ADDENDA

ANSI/ASHRAE Addenda an, at, au, av, aw, ax, and az to ANSI/ASHRAE Standard 135-2012

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

Approved by the ASHRAE Standards Committee on June 28, 2014; by the ASHRAE Board of Directors on July 2, 2014; and by the American National Standards Institute on July 3, 2014.

These addenda were approved 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 procedures for timely, documented, consensus action on requests for change to any part of the standard. The change submittal form, instructions, and deadlines may be obtained in electronic form from the ASHRAE website (www.ashrae.org) or in paper form from the Manager of Standards.

The latest edition of an ASHRAE Standard may be purchased on the ASHRAE website (www.ashrae.org) or from ASHRAE Customer Service, 1791 Tullie Circle, NE, Atlanta, GA 30329-2305. E-mail: orders@ashrae.org. Fax: 678-539-2129. Telephone: 404-636-8400 (worldwide), or toll free 1-800-527-4723 (for orders in US and Canada). For reprint permission, go to www.ashrae.org/permissions.

 ISSN 1041-2336

Approved American National Standard
ANSI
www.ansi.org

SPECIAL NOTE

This American National Standard (ANS) is a national voluntary consensus standard developed under the auspices of 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 committee members may or may not be ASHRAE members, 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, or
d. permission to reprint portions of the Standard.

DISCLAIMER

ASHRAE uses its best efforts to promulgate Standards and Guidelines 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, installed, or operated in accordance with ASHRAE's Standards or Guidelines or that any tests conducted under its Standards or Guidelines 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 or Guideline and in marking of equipment and in advertising, no claim shall be made, either stated or implied, that the product has been approved by ASHRAE.

[This foreword and the "rationales" on the following pages are not part of this standard. They are merely informative and do not contain requirements necessary for conformance to the standard.]

FOREWORD

The purpose of this addendum is to present a proposed change for public review. These modifications are the result of change proposals made pursuant to the ASHRAE continuous maintenance procedures and of deliberations within Standing Standard Project Committee 135. The proposed changes are summarized below.

135-2012*an*-1 Add Extended Length MS/TP Frames, p. 3
135-2012*an*-2 Add Procedure for Determining Maximum Conveyable APDU, p. 35

In the following document, language to be added to existing clauses of ANSI/ASHRAE 135-2012 and Addenda is indicated through the use of *italics*, while deletions are indicated by ~~strikethrough~~. Where entirely new subclauses are proposed to be added, plain type is used throughout. Only this new and deleted text is open to comment at this time. All other material in this addendum is provided for context only and is not open for public review comment except as it relates to the proposed changes.

135-2012*an*-1 Add COBS-Encoded MS/TP Frames

Rationale

BACnet now supports higher baud rates for MS/TP. Adding the capability to carry full ethernet-sized frames will improve throughput and open up possibilities for future applications.

However, increasing the size of MS/TP payloads beyond the current 501-octet maximum without also replacing the CRC-16-CCITT (a.k.a CRC-CCITT, CCITT-16 CRC, etc.) with a 32-bit CRC will result in an unacceptable increase in the probability of undetected bit errors.

There have been significant advances in coding theory since MS/TP was initially specified. In particular, the design space of CRCs for particular applications (e.g. embedded control networks) has been well explored. The probability of undetected errors (P_{ud}, bit errors passed up to the client) is affected by a combination of CRC length, CRC polynomial, bit error rate, and data word (payload) length.

Among the most recent (and readable) published surveys on the evaluation of CRCs are three papers by Dr. Philip Koopman, formerly of United Technologies and now on the faculty at Carnegie Mellon University [1][2][3]. Since the CRC-16-CCITT is widely used, many studies use it as a basis for comparison. The figure below demonstrates that, all other factors being equal, increasing the code word length (the covered data plus the CRC field) by a factor of three, from 512 octets (4096 bits) to 1536 octets (12288 bits) increases the probability of undetected errors by approximately two orders of magnitude (i.e. the effect is non-linear).

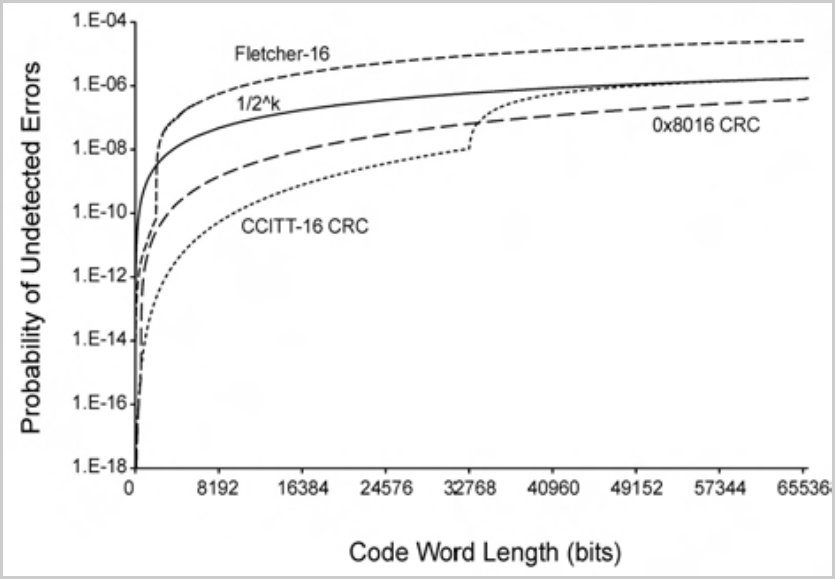


Fig. 10. Probability of undetected errors for a 16-bit Fletcher checksum and two 16-bit CRCs at a BER of 10^{-5}. The data values for the Fletcher checksum are the mean of 10 trials using random data. [3]

When we consider that nodes sending large frames are likely to utilize the recently approved higher baud rates (which may further decrease bit error rate margin) the combined effect may be even greater. The large frame discussion should focus on how to a) incorporate a larger CRC into the present MS/TP standard and b) which CRC polynomial should be selected.

Solution Summary:

This proposal can be summarized as a more robust alternative to the present MS/TP design, with BACnet NPDU length increased to 1497 octets and a larger (32-bit) CRC that affords greater protection from undetected bit errors. The new frame types proposed here are used to indicate frames that are covered by a larger CRC. The recommended 32-bit polynomial is CRC-32K (Koopman), which has better error detection properties than the IEEE 802.3 Frame Check Sequence polynomial for data lengths up to 114,663 bits [1].

