maximum value is restricted by the number-of-states
    description                 [28]  CharacterString OPTIONAL,
    device-type                 [31]  CharacterString OPTIONAL,
    status-flags                [111] BACnetStatusFlags,
    event-state                 [36]  BACnetEventState,
    reliability                 [103] BACnetReliability OPTIONAL,
    out-of-service              [81]  BOOLEAN,
    number-of-states            [74]  Unsigned,
    state-text                  [110] SEQUENCE OF CharacterString OPTIONAL, -- accessed as a BACnetARRAY
    time-delay                  [113] Unsigned OPTIONAL,
    notification-class          [17]  Unsigned OPTIONAL,
    alarm-values                [7]   SEQUENCE OF Unsigned OPTIONAL,
    fault-values                [39]  SEQUENCE OF Unsigned OPTIONAL,
    event-enable                [35]  BACnetEventTransitionBits OPTIONAL,
    acked-transitions           [0]   BACnetEventTransitionBits OPTIONAL,
    notify-type                 [72]  BACnetNotifyType OPTIONAL,
    event-time-stamps           [130] SEQUENCE OF BACnetTimeStamp OPTIONAL,
                                      -- accessed as a BACnetARRAY
    profile-name                [168] CharacterString OPTIONAL
    }

MULTI-STATE-OUTPUT ::= SEQUENCE {
    object-identifier           [75]  BACnetObjectIdentifier,
    object-name                 [77]  CharacterString,
    object-type                 [79]  BACnetObjectType,
    present-value               [85]  Unsigned, -- maximum value is restricted by the number-of-states
    description                 [28]  CharacterString OPTIONAL,
    device-type                 [31]  CharacterString OPTIONAL,
    status-flags                [111] BACnetStatusFlags,
    event-state                 [36]  BACnetEventState,
    reliability                 [103] BACnetReliability OPTIONAL,
    out-of-service              [81]  BOOLEAN,
    number-of-states            [74]  Unsigned,
    state-text                  [110] SEQUENCE OF CharacterString OPTIONAL, -- accessed as a BACnetARRAY
    priority-array              [87]  BACnetPriorityArray,
```

```
        relinquish-default          [104]  Unsigned,
        time-delay                  [113]  Unsigned OPTIONAL,
        notification-class          [17]   Unsigned OPTIONAL,
        feedback-value              [40]   Unsigned OPTIONAL,
        event-enable                [35]   BACnetEventTransitionBits OPTIONAL,
        acked-transitions           [0]    BACnetEventTransitionBits OPTIONAL,
        notify-type                 [72]   BACnetNotifyType OPTIONAL,
        event-time-stamps           [130]  SEQUENCE OF BACnetTimeStamp OPTIONAL,
                                           -- accessed as a BACnetARRAY
        profile-name                [168]  CharacterString OPTIONAL
        }

MULTI-STATE-VALUE ::= SEQUENCE {
        object-identifier           [75]   BACnetObjectIdentifier,
        object-name                 [77]   CharacterString,
        object-type                 [79]   BACnetObjectType,
        present-value               [85]   Unsigned,
                                           -- maximum value is restricted by the number-of-states
        description                 [28]   CharacterString OPTIONAL,
        status-flags                [111]  BACnetStatusFlags,
        event-state                 [36]   BACnetEventState,
        reliability                 [103]  BACnetReliability OPTIONAL,
        out-of-service              [81]   BOOLEAN,
        number-of-states            [74]   Unsigned,
        state-text                  [110]  SEQUENCE OF CharacterString OPTIONAL,
                                           -- accessed as a BACnetARRAY
        priority-array              [87]   BACnetPriorityArray OPTIONAL,
        relinquish-default          [104]  Unsigned OPTIONAL,
        time-delay                  [113]  Unsigned OPTIONAL,
        notification-class          [17]   Unsigned OPTIONAL,
        alarm-values                [7]    SEQUENCE OF Unsigned OPTIONAL,
        fault-values                [39]   SEQUENCE OF Unsigned OPTIONAL,
        event-enable                [35]   BACnetEventTransitionBits OPTIONAL,
        acked-transitions           [0]    BACnetEventTransitionBits OPTIONAL,
        notify-type                 [72]   BACnetNotifyType OPTIONAL,
        event-time-stamps           [130]  SEQUENCE OF BACnetTimeStamp OPTIONAL,
                                           -- accessed as a BACnetARRAY
        profile-name                [168]  CharacterString OPTIONAL
        }

NOTIFICATION-CLASS ::= SEQUENCE {
        object-identifier           [75]   BACnetObjectIdentifier,
        object-name                 [77]   CharacterString,
        object-type                 [79]   BACnetObjectType,
        description                 [28]   CharacterString OPTIONAL,
        notification-class          [17]   Unsigned,
        priority                    [86]   SEQUENCE SIZE(3) OF Unsigned, -- accessed as a BACnetARRAY
        ack-required                [1]    BACnetEventTransitionBits,
        recipient-list              [102]  SEQUENCE OF BACnetDestination,
        profile-name                [168]  CharacterString OPTIONAL
        }

PROGRAM ::= SEQUENCE {
        object-identifier           [75]   BACnetObjectIdentifier,
        object-name                 [77]   CharacterString,
        object-type                 [79]   BACnetObjectType,
```

```
        program-state               [92]  BACnetProgramState,
        program-change              [90]  BACnetProgramRequest,
        reason-for-halt             [100] BACnetProgramError OPTIONAL,
        description-of-halt         [29]  CharacterString OPTIONAL,
        program-location            [91]  CharacterString OPTIONAL,
        description                 [28]  CharacterString OPTIONAL,
        instance-of                 [48]  CharacterString OPTIONAL,
        status-flags                [111] BACnetStatusFlags,
        reliability                 [103] BACnetReliability OPTIONAL,
        out-of-service              [81]  BOOLEAN,
        profile-name                [168] CharacterString OPTIONAL
        }

PULSE-CONVERTER ::= SEQUENCE {
        object-identifier           [75]  BACnetObjectIdentifier,
        object-name                 [77]  CharacterString,
        object-type                 [79]  BACnetObjectType,
        description                 [28]  CharacterString OPTIONAL,