Lastly, MS/TP has not previously defined a method to escape the X'55' that is used to delimit frames. In some cases, this can lead to loss of frame synchronization and dropped frames. Extending the data field would make this problem worse. Therefore, this proposal uses an octet stuffing algorithm called Constant Overhead Byte Stuffing (COBS) to encode the data and CRC-32K fields of extended length frames and thereby eliminates preamble sequences from those fields.

[Append to **Clause 5.2.1.2**, p. 20]

See clause 19.X for an approach to determine the maximum APDU conveyable by the internetwork.

[Change **Table 6-1**, p. 53]

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~~ *1497* 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
ZigBee, as defined in Annex O	501 octets

[Change **Clause 9.1.1.2**, p. 79]

9.1.1.2 Semantics of the Service Primitive

...

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~~ *the data link interface* 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. *There is sufficient information associated with the LSDU for the MS/TP entity to determine the length of the data unit.*

...

[Change **Clause 9.1.1.4**, p. 79]

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. *BACnet NPDUs less than 502 octets in length are conveyed using BACnet Data Expecting Reply or BACnet Data Not Expecting Reply frames (see subclause 9.3). BACnet NPDUs between 502 and 1497 octets in length, inclusive, are conveyed using BACnet Extended Data Expecting Reply or BACnet Extended Data Not Expecting Reply frames.*

...

[Change **Clause 9.1.2.2**, p. 80]

9.1.2.2 Semantics of the Service Primitive

...

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~~ *the data link interface* 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. *There is sufficient information associated with the LSDU for the MS/TP entity to determine the length of the data unit. A data unit larger than the maximum supported NPDU shall be silently dropped.*

...

[Change **Clause 9.3**, p. 92]

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 *and frame is a non-encoded type*)
Data CRC	(present only if Length is non-zero *and frame is a non-encoded type*) two octets, least significant octet first
Encoded Data	*(present only if frame is a COBS-encoded type)*
Encoded CRC-32K	*(present only if frame is a COBS-encoded type)* *five octets; CRC-32K generated according to subclause 9.6 and COBS-encoded according to subclause 9.10*
(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
32	*BACnet Extended Data Expecting Reply*
33	*BACnet Extended Data Not Expecting Reply*

Frame Types 8 through *31 and 34 through* 127 are reserved by ASHRAE. *Open standard development organizations wishing to convey network protocols other than BACnet over MS/TP may apply for a Frame Type allocation from the reserved range by contacting the ASHRAE Manager of Standards.*

Frame Types 32 through 127 indicate COBS-encoded frames and shall convey Encoded Data and Encoded CRC-32K fields (see subclause 9.10). All other values for Frame Type indicate a non-encoded frame. Support for COBS-

encoded BACnet MS/TP frames is required for BACnet routing nodes and optional for non-routing nodes. Devices that support COBS-encoded BACnet MS/TP frames shall support both BACnet Extended Data Expecting Reply and BACnet Extended Data Not Expecting Reply Frame Types.

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~~ *all MS/TP* 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.

For non-encoded frames, the ~~The~~ Length field specifies the length in octets of the Data field *and shall be between 0 and 501 octets.*

For non-encoded frames, the ~~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.~~

For COBS-encoded frames, the Length field specifies the combined length of the Encoded Data and Encoded CRC-32K fields in octets, minus two. (Two octets are subtracted to maintain backward compatibility with devices that implement the SKIP_DATA state in their Receive Frame state machine.)

For COBS-encoded frames, the Length field shall be in the range $N_{min\_COBS\_length}$ *to* $N_{min\_COBS\_length}$ *(see subclause 9.5.3) and the Encoded Data and Encoded CRC-32K fields shall always be present.*

Subclause 9.6 and Annex G describe in detail the generation and checking of the Header, ~~and~~ Data CRC*, and CRC-32K* octets.

Subclause 9.10 and Annex X describe in detail the COBS-encoding of the Encoded Data and Encoded CRC-32K fields.

[Change **Clause 9.3.9**, renumbering to make room for new frame types, p. 93]

9.3.9*11* 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~~ *proprietary* frame shall be in the range of 2 to 501 octets.

[Insert new **Clauses 9.3.9-10**, p. 93]

9.3.9 Frame Type 32: BACnet Extended Data Expecting Reply

This COBS-encoded frame is used by master nodes to convey the data parameter of a DL_UNITDATA.request whose data_expecting_reply parameter is TRUE and whose data parameter length is between 502 and 1497 octets, inclusive.

9.3.10 Frame Type 33: BACnet Extended Data Not Expecting Reply

This COBS-encoded frame is used by master nodes to convey the data parameter of a DL_UNITDATA.request whose data_expecting_reply parameter is FALSE and whose data parameter length is between 502 and 1497 octets, inclusive.

[Change **Clause 9.5.2**, p. 95]

9.5.2 Variables

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

...

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[]. *Routing nodes shall support the maximum NPDU size. Non-routing nodes may support a smaller value. Devices that support COBS-encoded frames shall include additional octets to account for encoding overhead. The overhead is calculated as the maximum supported NPDU size divided by 254 (any fractional part is rounded up to the nearest integer) plus five octets for the Encoded CRC-32K. For example, the InputBufferSize required for a device that supports the maximum BACnet NPDU size is 1497 + 6 + 5 = 1508.*

GoodHeader *A Boolean flag set to TRUE or FALSE by the CheckHeader procedure (see 9.5.8).*

CRC32K *Used by devices that support COBS-encoded frames to accumulate the CRC-32K on the Encoded Data field.*

...

[Change **Clause 9.5.3**, p. 96]

9.5.3 Parameters

Parameter values used in the description:

...

$N_{min\_COBS\_type}$ *The first COBS-encoded Frame Type value: 32.*

$N_{max\_COBS\_type}$ *The last COBS-encoded Frame Type value: 127.*

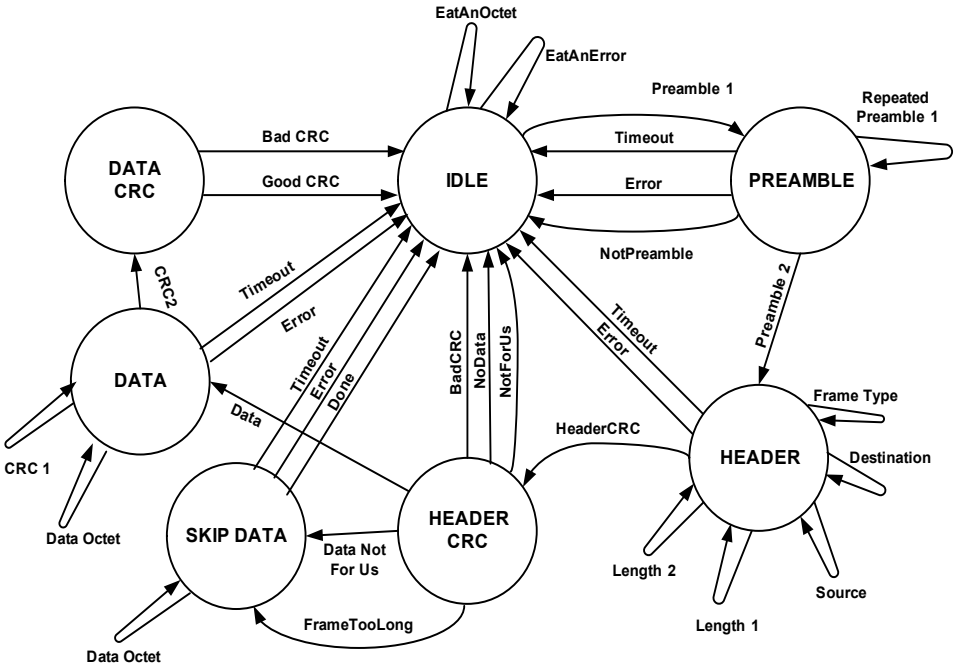
$N_{min\_COBS\_length}$ *The minimum valid Length value of any COBS-encoded frame: 5. The theoretical minimum Length is calculated as follows: COBS-encoded frames must contain at least one data octet. The minimum COBS encoding overhead for the Encoded Data field is one octet. The size of the Encoded CRC-32K field is always five octets. Adding the lengths of these fields and subtracting two (for backward compatibility) results in a minimum Length value of five (1 + 1 + 5 - 2). In practice, the minimum Length value is determined by the network-layer client and is likely to be larger (e.g. for BACnet the minimum Length is 502 + 1 + 3 = 506.*

$N_{max\_COBS\_length}$ *The maximum valid Length value of any COBS-encoded frame: 2043. The theoretical maximum Length is calculated as follows: the largest data parameter that any future network client may specify is 2032 octets (this is near the limit of the CRC-32K's maximum error-detection capability). The worst-case COBS encoding overhead for the Encoded Data*

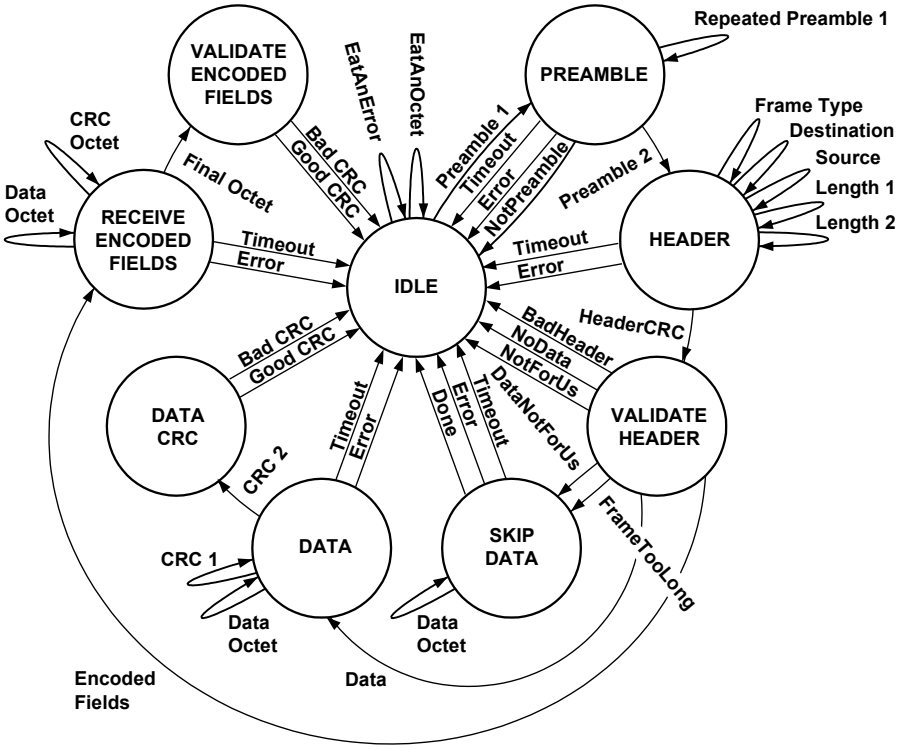
field would be 2032 / 254 = 8 octets. Adding in the size adjustment for the Encoded CRC-32K results in a maximum Length value of 2032 + 8 + 3 = 2043. In practice, the maximum Length value is determined by the network-layer client and is likely to be smaller (e.g. for BACnet the maximum Length is 1497 + 6 + 3 = 1506.

[Change **Figure 9-3**, p. 97]

[Existing Figure]



[Revised Figure]



[Change **Clause 9.5.4.3**, p. 98]

...

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; *call the CheckHeader procedure defined in subclause 9.5.8;* and enter the VALIDATE_HEADER state.

[Change **Clause 9.5.4.4**, p. 99]

9.5.4.4 VALIDATE_HEADER

In the VALIDATE_HEADER state, the node validates the CRC on the fixed message header *and tests for illegal values in other header fields*.

~~BadCRC~~*BadHeader*

If ~~the value of HeaderCRC is not X '55'~~ *GoodHeader is FALSE,*

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'~~ *GoodHeader is TRUE* and DataLength is zero 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.

DataNotForUs

If ~~the value of the HeaderCRC is X '55'~~ *GoodHeader is TRUE* and DataLength is not zero and the value of DestinationAddress is not equal to either TS (this station) or 255 (broadcast),

then set Index to zero and enter the SKIP_DATA state to consume the ~~data~~ *Data* and ~~data~~ *Data* CRC *or Encoded Data and Encoded CRC-32K* portions of the frame.

FrameTooLong

If ~~the value of the HeaderCRC is X '55'~~ *GoodHeader is TRUE* 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~~ *DataLength* has been received, set Index to zero, and enter the SKIP_DATA state to consume the ~~data~~ *Data* and ~~data~~ *Data* CRC *or Encoded Data and Encoded CRC-32K* portions of the frame.