        present-value               [85]  REAL,
        input-reference             [181] BACnetObjectPropertyReference OPTIONAL,
        status-flags                [111] BACnetStatusFlags,
        event-state                 [36]  BACnetEventState,
        reliability                 [103] BACnetReliability OPTIONAL,
        out-of-service              [81]  BOOLEAN,
        units                       [117] BACnetEngineeringUnits,
        scale-factor                [188] REAL,
        adjust-value                [176] REAL,
        count                       [177] Unsigned,
        update-time                 [189] BACnetDateTime,
        count-change-time           [179] BACnetDateTime,
        count-before-change         [178] Unsigned,
        cov-increment               [22]  REAL OPTIONAL,
        cov-period                  [180] Unsigned OPTIONAL,
        notification-class          [17]  Unsigned OPTIONAL,
        time-delay                  [113] Unsigned OPTIONAL,
        high-limit                  [45]  REAL OPTIONAL,
        low-limit                   [59]  REAL OPTIONAL,
        deadband                    [25]  REAL OPTIONAL,
        limit-enable                [52]  BACnetLimitEnable OPTIONAL,
        event-enable                [35]  BACnetEventTransitionBits OPTIONAL,
        acked-transitions           [0]   BACnetEventTransitionBits OPTIONAL,
        notify-type                 [72]  BACnetNotifyType OPTIONAL,
        event-time-stamps           [130] SEQUENCE OF BACnetTimeStamp OPTIONAL,
        profile-name                [167] CharacterString OPTIONAL
        }

SCHEDULE ::= SEQUENCE {
        object-identifier           [75]  BACnetObjectIdentifier,
        object-name                 [77]  CharacterString,
        object-type                 [79]  BACnetObjectType,
        present-value               [85]  ABSTRACT-SYNTAX.&Type,  -- Any datatype
        description                 [28]  CharacterString OPTIONAL,
        effective-period            [32]  BACnetDateRange,
        weekly-schedule             [123] SEQUENCE SIZE(7) OF BACnetDailySchedule OPTIONAL,
                                          -- accessed as a BACnetARRAY
        schedule-default            [174] ABSTRACT-SYNTAX.&Type, -- Any primitive datatype,
```

```
        exception-schedule                [38]  SEQUENCE OF BACnetSpecialEvent OPTIONAL,
                                                -- accessed as a BACnetARRAY
        list-of-object-property-references [54]  SEQUENCE OF BACnetDeviceObjectPropertyReference,
        priority-for-writing              [88]  Unsigned (1..16),
        status-flags                      [111] BACnetStatusFlags,
        reliability                       [103] BACnetReliability,
        out-of-service                    [81]  BOOLEAN,
        profile-name                      [168] CharacterString OPTIONAL
        }

TREND-LOG :: = SEQUENCE {
        object-identifier                 [75]  BACnetObjectIdentifier,
        object-name                       [77]  CharacterString,
        object-type                       [79]  BACnetObjectType,
        description                       [28]  CharacterString OPTIONAL,
        log-enable                        [133] BOOLEAN,
        start-time                        [142] BACnetDateTime OPTIONAL,
        stop-time                         [143] BACnetDateTime OPTIONAL,
        log-device-object-property        [132] BACnetDeviceObjectPropertyReference OPTIONAL,
        log-interval                      [134] Unsigned OPTIONAL,
        cov-resubscription-interval       [128] Unsigned OPTIONAL,
        client-cov-increment              [127] BACnetClientCOV OPTIONAL,
        stop-when-full                    [144] BOOLEAN,
        buffer-size                       [126] Unsigned32,
        log-buffer                        [131] SEQUENCE OF BACnetLogRecord,
        record-count                      [141] Unsigned32,
        total-record-count                [145] Unsigned32,
        notification-threshold            [137] Unsigned32 OPTIONAL,
        records-since-notification        [140] Unsigned32 OPTIONAL,
        last-notify-record                [173] Unsigned32 OPTIONAL,
        event-state                       [36]  BACnetEventState,
        notification-class                [17]  Unsigned OPTIONAL,
        event-enable                      [35]  BACnetEventTransitionBits OPTIONAL,
        acked-transitions                 [0]   BACnetEventTransitionBits OPTIONAL,
        notify-type                       [72]  BACnetNotifyType OPTIONAL,
        event-time-stamps                 [130] SEQUENCE OF BACnetTimeStamp OPTIONAL,
                                                --accessed as a BACnetARRAY
        profile-name                      [168] CharacterString OPTIONAL
        }
```

ANNEX D - EXAMPLES OF STANDARD OBJECT TYPES (INFORMATIVE)

(This annex is not part of this standard but is included for informative purposes.)

This annex provides examples of the BACnet standard object types defined in Clause 12.

D.1 Example of an Accumulator object

Property:	Object_Identifier =	(Accumulator, Instance 1)
Property:	Object_Name =	"Tenant 1"
Property:	Object_Type =	ACCUMULATOR
Property	Present_Value =	323
Property:	Description =	""
Property:	Device_Type =	"Electric Pulse"
Property:	Status_Flags =	{FALSE, FALSE, FALSE, FALSE}
Property:	Event_State =	NORMAL
Property:	Out_Of_Service =	FALSE
Property:	Scale =	2
Property:	Units =	KILOWATT_HOURS
Property:	Prescale =	(1,10000)
Property:	Max_Pres_Value =	9999
Property:	Value_Change_Time =	(23-MAR-01,18:50:21.2)
Property:	Value_Before_Change =	0
Property:	Value_Set =	67
Property:	Logging_Record =	((13-JUL-01,13:00:00.2),120,1,NORMAL)
Property:	Logging_Object =	(Trend Log, Instance 100)
Property:	Pulse_Rate =	3
Property:	High_Limit =	15
Property:	Low_Limit =	0
Property:	Limit_Monitoring_Interval =	300
Property:	Notification_Class =	3
Property:	Time_Delay =	10
Property:	Limit_Enable =	{TRUE, FALSE}
Property:	Event_Enable =	{TRUE, FALSE, TRUE}
Property:	Acked_Transitions =	{TRUE, TRUE, TRUE}
Property:	Notify_Type =	ALARM
Property:	Event_Time_Stamps =	((12-JUL-01,18:50:21.2), (*-*-*,*:*:*.*), (12-JUL-01,19:01:34.0))

D.2 Example of an Analog Input Object

The following is an example of an Analog Input object that is used for mixed air temperature of an air handler. The object supports both COV and intrinsic reporting.