NoData

If ~~the value of the HeaderCRC is X '55'~~ *GoodHeader is TRUE* 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 the HeaderCRC is X '55'~~ *GoodHeader is TRUE* 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 *and either this device does not support COBS-encoded frames or FrameType indicates a non-encoded frame,*

then set Index to zero; set DataCRC to X'FFFF'; and enter the DATA state to receive the data portion of the frame.

EncodedFields

If GoodHeader is TRUE and the value of DestinationAddress is equal to either TS (this station) or 255 (broadcast) and DataLength is less than or equal to InputBufferSize and this device supports COBS-encoded frames and FrameType indicates a COBS-encoded frame,

then set Index to zero; set CRC32K to X'FFFFFFFF'; and enter the RECEIVE_ENCODED_FIELDS state to receive the Encoded Data and Encoded CRC-32K fields of the frame.

[Insert new **Clauses 9.5.4.8-9**, p. 101]

9.5.4.8 RECEIVE_ENCODED_FIELDS

In the RECEIVE_ENCODED_FIELDS state, the node waits for and decodes the Encoded Data and Encoded CRC-32K fields of a frame according to the procedure described in 9.10.3. If the device does not support COBS-encoded frames, this state is unreachable and should not be implemented.

Timeout

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

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 minus 3,

then set DataAvailable to FALSE; set SilenceTimer to zero; accumulate the contents of DataRegister into CRC32K; save the contents of DataRegister at InputBuffer[Index]; increment Index by 1; and enter the RECEIVE_ENCODED_FIELDS state.

CRCOctet

If ReceiveError is FALSE and DataAvailable is TRUE and Index is greater than or equal to DataLength minus 3 and Index is less than DataLength plus 1,

then set DataAvailable to FALSE; set SilenceTimer to zero; save the contents of DataRegister at InputBuffer[Index]; increment Index by 1; and enter the RECEIVE_ENCODED_FIELDS state.

FinalOctet

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; save the contents of DataRegister at InputBuffer[Index]; decode the Encoded Data and Encoded CRC-32K fields according to subclause 9.10.3; and enter the VALIDATE_ENCODED_FIELDS state.

9.5.4.9 VALIDATE_ENCODED_FIELDS

In the VALIDATE_ENCODED_FIELDS state, the node validates the received CRC-32K of the frame. If the device does not support COBS-encoded frames, this state is unreachable and should not be implemented.

BadCRC

If the value of CRC32K is not X'0843323B',

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 CRC32K is X'0843323B',

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.

[Change **Clause 9.5.5**, p. 101]

9.5.5 The SendFrame Procedure

The transmission of an MS/TP frame proceeds as follows:

Procedure SendFrame

If Frame Type is less then $N_{min\_COBS\_type}$ *or Frame Type is greater than* $N_{max\_COBS\_type}$,

then call SendNonEncodedFrame,

else call SendCOBS_EncodedFrame.

9.5.5.1 SendNonEncodedFrame Procedure for Non-Encoded Frame Types

(a) If SilenceTimer is less than $\mathbf{T_{turnaround}}$, wait ($\mathbf{T_{turnaround}}$ - SilenceTimer).

...

(k) Disable the transmit line driver *and enable the receiver*.

...

[Insert new **Clause 9.5.5.2**, p. 101]

9.5.5.2 SendCOBS_EncodedFrame Procedure for COBS-Encoded Frame Types

This procedure shall be implemented by devices that support COBS-encoded Frame Types.

(a) COBS-encode the NPDU according to subclause 9.10.2 to generate the Encoded Data field. Set DataLength equal to the length of the Encoded Data field in octets, plus three.

(b) If SilenceTimer is less than $\mathbf{T_{turnaround}}$, wait ($\mathbf{T_{turnaround}}$ - SilenceTimer).

(c) Disable the receiver and enable the transmit line driver.

(d) Transmit the preamble octets X'55', X'FF'. As each octet is transmitted, set SilenceTimer to zero.

(e) Initialize HeaderCRC to X'FF'.

(f) 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.

(g) Transmit the ones-complement of HeaderCRC. Set SilenceTimer to zero.

(h) Initialize CRC32K to X'FFFFFFFF'.

(i) Transmit the Encoded Data octets. Accumulate each octet into CRC32K. As each octet is transmitted, set SilenceTimer to zero.

(j) Compute the ones-complement of CRC32K and order it least significant octet first. COBS-encode the four octets according to subclause 9.10.2 to generate the five-octet Encoded CRC-32K field.

(k) Transmit the Encoded CRC-32K octets. As each octet is transmitted, set SilenceTimer to zero.

(l) Wait until the final stop bit of the last Encoded CRC-32K octet has been transmitted but not more than $\mathbf{T_{postdrive}}$.

(m) Disable the transmit line driver and enable the receiver.

(n) Return.

[Change **Clause 9.5.6.2**, p. 104]

9.5.6.2 IDLE

In the IDLE state, the node waits for a 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, *BACnet Extended Data Not Expecting Reply,* or a ~~proprietary type~~ *FrameType* 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~~*Test_Request*, *BACnet Extended Data Expecting Reply,* or a ~~proprietary type~~ *FrameType* 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.

BroadcastDataNeedingReply

If ReceivedValidFrame is TRUE and DestinationAddress is equal to 255 (broadcast) and FrameType is equal to *either* BACnet Data Expecting Reply *or BACnet Extended Data Expecting Reply,*

then indicate successful reception to the higher layers; set ReceivedValidFrame to FALSE; and enter the IDLE state to wait for the next frame.

[Change **Clause 9.5.6.3**, p. 105]

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 *other standard or* proprietary frames.

NothingToSend

If there is no data frame awaiting transmission,

then set FrameCount to $\mathbf{N}_{\mathbf{max\_info\_frames}}$ and enter the DONE_WITH_TOKEN state.

SendNoWait

If the next frame awaiting transmission is of type Test_Response, BACnet Data Not Expecting Reply, *BACnet Extended Data Not Expecting Reply,* a ~~proprietary type~~ *FrameType known to this node* that does not expect a reply, ~~or a frame of type Data Expecting Reply with a DestinationAddress that is equal to 255 (broadcast)~~ *or has a DestinationAddress that is equal to 255 (broadcast) and a FrameType equal to either BACnet Data Expecting Reply or BACnet Extended Data Expecting Reply,*

then call SendFrame to transmit the frame; increment FrameCount; and enter the DONE_WITH_TOKEN state.

SendAndWait

If the next frame awaiting transmission is of type Test_Request, ~~BACnet Data Expecting Reply,~~ a ~~proprietary type~~ *FrameType known to this node* that expects a reply, ~~or a frame of type Data Expecting Reply with a DestinationAddress that is not equal to 255 (broadcast)~~ *or has a DestinationAddress that is not equal to 255 (broadcast) and a FrameType equal to either BACnet Data Expecting Reply or BACnet Extended Data Expecting Reply,*

then call SendFrame to transmit the data frame; increment FrameCount; and enter the WAIT_FOR_REPLY state.

[Change **Clause 9.5.6.4**, p. 105]

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 $\mathbf{T}_{\mathbf{reply\_timeout}}$,

then assume that the request has failed. Set FrameCount to $\mathbf{N}_{\mathbf{max\_info\_frames}}$ 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 $\mathbf{T}_{\mathbf{reply\_timeout}}$ 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 $\mathbf{T}_{\text{reply\_timeout}}$ and ReceivedValidFrame is TRUE and DestinationAddress is equal to TS (this station) and FrameType is equal to Test_Response, BACnet Data Not Expecting Reply, *BACnet Extended Data Not Expecting Reply,* or a ~~proprietary type~~ *FrameType known to this node* 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 $\mathbf{T}_{\text{reply\_timeout}}$ 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 $\mathbf{T}_{\text{reply\_timeout}}$ 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, *BACnet Extended Data Not Expecting Reply,* or ~~proprietary reply frame~~ *a FrameType known to this node that indicates a reply*,

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.

…

[Change **Clause 9.5.6.9**, p. 109]

9.5.6.9 ANSWER_DATA_REQUEST

The ANSWER_DATA_REQUEST state is entered when a BACnet Data Expecting Reply, *BACnet Extended Data Expecting Reply,* a Test_Request, or a ~~proprietary frame~~ *FrameType known to this node* 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.

…

[Change **Clause 9.5.7.2**, p. 110]

9.5.7.2 IDLE

In the IDLE state, the node waits for a 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~~*Test_Request*, or a ~~proprietary type~~ *FrameType* 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~~ *FrameType* 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~~ *FrameType* 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~~*Test_Request*, or a ~~proprietary type~~ *FrameType* 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.

[Change **Clause 9.5.7.3**, p. 110]

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~~ *FrameType known to this node* that expects a reply is received.

Reply

If a reply is available from the higher layers within $\mathbf{T_{reply\_delay}}$ 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.

…

[Add **Clause 9.5.8**, p. 111]

9.5.8 The CheckHeader Procedure

This procedure checks for invalid header fields and sets the value of GoodHeader accordingly.

If the value of HeaderCRC is not X '55',
or the value of SourceAddress is 255 (broadcast) ,
or the value of FrameType is less than $N_{min\_COBS\_type}$ and DataLength is greater than 501,
or the value of FrameType is greater than $N_{max\_COBS\_type}$ and DataLength is greater than 501,
or the value of FrameType is greater than or equal to $N_{min\_COBS\_type}$ and
the value of FrameType is less than or equal to $N_{max\_COBS\_type}$ and
DataLength is less than $N_{min\_COBS\_length}$,
or the value of FrameType is greater than or equal to $N_{min\_COBS\_type}$ and

the value of FrameType is less than or equal to $N_{max\_COBS\_type}$ and
DataLength is greater than $N_{max\_COBS\_length}$,

then set GoodHeader to FALSE, else set GoodHeader to TRUE.

[Change **Clause 9.6**, p. 111]

9.6 Cyclic Redundancy Check (CRC)

…

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. *Annex G describes the implementation of the CRC algorithms in software.*

9.6.1 Frame Header CRC

The MS/TP frame header uses the polynomial

…

9.6.2 Data CRC

The MS/TP ~~data~~ *Data* CRC uses the ~~CRC-CCITT~~ *CRC-16-CCITT* polynomial

…

9.6.3 CRC-32K

COBS-encoded MS/TP frames use the CRC-32K (Koopman) polynomial

$$G(X) = X^{32} + X^{30} + X^{29} + X^{28} + X^{26} + X^{20} + X^{19} + X^{17} + X^{16} + X^{15} + X^{11} + X^{10} + X^7 + X^6 + X^4 + X^2 + X + 1$$
$$= (X + 1)(X^3 + X^2 + 1)(X^{28} + X^{22} + X^{20} + X^{19} + X^{16} + X^{14} + X^{12} + X^9 + X^8 + X^6 + 1)$$

In operation, at the transmitter, the initial content of the CRC-32K 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 Encoded Data field (see subclause 9.10.2). The ones-complement of the resulting remainder is ordered least-significant octet first and then COBS-encoded to generate the five-octet Encoded CRC-32K field.

At the receiver, the initial content of the CRC-32K 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 Encoded Data field octets of the incoming message before they are decoded. The Encoded CRC-32K field is then decoded (see 9.10.3) and the register is modified by division by the generator polynomial G(x) of the four decoded octets. In the absence of transmission errors, the resultant remainder will be

0000 1000 0100 0011 0011 0010 0011 1011 (x^0 through x^{31}, respectively).

NOTE: The initialization of the CRC-32K register to all ones and the complementing of the register before transmission prevent the CRC-32K from having a value of zero if the covered field is all zeros.

~~Annex G describes the implementation of the CRC algorithms in software.~~

[Change **Clause 9.7.1**, p. 112]

9.7.1 Routing of *BACnet* Messages from MS/TP

When a *BACnet* network entity with routing capability receives from a directly connected MS/TP data link a NPDU whose 'data_expecting_reply' parameter is TRUE and the NPDU is to be routed to another *BACnet* 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.

BACnet routers directly connected to MS/TP links must be updated to support COBS-encoded BACnet MS/TP frame types (i.e. BACnet Extended Data Expecting Reply and BACnet Extended Data Not Expecting Reply) before any non-routing devices installed on the link can utilize COBS-encoded frames. This does not, however, ensure that all links on the path from a given source to a given destination network have been updated. Before sending large MS/TP frames, a sending device should determine whether the given destination network is reachable using large MS/TP frames. A procedure by which this can be achieved is described in Clause 19.X.