Property:	Object_Identifier =	(Analog Input, Instance 1)
Property:	Object_Name =	"1AH1MAT"
Property:	Object_Type =	ANALOG_INPUT
Property:	Present_Value =	58.1
Property:	Description =	"Mixed Air Temperature"
Property:	Device_Type =	"1000 OHM RTD"
Property:	Status_Flags =	{FALSE, FALSE, FALSE, FALSE}
Property:	Event_State =	NORMAL
Property:	Reliability =	NO_FAULT_DETECTED
Property:	Out_Of_Service =	FALSE
Property:	Update_Interval =	10

Property:	Units =	DEGREES_FAHRENHEIT
Property:	Min_Pres_Value =	-50.0
Property:	Max_Pres_Value =	250.0
Property:	Resolution =	0.1
Property:	COV_Increment =	0.2
Property:	Time_Delay =	10
Property:	Notification_Class =	3
Property:	High_Limit =	60.0
Property:	Low_Limit =	55.0
Property:	Deadband =	1.0
Property:	Limit_Enable =	{TRUE, TRUE}
Property:	Event_Enable =	{TRUE, FALSE, TRUE}
Property:	Acked_Transitions =	{TRUE, TRUE, TRUE}
Property:	Notify_Type =	EVENT
Property:	Event_Time_Stamps =	((23-MAR-95,18:50:21.2), (*-*-*,*:*:*.*), (23-MAR-95,19:01:34.0))

D.3 Example of an Analog Output Object

The following is an example of an Analog Output object that is used for a damper in an air handler. The object supports neither COV nor intrinsic reporting.

Property:	Object_Identifier =	(Analog Output, Instance 1)
Property:	Object_Name =	"1AH1DMPR"
Property:	Object_Type =	ANALOG_OUTPUT
Property:	Present_Value =	75.0
Property:	Description =	"Damper Actuator"
Property:	Device_Type =	"3-8 PSI Actuator"
Property:	Status_Flags =	{FALSE, FALSE, FALSE, FALSE}
Property:	Event_State =	NORMAL
Property:	Reliability =	NO_FAULT_DETECTED
Property:	Out_Of_Service =	FALSE
Property:	Units =	PERCENT
Property:	Min_Pres_Value =	0.0
Property:	Max_Pres_Value =	100.0
Property:	Resolution =	0.1
Property:	Priority_Array =	{NULL, NULL, NULL, NULL, 75.0... NULL}
Property:	Relinquish_Default =	50.0

D.4 Example of an Analog Value Object

The following is an example of an Analog Value object that is used for enthalpy calculation. The object supports neither COV nor intrinsic reporting.

Property:	Object_Identifier =	(Analog Value, Instance 1)
Property:	Object_Name =	"1AH1ENTH"
Property:	Object_Type =	ANALOG_VALUE
Property:	Present_Value =	38.1
Property:	Description =	"Enthalpy"
Property:	Status_Flags =	{FALSE, FALSE, FALSE, FALSE}
Property:	Event_State =	NORMAL
Property:	Reliability =	NO_FAULT_DETECTED
Property:	Out_Of_Service =	FALSE
Property:	Units =	BTUS-PER-POUND-DRY-AIR

D.5 Example of an Averaging Object

The following is an example of an Averaging object that is used for determining average and maximum electrical demand. The object property it refers to is a standard analog input measuring KW. In the current period, one sample was missed.

Property:	Object_Identifier =	(Averaging, Instance 1)
Property:	Object_Name =	"FLR 12 DEMAND"
Property:	Object_Type =	AVERAGING
Property:	Minimum_Value =	2.4
Property:	Minimum_Value_Timestamp =	(16-DEC-1999,13:15:07.32)
Property:	Average_Value =	12.7
Property:	Maximum_Value =	18.8
Property:	Maximum_Value_Timestamp =	(16-DEC-1999,13:06:12.19)
Property:	Description =	"Floor 12 Electrical Demand"
Property:	Attempted_Samples =	15
Property:	Valid_Samples =	14
Property:	Object_Property_Reference =	(Analog Input, Instance 12)
Property:	Window_Interval =	900
Property:	Window_Samples =	15

D.6 Examples of a Binary Input Object

Example 1: A typical Binary Input object.

In this example, the Binary Input is connected to a high static pressure cutoff switch with normally open contacts. Exceeding the static pressure limit of the switch causes the normally open contacts to close. The ACTIVE state of the physical input indicates that the contacts are closed. The object supports both COV and intrinsic reporting.

Property:	Object_Identifier =	(Binary Input, Instance 1)
Property:	Object_Name =	"HighPressSwitch"
Property:	Object_Type =	BINARY_INPUT
Property:	Present_Value =	ACTIVE
Property:	Description =	"Penthouse Supply High Static"
Property:	Device_Type =	"ABC Pressure Switch"
Property:	Status_Flags =	{TRUE, FALSE, FALSE, FALSE}
Property:	Event_State =	OFFNORMAL
Property:	Reliability =	NO_FAULT_DETECTED
Property:	Out_Of_Service =	FALSE
Property:	Polarity =	NORMAL
Property:	Inactive_Text =	"Static Pressure OK"
Property:	Active_Text =	"High Pressure Alarm"
Property:	Change_Of_State_Time =	(23-MAR-1995, 19:01:34.0)
Property:	Change_Of_State_Count =	134
Property:	Time_Of_State_Count_Reset =	(1-JAN-1995, 00:00:00.0)
Property:	Elapsed_Active_Time =	401
Property:	Time_Of_Active_Time_Reset =	(1-JAN-1995, 00:00:00.0)
Property:	Time_Delay =	10
Property:	Notification_Class =	3
Property:	Alarm_Value =	ACTIVE
Property:	Event_Enable =	{TRUE, FALSE, TRUE}
Property:	Acked_Transitions =	{FALSE, TRUE, TRUE}
Property:	Notify_Type =	ALARM
Property:	Event_Time_Stamps =	((23-MAR-95,18:50:21.2), (*-*-*,*:*:*.*), (21-MAR-95,01:02:03.0))

Example 2: A Binary Input object that is out-of-service.

In this second example, an open circuit has been found by the control system in the binary input described above. The input has been taken out of service.