[Change **Clause 9.7.2**, p. 112]

9.7.2 Routing of *BACnet* Messages to MS/TP

When a *BACnet* network entity issues a ~~DL_UNITDATA.request~~ *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.

…

[Insert new **Clause 9.10**, p. 113]

9.10 COBS (Consistent Overhead Byte Stuffing) Encoding

The original MS/TP specification did not prevent preamble sequences from appearing in the Data or Data CRC fields. This resulted in lost synchronization or dropped frames under certain circumstances. The Receive Frame state machine was later revised to mitigate this problem, but many deployed MS/TP devices do not have the update.

In order to decrease the likelihood of compatibility problems in environments with a mix of new and legacy devices, all MS/TP devices that support COBS-encoded frames shall COBS-encode the Encoded Data and Encoded CRC-32K fields before transmission and decode these fields upon reception. The result of this encoding is to remove X'55' octets from these portions of the frame, therefore preamble sequences cannot appear "on the wire" except where intended to indicate the beginning of a frame.

9.10.1 COBS Description

This subclause provides a basic description of the COBS encoding excerpted from "Consistent Overhead Byte Stuffing" by S. Cheshire and M. Baker. A reference to this paper appears in Clause 25.

COBS performs a reversible transformation on a data field to eliminate a single octet value (e.g. X'55') from it. Once eliminated from the data, that octet value may then be safely used in the framing marker (preamble). The most efficient way to accomplish this is to first eliminate all X'00' (zero) octets as described below, then exclusive OR (XOR) the output with the octet selected for removal.

COBS first takes its input data and logically appends a single zero octet. It then locates all the zero octets in the frame (including the appended one) and then divides the frame at these boundaries into one or more zero-terminated chunks. Every zero-terminated chunk contains exactly one zero octet, and that zero is always the last octet of the chunk. A chunk may be as short as one octet (i.e. a chunk containing a single zero octet) or as long as the entire input sequence (i.e. no zeros in the data except the appended one).

COBS encodes each zero-terminated chunk using one or more variable length COBS code blocks. Chunks of 254 octets or less (counting the terminating zero) are encoded as single COBS code blocks. Chunks longer than 254 octets are encoded using multiple code blocks, as described later in this subclause. After a packet's constituent chunks have all been encoded using COBS code blocks and concatenated, the resulting aggregate block of data is completely free of zero octets so zeros can then be placed around the encoded packet to mark clearly where it begins and ends.

A COBS code block consists of a single code octet, followed by zero or more data octets. The number of data octets is represented by the code octet. For codes X'01' to X'FE', the meaning of each code block is that it represents the sequence of data octets contained within the code block, followed by an implicit zero octet. The zero octet is implicit, meaning it is not actually contained within the sequence of data octets in the code block, but it is counted.

These code blocks encode data without adding any overhead. Each code block begins with one code octet followed by *n* data octets, and represents *n* data octets followed by one trailing zero octet. Thus the code block adds no overhead to the data: a chunk (*n*+1) octets long is encoded using a code block (1+*n*) octets long. These basic codes are suitable for encoding zero-terminated chunks up to 254 octets in length, but some of the zero-terminated chunks that make up a packet may be longer than that. To encode these chunks, code X'FF' is defined slightly differently. Code X'FF' represents the sequence of 254 data octets contained within the code block, without any implicit zero. This encoding is illustrated in Figure 9-6 below.

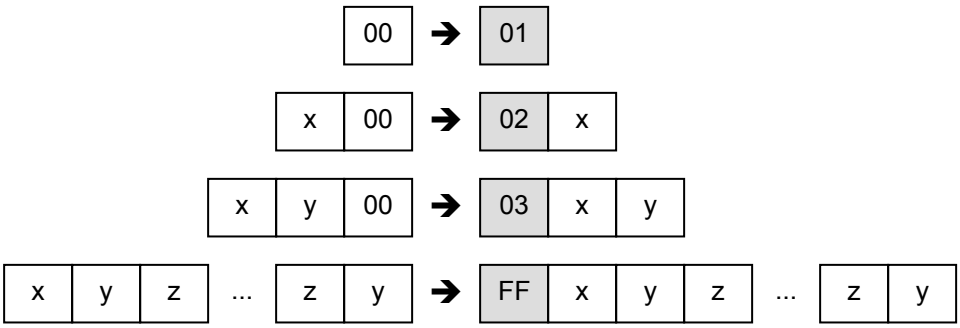


Figure 9-6. COBS Encoding of Chunks into Code Blocks

The two examples in Figure 9-7 show the effect of logically appending a zero (X'00') octet to the input data before encoding. For codes less than 255, a "phantom zero" after the data is used to differentiate between final chunks that end in X'00' and those that do not.

In the upper example, the final chunk of data is the string "Hello" without a terminating zero. A phantom zero is appended to the data and this results in consistent encoding of the final chunk. The phantom zero is discarded by the COBS decoder upon reception and the original data is restored in the receive buffer.

In the lower example, the final chunk of data is again the string "Hello" but this is terminated by a null in the data. Again, a phantom zero is appended to the data. Note the difference after encoding; the result is actually two code blocks transmitted. At the receiver, the phantom zero is discarded and the original data is restored, including the terminating null.

In the case of a final chunk that consists of exactly 254 non-zero octets, a phantom zero SHALL NOT be appended to the encoding since the meaning of code 255 is unambiguous: no implicit zero follows this data.

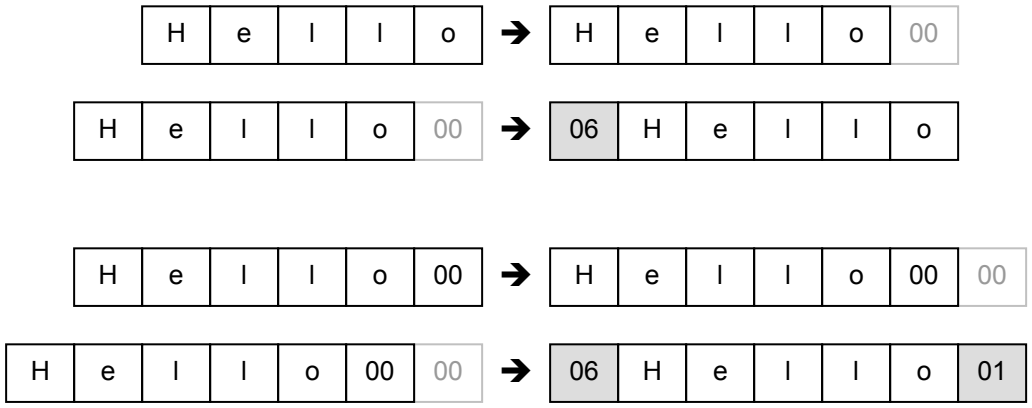


Figure 9-7. Effect of Logically Appending X'00' to the Final Code Block Before Encoding

It can be seen from this example that COBS encoding always adds at least one octet of overhead. In the worst case, COBS encoding adds one octet of overhead for every 254 data octets (a maximum of six octets for a full NPDU, or less than 0.4%). Annex X describes the implementation of the cobs_encode() and cobs_decode() algorithms in software.