Property:	Object_Identifier =	(Binary Input, Instance 1)
Property:	Object_Name =	"HighPressSwitch"
Property:	Object_Type =	BINARY_INPUT
Property:	Present_Value =	INACTIVE
Property:	Description =	"Penthouse Supply High Static"
Property:	Device_Type =	"ABC Pressure Switch"
Property:	Status_Flags =	{FALSE, TRUE, FALSE, TRUE}
Property:	Event_State =	NORMAL
Property:	Reliability =	OPEN_LOOP
Property:	Out_Of_Service =	TRUE
Property:	Polarity =	NORMAL
Property:	Inactive_Text =	"Static Pressure OK"
Property:	Active_Text =	"High Pressure Alarm"
Property:	Change_Of_State_Time =	(23-MAR-1995, 19:01:34.0)
Property:	Change_Of_State_Count =	135
Property:	Time_Of_State_Count_Reset =	(1-JAN-1995, 00:00:00.0)
Property:	Elapsed_Active_Time =	451
Property:	Time_Of_Active_Time_Reset =	(1-JAN-1995, 00:00:00.0)
Property:	Time_Delay =	10
Property:	Notification_Class =	3
Property:	Alarm_Value =	ACTIVE
Property:	Event_Enable =	{TRUE, FALSE, TRUE}
Property:	Acked_Transitions =	{TRUE, TRUE, TRUE}
Property:	Notify_Type =	ALARM
Property:	Event_Time_Stamps =	((21-MAR-95,01:09:34.0), (23-MAR-95,19:01:34.0), (21-MAR-95,01:11:21.2))

D.7 Examples of a Binary Output Object

Example 1: A typical Binary Output object.

In this example, a fan is controlled by a binary output connected to a relay with normally closed contacts. The fan operates unless the relay is energized. The INACTIVE state of the binary output object is reversed by the polarity property to energize the relay and open the contacts turning off the fan. Note that in this example the Priority_Array contains all NULLs and thus Relinquish_Default determines the Present_Value. The object does not support intrinsic reporting.

Property:	Object_Identifier =	(Binary Output, Instance 1)
Property:	Object_Name =	"Floor3ExhaustFan"
Property:	Object_Type =	BINARY_OUTPUT
Property:	Present_Value =	INACTIVE
Property:	Description =	"Third floor bathroom exhaust fan"
Property:	Device_Type =	"ABC 100 Relay"
Property:	Status_Flags =	{FALSE, FALSE, FALSE, FALSE}
Property:	Event_State =	NORMAL
Property:	Reliability =	NO_FAULT_DETECTED
Property:	Out_Of_Service =	FALSE
Property:	Polarity =	REVERSE

Property:	Inactive_Text =	"Fan is turned off"
Property:	Active_Text =	"Fan is running"
Property:	Change_Of_State_Time =	(23-MAR-1995, 19:01:34.0)
Property:	Change_Of_State_Count =	47
Property:	Time_Of_State_Count_Reset =	(1-JAN-1995, 00:00:00.0)
Property:	Elapsed_Active_Time =	650
Property:	Time_Of_Active_Time_Reset =	(1-JAN-1995, 00:00:00.0)
Property:	Minimum_Off_Time =	100
Property:	Minimum_On_Time =	10
Property:	Priority_Array =	{NULL, NULL, NULL, NULL, NULL, NULL, NULL, NULL, NULL, NULL, NULL, NULL, NULL, NULL, NULL, NULL}
Property:	Relinquish_Default =	INACTIVE

Example 2: A Binary Output object that has been overridden.

In this second example, the fan has been found to be inoperable. Since the fan cannot operate until repaired, an operator has overridden the control system to maintain the fan in an inactive state.

Property:	Object_Identifier =	(Binary Output, Instance 1)
Property:	Object_Name =	"Floor3ExhaustFan"
Property:	Object_Type =	BINARY_OUTPUT
Property:	Present_Value =	INACTIVE
Property:	Description =	"Third floor bathroom exhaust fan"
Property:	Device_Type =	"ABC 100 Relay"
Property:	Status_Flags =	{FALSE, TRUE, TRUE, FALSE)
Property:	Event_State =	NORMAL
Property:	Reliability =	OPEN_LOOP
Property:	Out_Of_Service =	FALSE
Property:	Polarity =	REVERSE
Property:	Inactive_Text =	"Fan is turned off"
Property:	Active_Text =	"Fan is running"
Property:	Change_Of_State_Time =	(23-MAR-1995, 19:01:34.0)
Property:	Change_Of_State_Count =	134
Property:	Time_Of_State_Count_Reset =	(1-JAN-1995, 00:00:00.0)
Property:	Elapsed_Active_Time =	401
Property:	Time_Of_Active_Time_Reset =	(1-JAN-1995, 00:00:00.0)
Property:	Minimum_Off_Time =	100
Property:	Minimum_On_Time =	10
Property:	Priority_Array =	{NULL, NULL, NULL, NULL, NULL, NULL, NULL, NULL, NULL, NULL, NULL, NULL, NULL, NULL, NULL, NULL}
Property:	Relinquish_Default =	INACTIVE

D.8 Example of a Binary Value Object

In this example, the Binary Value is a mechanism that allows an exhaust fan to be enabled by an operator. The fan control logic uses this value to determine if the fan should operate but only if other conditions are met. For example, the fan may not be turned off if a fire alarm has occurred or if the system is shut down. The object does not support intrinsic reporting.

Property:	Object_Identifier =	(Binary Value, Instance 1)
Property:	Object_Name =	"ExhaustFanEnable"
Property:	Object_Type =	BINARY_VALUE
Property:	Present_Value =	ACTIVE
Property:	Description =	"Exhaust Fan Operator Enable"
Property:	Status_Flags =	{FALSE, FALSE, FALSE, FALSE}

Property:	Event_State =	NORMAL
Property:	Reliability =	NO_FAULT_DETECTED
Property:	Out_Of_Service =	FALSE
Property:	Inactive_Text =	"Enabled by Operator"
Property:	Active_Text =	"Fan Not Enabled by Operator"
Property:	Change_Of_State_Time =	(23-MAR-1995, 19:01:34.0)
Property:	Change_Of_State_Count =	134
Property:	Time_Of_State_Count_Reset =	(1-JAN-1995, 00:00:00.0)
Property:	Elapsed_Active_Time =	401
Property:	Time_Of_Active_Time_Reset =	(1-JAN-1995, 00:00:00.0)
Property:	Minimum_Off_Time =	0
Property:	Minimum_On_Time =	0
Property:	Priority_Array =	{NULL...NULL, ACTIVE}
Property:	Relinquish_Default =	INACTIVE

D.9 Example of a Calendar Object

The following is an example of a CALENDAR object that specifies the holidays for a school district.

Property:	Object_Identifier =	(Calendar, Instance 1)
Property:	Object_Name =	"HOLIDAYS"
Property:	Object_Type =	CALENDAR
Property:	Description =	"1995-1996 School District Holidays"
Property:	Present_Value =	TRUE
Property:	Date_List =	(((23-DEC-1995)-(3-JAN-1996)), (19-FEB-1996), (27-MAY-1996))

This is a calendar called "HOLIDAYS". Holidays are defined for Christmas Vacation between December 23 and January 3, for Presidents' Day on February 19, and for Memorial Day on May 27. On these dates the Present_Value of the calendar will be TRUE. On all other dates, the Present_Value will be FALSE. A real school calendar would likely have more members in the Date_List; only three are shown here for simplicity.

D.10 Examples of a Command Object

Example 1: Occupied and unoccupied zone.

In this example, a particular zone of an office building has two temperature setpoints: one for unoccupied and one for occupied zone operation. The zone also has a lighting override to force lighting off during unoccupied periods. Setting the Present_Value of the COMMAND object to 1 will select unoccupied mode, while 2 will select occupied mode.

Assumed objects:

Object_Identifier	Object_Name	Object_Type
X'00800005'	ZONE43	ANALOG_VALUE
X'01000003'	LIGHTING43	BINARY_OUTPUT

Property:	Object_Identifier =	(Command, Instance 1)
Property:	Object_Name =	"ZONE43CONTROL"
Property:	Object_Type =	COMMAND
Property:	Description =	"Fourth Floor, West Wing Office Suite"
Property:	Present_Value =	1
Property:	In_Process =	FALSE
Property:	All_Writes_Successful =	TRUE

```
Property:    Action =        {((,(Analog Value, Instance 5),Present_Value,,65.0,,TRUE,TRUE),
                                 (,(Binary Output, Instance 3),Present_Value,
                                  ,INACTIVE,8,1,TRUE,TRUE)),
                                 ((,(Analog Value, Instance 5),
                                 Present_Value,,72.0,,TRUE,TRUE),
                                 (,(Binary Output, Instance 3), Present_Value,
                                  ,ACTIVE,8,2,TRUE,TRUE))}
Property:    Action_Text =   {"Unoccupied", "Occupied"}
```

Example 2: Occupied and unoccupied zone.

This example builds on the previous example by adding communication between the device containing the COMMAND object and another device that controls an elevator subsystem. When the zone is placed in an unoccupied mode, that floor of the building is locked out for elevator service. In this example, the elevator subsystem is located in the BACnet device "DDC4".

Assumed objects:

Object_Identifier	Object_Name	Object_Type
X'00800005'	ZONE43	ANALOG_VALUE
X'01000003'	LIGHTING43	BINARY_OUTPUT
X'02000001'	DDC4	DEVICE
X'01400001'	FL4	BINARY_VALUE

```
Property:    Object_Identifier =      (Command, Instance 1)
Property:    Object_Name =            "ZONE43CONTROL"
Property:    Object_Type =            COMMAND
Property:    Description =            "Fourth Floor, West Wing Office Suite"
Property:    Present_Value =          2
Property:    In_Process =             FALSE
Property:    All_Writes_Successful =  TRUE
Property:    Action =                 {((,(Analog Value, Instance 5), Present_Value,,65.0, 8,,TRUE,TRUE),
                                          (,(Binary Output, Instance 3), Present_Value,
                                           ,INACTIVE,8,1,TRUE,TRUE),
                                           ((Device, Instance 1),
                                              (Binary Value, Instance 1),Present_Value,
                                                ,INACTIVE, 8,1,TRUE,TRUE)),

                                          ((,(Analog Value, Instance 5), Present_Value,,72.0,
                                             8,2,TRUE,TRUE),
                                           (,(Binary Output, Instance 3), Present_Value,
                                             ,ACTIVE,8,,TRUE,TRUE),
                                          ((Device, Instance 1),
                                             (Binary Value, Instance 1),Present_Value,,ACTIVE,
                                             8,,TRUE,TRUE))}
Property:    Action_Text =            {"Unoccupied", "Occupied"}
```

D.11 Examples of a Device Object

Example 1: A "sophisticated" BACnet device.

```
Property:    Object_Identifier =      (Device, Instance 1)
Property:    Object_Name =            "AC1 System Controller"
Property:    Object_Type =            DEVICE
Property:    System_Status =          OPERATIONAL
Property:    Vendor_Name =            "ABC Controls"
Property:    Vendor_Identifier =      1001
Property:    Model_Name =             "1000 Plus"
```

Property:	Firmware_Revision =	"1.2"
Property:	Application_Software_Version =	"V4.0 - April 12, 1989"
Property:	Location =	"Basement Mechanical Room"
Property:	Description =	"AC1 Controller"
Property:	Protocol_Version =	1
Property:	Protocol_Revision =	4
Property:	Protocol_Services_Supported =	B'11111111111111111111111111111111111111'
Property:	Protocol_Object_Types_Supported =	B'11111111111111111'
Property:	Object_List =	((Analog Input, Instance 1), (Analog Input, Instance 2), ...)
Property:	Max_APDU_Length_Accepted =	480
Property:	Segmentation_Supported =	SEGMENTED_BOTH
Property:	Max_Segments_Accepted =	16
Property:	VT_Classes_Supported =	(DEFAULT-TERMINAL, DEC-VT100)
Property:	Active_VT_Sessions	((29,5,1, X'0000C0B3EF13'),(23,8,2, X'05'))
Property:	Local_Time	12:34:56.78
Property:	Local_Date	29-SEP-1989, FRIDAY
Property:	UTC_Offset =	6.0
Property:	Daylight_Savings_Status =	FALSE
Property:	APDU_Segment_Timeout =	2000
Property:	APDU_Timeout =	3000
Property:	Number_Of_APDU_Retries =	3
Property:	List_Of_Session_Keys =	((X'3799246237984589', 1, X'03'), (X'4446214686489744', 1, X'05'))
Property:	Time_Synchronization_Recipients =	(Device, Instance 18)
Property:	Device_Address_Binding =	(((Device, Instance 1), 1, X'01'), ((Device, Instance 12), 1, X'17'), ((Device, Instance 40), 2, X'02608C41A606'), ...)
Property:	Database_Revision =	123
Property:	Configuration_Files =	((File, Instance 1), (File, Instance 2))
Property:	Last_Restore_Time =	(2, (29-SEP-1989, 01:00:00.0))
Property:	Backup_Failure_Timeout =	300
Property:	Active_COV_Subscriptions =	((((0, (Device, Instance 12)), 300), ((Analog Input, Instance 1),Present_Value),TRUE,100,1.0), (((0, (Device, Instance 40)), 600), ((Analog Input, Instance 1),Present_Value),TRUE,3,1.5))

Example 2: A "simple" BACnet device.