9.10.2 Preparing COBS-Encoded MS/TP Frames for Transmission

The COBS encoding described above eliminates all X'00' octets from the input stream and writes the encoded chunks to output. However, the actual goal is to eliminate MS/TP preamble sequences from the Encoded Data and Encoded CRC-32K fields, which can only be accomplished by eliminating either X'55' or X'FF' from these fields. X'FF' cannot be eliminated as this octet may be optionally appended to the end of any frame, leaving X'55' as the only option. The goal is accomplished by first running the encoding function over the input stream, then XOR'ing every octet in the output stream with X'55'. This modifies every octet in the output stream and transforms all occurrences of X'55' into X'00' (which is an acceptable value in the Encoded Data or Encoded CRC-32K fields).

The logical procedure for preparing COBS-encoded MS/TP frames for transmission is as follows:

(a) Pass the client data through the COBS encoder and save the returned length in DataLength. XOR each of the DataLength octets in the output buffer with X'55' to produce the Encoded Data field.

(b) Initialize the value of CRC32K to X'FFFFFFFF'.

(c) Pass the input stream (Encoded Data field) through CalcCRC32K() and accumulate the result in CRC32K. (It is recommended that this be done as each octet is transmitted to reduce latency.)

(d) Take the ones' complement of CRC32K and if necessary reorder it for transmission least significant octet first.

(e) Pass the modified CRC32K through the COBS encoder and XOR each of the five output octets with X'55' to produce the Encoded CRC-32K field.

(f) Add three to DataLength (the five octet length of Encoded CRC-32K field, minus two octets for backward compatibility with legacy MS/TP devices that implement the SKIP_DATA state in their Receive Frame State Machine). Use this modified value of DataLength as the Length field of the outgoing frame.

9.10.3 Decoding COBS-Encoded MS/TP Frames Upon Reception

The logical procedure for decoding COBS-encoded MS/TP frames upon reception is the inverse of encoding and restores the sender's original NPDU at the receiving node. This procedure assumes the Encoded Data and Encoded

CRC-32K octets reside contiguously in InputBuffer in the order received and that the Index and CRC32K variables are initialized on the EncodedFields transition of the Receive Frame Finite State Machine (see subclause 9.5.4.4). The required local variables are initialized as described below. Note that this procedure shows CRC32K being calculated over the Extended Data octets before decoding. However, it is recommended this be performed as the octets are received to reduce latency (see subclause 9.5.4.8).

9.10.3.1 Local Variables

DecodeIndex	Used as an index into InputBuffer[] during decoding. Initialize to zero.
DecodeCount	Used to count down the number of octets remaining to be decoded in the InputBuffer. Initialize to DataLength minus three.
CodeOctet	Used to count down the number of octets remaining to be decoded in the current code block. Initialized in the procedure to the code octet of the current code block.
LastCode	Used to preserve the value (1 - 255) of the code octet of the current code block.
DataOctet	Used to buffer the current octet being decoded in the current code block.

9.10.3.2 Procedure

(a) (Process a code octet in the Encoded Data field)

Set the contents of CodeOctet to the value of InputBuffer[DecodeIndex]; accumulate the contents of CodeOctet into CRC32K; and set CodeOctet equal to the value of CodeOctet XOR'd with X'55'.

If CodeOctet is equal to zero or greater than DecodeCount,

then set the value of CRC32K to X'FFFFFFFF' to indicate that an error has occurred during the reception of a frame and return;

else increment DecodeIndex by 1; decrement DecodeCount by 1; set LastCode to the value of CodeOctet; and decrement CodeOctet by 1.

If CodeOctet is equal to zero then goto step (c) else goto step (b).

(b) (Process a data octet in the Encoded Data field)

Set the contents of DataOctet to the value of InputBuffer[DecodeIndex]; accumulate the contents of DataOctet into CRC32K; set DataOctet equal to the contents of DataOctet XOR'd with X'55'; save the contents of DataOctet at InputBuffer[Index]; increment Index by 1; increment DecodeIndex by 1; decrement DecodeCount by 1; and decrement CodeOctet by 1.

If CodeOctet is equal to zero then goto step (c) else goto step (b).

(c) If DecodeCount is greater than zero and LastCode is not equal to 255,

then save the value zero at InputBuffer[Index]; and increment Index by 1.

If DecodeCount is greater than zero then goto step (a);

else set DataLength to Index; set the value of DecodeCount to five; and goto step (d).

(d) (Process a code octet in the Encoded CRC-32K field)

Set the contents of CodeOctet to the value of InputBuffer[DecodeIndex]; and set CodeOctet equal to the contents of CodeOctet XOR'd with X'55'.

If CodeOctet is equal to zero or greater than DecodeCount,

then set the value of CRC32K to X'FFFFFFFF' to indicate that an error has occurred during the reception of a frame and return;

else increment DecodeIndex by 1; decrement DecodeCount by 1; and decrement CodeOctet by 1.

If CodeOctet is equal to zero then goto step (f) else goto step (e).

(e) (Process a data octet in the Encoded CRC-32K field, i.e. CRC1, CRC2, CRC3, or CRC4)

Set the contents of DataOctet to the value of InputBuffer[DecodeIndex] XOR'd with X'55'; accumulate the contents of DataOctet into CRC32K; increment DecodeIndex by 1; decrement DecodeCount by 1; and decrement CodeOctet by 1.

If CodeOctet is equal to zero then goto step (f) else goto step (e).

(f) If DecodeCount is greater than zero,

then accumulate the value zero into CRC32K; and goto step (d);

else return.

[Add new entries to **Clause 25**, p. 772]

25 REFERENCES

...

IEEE/ACM TRANSACTIONS ON NETWORKING, VOL.7, NO.2, APRIL, 1999, *Consistent Overhead Byte Stuffing*, S. Cheshire and M. Baker.