Property:	Object_Identifier =	(Device, Instance 2)
Property:	Object_Name =	"Room 101 VAV Controller"
Property:	Object_Type =	DEVICE
Property:	System_Status =	DOWNLOAD_REQUIRED
Property:	Vendor_Name =	"XYZ Controls"
Property:	Vendor_Identifier =	1001
Property:	Model_Name =	"VAV 100"
Property:	Firmware_Revision =	"1.0"
Property:	Application_Software_Version =	"2-1-88"
Property:	Protocol_Version =	1
Property:	Protocol_Revision =	1
Property:	Protocol_Services_Supported =	B'11110100000010010000100000111100011'
Property:	Protocol_Object_Types_Supported =	B'110110110100010001'

Property:	Object_List =	((Analog Input, Instance 1), (Analog Input, Instance 2), (Analog Output, Instance 1), (Binary Input, Instance 1), (Binary Output, Instance 1), (Device, Instance 9))
Property:	Max_APDU_Length_Supported =	50
Property:	Segmentation_Supported =	NO_SEGMENTATION
Property:	APDU_Segment_Timeout =	2000
Property:	APDU_Timeout =	60,000
Property:	Number_Of_APDU_Retries =	3
Property:	Max_Master	85
Property:	Max_Info_Frames	3
Property:	Device_Address_Binding =	(((Device, Instance 1), 1, X'01'), ((Device, Instance 12), 1, X'17'))
Property:	Database_Revision =	69

D.12 Examples of an Event Enrollment Object

The following Analog Input object is assumed for the examples below. All objects are assumed to be located in device 12.

ANALOG_INPUT

Property:	Object_Identifier =	(Analog Input, Instance 2)
Property:	Object_Name =	"Zone1_Temp"
Property:	Object_Type =	ANALOG_INPUT
Property:	Present_Value =	86.0
Property:	Description =	"Receptionist Lobby Temp"
Property:	Device_Type =	"PT 3K RTD"
Property:	Status_Flags =	{FALSE, FALSE, FALSE, FALSE}
Property:	Event_State =	NORMAL
Property:	Reliability =	NO_FAULT_DETECTED
Property:	Out_Of_Service =	FALSE
Property:	Update_Interval =	5
Property:	Units =	DEGREES_FAHRENHEIT
Property:	Min_Pres_Value =	55.0
Property:	Max_Pres_Value =	95.0
Property:	Resolution =	0.1

Example 1: OUT_OF_RANGE event type.

This is an example of an OUT_OF_RANGE Event_Type. The State transition NORMAL to HIGH_LIMIT has not been acknowledged. The management of recipients is done through the use of a Notification Class object.

EVENT_ENROLLMENT

Property:	Object_Identifier =	(Event Enrollment, Instance 1)
Property:	Object_Name =	"Zone1_Alarm"
Property:	Object_Type =	EVENT_ENROLLMENT
Property:	Description =	"Zone 1 Alarms"
Property:	Event_Type =	OUT_OF_RANGE
Property:	Notify_Type =	ALARM
Property:	Event_Parameters =	(30, 65.0, 85.0, 0.25)
Property:	Object_Property_Reference =	((Device, Instance 12),(Analog Input, Instance 2), Present_Value)
Property:	Event_State =	HIGH_LIMIT

Property:	Event_Enable =	(TRUE, TRUE, TRUE)
Property:	Acked_Transitions =	(FALSE, TRUE, TRUE)
Property:	Notification_Class =	1
Property:	Event_Time_Stamps =	((23-MAR-95,18:50:21.2), (*-*-*,*:*:*.*), (21-MAR-95,01:02:03.0))

Example 2: CHANGE_OF_VALUE_EVENT event type.

This is an example of a CHANGE_OF_VALUE Event_Type. The management of recipients is done through the use of optional properties of the Event Enrollment object instead of a Notification Class object.

EVENT_ENROLLMENT

Property:	Object_Identifier =	(Event Enrollment, Instance 2)
Property:	Object_Name =	"Zone1TempCOV"
Property:	Object_Type =	EVENT_ENROLLMENT
Property:	Description =	"Zone 1 Temperature COV"
Property:	Event_Type =	CHANGE_OF_VALUE
Property:	Notify_Type =	EVENT
Property:	Event_Parameters =	(5, 0.25)
Property:	Object_Property_Reference =	((Device, Instance 12),(Analog Input, Instance 2), Present_Value)
Property:	Event_State =	NORMAL
Property:	Event_Enable =	(TRUE, FALSE, FALSE)
Property:	Acked_Transitions =	(TRUE, TRUE, TRUE)
Property:	Event_Time_Stamps =	((23-MAR-95,18:50:21.2), (*-*-*,*:*:*.*), (23-MAR-95,19:01:34.0))

Example 3: CHANGE_OF_BITSTRING event type.

This example illustrates the use of a CHANGE_OF_BITSTRING event.

EVENT_ENROLLMENT

Property:	Object_Identifier =	(Event Enrollment, Instance 3)
Property:	Object_Name =	"Zone1 Rel"
Property:	Object_Type =	EVENT_ENROLLMENT
Property:	Description =	"Reliability alarm for zone 1 temperature"
Property:	Event_Type =	CHANGE_OF_BITSTRING
Property:	Notify_Type =	ALARM
Property:	Event_Parameters =	(30, B'0111', (B'0100', B'0010', B'0001', B'0110', B'0101', B'0011'))
Property:	Object_Property_Reference =	((Device, Instance 12),(Analog Input, Instance 2), Status_Flags)
Property:	Event_State =	NORMAL
Property:	Event_Enable =	(TRUE, TRUE, FALSE)
Property:	Acked_Transitions =	(TRUE, TRUE, TRUE)
Property:	Notification_Class	3
Property:	Event_Time_Stamps =	((23-MAR-95,18:50:21.2), (*-*-*,*:*:*.*), (23-MAR-95,19:01:34.0))

D.13 Example of a File Object

A File Object holding trend information:

Property:	Object_Identifier =	(File, Instance 7)
Property:	Object_Name =	"TREND_AI1"
Property:	Object_Type =	FILE
Property:	Description =	"Trend of AI1"
Property:	File_Type =	"TREND"
Property:	File_Size =	750
Property:	Modification_Date =	(1-NOV-1995, 08:30:49.0)
Property:	Archive =	FALSE
Property:	Read_Only =	FALSE
Property:	File_Access_Method=	RECORD_ACCESS
Property:	Record_Count =	150

D.14 Example of a Group Object

The following is an example of a group object that is used to reference temperatures in a particular zone of a building.

Property:	Object_Identifier =	(Group, Instance 1)
Property:	Object_Name =	"ZONE1_TEMPS"
Property:	Object_Type =	GROUP
Property:	Description =	"Zone 1 Temperature Group"
Property:	List_Of_Group_Members =	(((Analog Input, Instance 8),(Present_Value, Reliability, Description)), ((Analog Input, Instance 9),(Present_Value, Reliability, Description)), ((Analog Input, Instance 10),(Present_Value, Reliability, Description)), ((Analog Input, Instance 11),(Present_Value, Reliability, Description)), ((Analog Input, Instance 12),(Present_Value, Reliability, Description)))
Property:	Present_Value =	(((Analog Input, Instance 8), Present_Value, 69.7, Reliability, NO_FAULT_DETECTED, Description, "Room 1"), ((Analog Input, Instance 9), Present_Value, 71.2, Reliability, NO_FAULT_DETECTED, Description, "Room 2"), ((Analog Input, Instance 10), Present_Value, -50.0, Reliability, UNRELIABLE_OTHER, Description, "Room 3"), ((Analog Input, Instance 11), Present_Value, 69.7, Reliability, NO_FAULT_DETECTED, Description, "Room 4"), ((Analog Input, Instance 12), Present_Value, 73.3, Reliability, NO_FAULT_DETECTED, Description, "Room 5"))

D.15 Example of a Life Safety Point Object

In this example, a smoke detector is represented as a Life Safety Point object.

Property:	Object_Identifier =	(Life Safety Point, Instance 2)
Property:	Object_Name =	"SMK3W"
Property:	Object_Type =	LIFE_SAFETY_POINT
Property:	Present_Value =	PREALARM
Property:	Tracking_Value =	PREALARM
Property:	Description =	"Floor 3, West Zone Smoke Detector"
Property:	Device_Type =	"Old Smokey model 123"
Property:	Status_Flags =	{TRUE, FALSE, FALSE, FALSE}
Property:	Event_State =	LIFE_SAFETY_ALARM
Property:	Reliability =	NO_FAULT_DETECTED
Property:	Out_Of_Service =	FALSE
Property:	Mode =	ON

Property:	Accepted_Modes =	{ENABLED, DISABLED, TEST}
Property:	Time_Delay =	10
Property:	Notification_Class =	39
Property:	Life_Safety_Alarm_Values =	(ALARM)
Property:	Alarm_Values =	(PREALARM)
Property:	Fault_Values =	(FAULT)
Property:	Event_Enable =	{TRUE, TRUE, TRUE}
Property:	Acked_Transitions =	{TRUE, TRUE, TRUE}
Property:	Notify_Type =	ALARM
Property:	Event_Time_Stamps =	((23-MAR-95, 18:50:21.2), (*-*-*, *:*:*.*), (23-MAR-95, 19:01:34.0))
Property:	Silenced =	SILENCE_AUDIBLE
Property:	Operation_Expected =	RESET_ALARM
Property:	Maintenance_Required =	NONE
Property:	Setting =	50
Property:	Direct_Reading =	84.3
Property:	Units =	PERCENT-OBSCURATION-PER-METER
Property:	Member_Of =	((Life Safety Zone, Instance 5))

D.16 Example of a Life Safety Zone Object

In this example, a fire zone is represented as a Life Safety Zone object.

Property:	Object_Identifier =	(Life Safety Zone, Instance 2)
Property:	Object_Name =	"SMK3"
Property:	Object_Type =	LIFE_SAFETY_ZONE
Property:	Present_Value =	PREALARM
Property:	Tracking_Value =	PREALARM
Property:	Description =	"Floor 3 Smoke"
Property:	Status_Flags =	{TRUE, FALSE, FALSE, FALSE}
Property:	Event_State =	LIFESAFETY_ALARM
Property:	Reliability =	NO_FAULT_DETECTED
Property:	Out_Of_Service =	FALSE
Property:	Mode =	ON
Property:	Accepted_Modes =	{ENABLED, DISABLED, TEST}
Property:	Time_Delay =	10
Property:	Notification_Class =	39
Property:	Life_Safety_Alarm_Values =	(ALARM)
Property:	Alarm_Values =	(PREALARM)
Property:	Fault_Values =	(FAULT)
Property:	Event_Enable =	{TRUE, TRUE, TRUE}
Property:	Acked_Transitions =	{TRUE, TRUE, TRUE}
Property:	Notify_Type =	ALARM
Property:	Event_Time_Stamps =	((23-MAR-95, 18:50:21.2), (*-*-*, *:*:*.*), (23-MAR-95,19:01:34.0))
Property:	Silenced =	UNSILENCED
Property:	Operation_Expected =	SILENCE_AUDIBLE
Property:	Maintenance_Required =	NONE
Property:	Zone_Members =	((Life Safety Point, Instance 22), (Life Safety Point, Instance 23))
Property:	Member_Of =	((Life Safety Zone, Instance 5))

D.17 Example of a Loop Object

The following is an example of a LOOP object that is used for supply air temperature control of an air handler. The algorithm represented is a positioning PI algorithm of the following form:

Output = {Proportional_Constant * [Error + (Integral_Constant * "Integral of the Error")]} + Bias

Where: Error = (Process_Variable_Value - Setpoint) and "Integral of the Error" has units of °F-min.

Assumed objects:	Object_Identifier	Object_Name	Object_Type
	X'00400005'	AHU_VALVE	ANALOG_OUTPUT
	X'00000003'	AHU_SAT	ANALOG_INPUT
	X'00800007'	RESET_OAT	ANALOG_VALUE

The object supports COV reporting.

Property:	Object_Identifier =	(Loop, Instance 1)
Property:	Object_Name =	"AHU_SAT_LOOP"
Property:	Object_Type =	LOOP
Property:	Present_Value =	8.3
Property:	Description =	"Supply air temp. PI control"
Property:	Status_Flags =	{FALSE, FALSE, FALSE, FALSE}
Property:	Event_State =	NORMAL
Property:	Reliability =	NO_FAULT_DETECTED
Property:	Out_Of_Service =	FALSE
Property:	Update_Interval =	1
Property:	Output_Units =	POUNDS_FORCE_PER_SQUARE_INCH
Property:	Manipulated_Variable_Reference =	((Analog Output, Instance 5),Present_Value)
Property:	Controlled_Variable_Reference =	((Analog Input, Instance 3), Present_Value)
Property:	Controlled_Variable_Value =	56.1
Property:	Controlled_Variable_Units =	DEGREES_FAHRENHEIT
Property:	Setpoint_Reference =	((Analog Value, Instance 7), Present_Value)
Property:	Setpoint =	57.0
Property:	Action =	DIRECT
Property:	Proportional_Constant =	0.5
Property:	Proportional_Constant_Units =	PSI_PER_DEGREE_FAHRENHEIT
Property:	Integral_Constant =	0.1
Property:	Integral_Constant_Units =	PER-MINUTE
Property:	Derivative_Constant =	0.0
Property:	Derivative_Constant_Units =	NO-UNITS
Property:	Bias =	9.0
Property:	Maximum_Output =	15.0
Property:	Minimum_Output =	3.0
Property:	Priority_For_Writing =	10
Property:	COV_Increment =	0.2
Property:	Time_Delay =	3
Property:	Notification_Class =	1
Property:	Error_Limit =	5.0
Property:	Event_Enable =	{TRUE, TRUE, TRUE}
Property:	Acked_Transitions =	{TRUE, TRUE, TRUE}
Property:	Notify_Type =	ALARM
Property:	Event_Time_Stamps =	((23-MAR-95,18:50:21.2), (*-*-*,*:*:*.*), (23-MAR-95,19:01:34.0))

D.18 Examples of a Multi-state Input Object

Example 1 - Two-speed fan.

In this example Input #1 is connected to the "aux" contact on the low-speed starter and Input #2 is connected to the "aux" contact of the high-speed starter. If the low-speed starter is "off," then the high-speed starter is disabled. The table below shows the relationship between the two inputs and the present value. The actual logic used to determine and establish the present value is a local matter. The object supports intrinsic reporting.

Present Value	Low-Speed Starter	High-Speed Starter
1	OFF (inactive)	OFF (inactive)
2	ON (active)	OFF (inactive)
3	ON (active)	ON (active)

Property:	Object_Identifier =	(Multi-state Input, Instance 1)
Property:	Object_Name =	"Fan1_Input"
Property:	Object_Type =	MULTI_STATE_INPUT
Property:	Present_Value =	2
Property:	Description =	"2-speed Fan#1"
Property:	Device_Type =	"ZZZ Fan Motor"
Property:	Status_Flags =	{FALSE, FALSE, FALSE, FALSE}
Property:	Event_State =	NORMAL
Property:	Reliability =	NO_FAULT_DETECTED
Property:	Out_Of_Service =	FALSE
Property:	Number_Of_States =	3
Property:	State_Text =	("Off", "On_Low", "On_High")
Property:	Time_Delay =	3
Property:	Notification_Class =	4
Property:	Alarm_Values =	(3)
Property:	Fault_Values =	(2)
Property:	Event_Enable =	{TRUE, TRUE, TRUE}
Property:	Acked_Transitions =	{TRUE, TRUE, TRUE}
Property:	Notify_Type =	EVENT
Property:	Event_Time_Stamps =	((23-MAR-95,18:50:21.2), (*-*-*,*:*:*.*), (23-MAR-95,19:01:34.0))

Example 2: Hand-off-auto switch.

In this example the contacts of a three-position hand-off-auto switch are monitored as binary inputs. If the first input is active, then the present value is set to 1 (hand); if the second input is active, the present value is 2 (off); and if the third input is active, then the present value is 3 (auto). The object does not support intrinsic reporting.

Property:	Object_Identifier =	(Multi-state Input, Instance 2)
Property:	Object_Name =	"H-O-A"
Property:	Object_Type =	MULTISTATE_INPUT
Property:	Present_Value =	1
Property:	Description =	"Hand-Off-Auto 1"
Property:	Device_Type =	"ZZZ switch"
Property:	Status_Flags =	{FALSE, FALSE, FALSE, FALSE}
Property:	Event_State =	NORMAL
Property:	Reliability =	NO_FAULT_DETECTED
Property:	Out_Of_Service =	FALSE

Property:	Number_Of_States =	3
Property:	State_Text =	("Hand", "Off", "Auto")

D.19 Examples of a Multi-state Output Object

Example 1: Two-speed fan.

In this example Output #1 is connected to the coil of the low-speed starter and Output #2 is connected to the coil of the high-speed starter. If the low-speed starter is "off," then the high-speed starter is disabled. The table below shows the relationship between the two outputs and the present value. The actual logic used to determine, establish, and use the present value is a local matter. The object supports intrinsic reporting.

Present Value	Low-Speed Starter	High-Speed Starter
1	OFF (inactive)	OFF (inactive)
2	ON (active)	OFF (inactive)
3	ON (active)	ON (active)

Property:	Object_Identifier =	(Multi-state Output, Instance 1)
Property:	Object_Name =	"Fan1_Output"
Property:	Object_Type =	MULTI_STATE_OUTPUT
Property:	Present_Value =	2
Property:	Description =	"2-speed Fan#1"
Property:	Device_Type =	"ABC Fan Model A-6"
Property:	Status_Flags =	{FALSE, FALSE, FALSE, FALSE}
Property:	Event_State =	OFFNORMAL
Property:	Reliability =	NO_FAULT_DETECTED
Property:	Out_Of_Service =	FALSE
Property:	Number_Of_States =	3
Property:	State_Text =	{"Off", "On_Low", "On_High"}
Property:	Priority_Array =	{NULL, NULL...2...NULL}
Property:	Relinquish_Default =	1
Property:	Time_Delay =	3
Property:	Notification_Class =	4
Property:	Feedback_Value =	3
Property:	Event_Enable =	{TRUE, TRUE, TRUE}
Property:	Acked_Transitions =	{TRUE, TRUE, TRUE}
Property:	Notify_Type =	EVENT
Property:	Event_Time_Stamps =	((23-MAR-95,18:50:21.2), (*-*-*,*:*:*.*), (21-MAR-95,01:02:03.0))

Example 2: Three-position switch.

In this example the contacts of a three-position switch are controlled as binary outputs. If the present value is 1, the first output will be active. If the present value is 2, then the second output is active. If the present value is 3, then the third output is active. The object does not support intrinsic reporting.

Property:	Object_Identifier =	(Multi-state Output, Instance 2)
Property:	Object_Name =	"3-POS-SW"
Property:	Object_Type =	MULTI_STATE_OUTPUT
Property:	Present_Value =	1
Property:	Description =	"3 POSITION SWITCH #1"