...

IEEE/IFIP International Conference on Dependable Systems and Networks (DSN 2002), *32-Bit Cyclic Redundancy Codes for Internet Applications*, P. Koopman.

...

IEEE/IFIP International Conference on Dependable Systems and Networks (DSN 2004), *Cyclic Redundancy Code (CRC) Polynomial Selection for Embedded Networks*, P. Koopman and T. Chakravarty

...

IEEE Transactions on Dependable and Secure Computing, Vol. 6, No. 1, January – March 2009, *The Effectiveness of Checksums for Embedded Control Networks*, T. Maxino and P. Koopman

...

[Add new entries to **Clause 25**, **Sources for Reference Material** section, p. 772]

...

Cheshire: http://www.stuartcheshire.org/papers/COBSforToN.pdf

...

Koopman: http://www.ece.cmu.edu/~koopman/projects.html#crc

...

[Change **Annex A**, p. 775]

Data Link Layer Options

...

☐ MS/TP master (Clause 9), baud rate(s): _____

☐ *MS/TP master supports extended length BACnet frames (BACnet Extended Data Expecting Reply and BACnet Extended Data Not Expecting Reply).*

...

[Insert new **Clause G.3** and subclauses, p. 837]

G.3 Calculation of the Encoded CRC-32K

The equivalence of hardware and software methods for calculating CRCs is demonstrated by the examples shown previously in Clauses G.1 and G.2. There are numerous open source examples of CRC algorithms available, including "A Painless Guide to CRC Error Detection Algorithms." It describes a library that can implement arbitrary CRC systems based on input parameters such as CRC length, polynomial, initial value, shift direction (based on the transmission order of the underlying data link), etc.

G.3.1 Sample Implementation of the CRC-32K in C

An example C language implementation for the CRC-32K (Koopman) polynomial is shown below. Inputs are the octet to be processed and the accumulated CRC value (i.e. the value of the CRC32K variable defined in subclause 9.5.2). The function return value is the updated CRC (to be copied back into the CRC32K variable). An example of use is shown in Addendum X.1. Note that Koopman expresses CRC polynomial representations in X^{32} ... X^{1} order (X^{0} is implied because it is always set to X'01'). The polynomial specified in subclause 9.6 is represented in his notation as X'BA0DC66B'. When this is reversed and represented in X^{0} ... X^{31} order (X^{32} is implied because it is always set to X'01'), the equivalent polynomial representation becomes X'EB31D82E'.

```
#include <stdint.h>

/* Accumulate "dataValue" into the CRC in "crc32kValue".
 * Return value is updated CRC.
 *
 * Assumes that "uint8_t" is equivalent to one octet.
 * Assumes that "uint32_t" is four octets.
 * The ^ operator means exclusive OR.
 */
uint32_t
CalcCRC32K(uint8_t dataValue, uint32_t crc32kValue)
{
  uint8_t data, b;
  uint32_t crc;

  data = dataValue;
  crc  = crc32kValue;

  for (b = 0; b < 8; b++) {
    if ((data & 1) ^ (crc & 1)) {
      crc >>= 1;
      crc ^= 0xEB31D82E;  /* CRC-32K polynomial, 1 + x**1 + ... + x**30 (+ x**32) */
    } else {
      crc >>= 1;
    }
    data >>= 1;
  }
  return crc;  /* Return updated crc value */
}
```

In operation, the value of the CRC32K variable is initialized to all ones. During transmission, the Encoded Data octets are passed through the calculation after encoding but before transmission. The ones' complement of the final CRC32K variable value is then ordered least-significant octet first (i.e. in right-to-left order) and COBS-encoded according to subclause 9.10.2. At the receiving end, the Encoded Data octets are passed through the calculation before decoding. Then, the received Encoded CRC-32K field is decoded according to subclause 9.10.3 and the resulting octets (i.e. the ones' complement of the sender's CRC32K, least-significant octet first) are passed through the calculation. If all the octets are received correctly, then the final value of the receiver's CRC32K variable (termed the "residue") will be the constant X'0843323B'. NOTE: the correct residue for a given polynomial may be determined by passing a string of zeros equal in length to the size of the CRC register through the calculation.

```
| 0000 | 1000 | 0100 | 0011 | 0011 | 0010 | 0011 | 1011  |
|<- 0 -><- 8 -><- 4 -><- 3 -><- 3 -><- 2 -><- 3 -><- B ->|
```

As a simplified example of usage, consider a non-encoded data sequence X'01', X'22', X'30':

Description	Value	CRC-32K after octet is processed
		X'FFFFFFFF' (initial value)
first data octet	X'01'	X'56381747'
second data octet	X'22'	X'12557D20'
third data octet	X'30'	X'83DD5A41'
		(ones' complement is X'7C22A5BE')
CRC1 (least significant octet)	X'BE'	X'C0D1F128'
CRC2	X'A5'	X'1C708944'
CRC3	X'22'	X'7FC2C8BD'
CRC4 (most significant octet)	X'7C'	X'0843323B'

Thus, the sender would calculate the CRC-32K on the three octets X'01', X'22', X'30' to be X'83DD5A41'. The ones' complement of this is X'7C22A5BE', which is appended to the frame least significant octet first as X'BE', X'A5', X'22', X'7C'.

The receiver would calculate the CRC-32K on the seven octets X'01', X'22', X'30', X'BE', X'A5', X'22', and X'7C' to be X'0843323B', which is the expected result for a correctly received frame.

G.3.2 Sample Implementation of the CRC-32K in Assembly Language

One way to generate an efficient assembly language implementation of the CRC-32K function is to cross-compile the C example above for a given target processor, examine the assembly language listing, and hand optimize register usage, etc. An assembly language implementation of the CRC-32K for the 8-bit AVR[1] instruction set is presented below. The avr-gcc calling conventions are observed and only scratch registers available for use by the compiler are modified.

```
#include <stdint.h>

; uint32_t
; CalcCRC32K(uint8_t dataValue, uint32_t crcValue)
;
; call:
;
;   r24     dataValue
;   r23-r20 crcValue (LSB-MSB)
;
; return:
;
;   r25-r22 updated crcValue (LSB-MSB)
;
```

[1] AVR is a registered trademark of Atmel Corporation.

```
CalcCRC32K:
    ldi     r18, 0x01   ; r18 = constant ONE
    ldi     r19, 0x00   ; for (b = 0; ...

_for:
; Calculate and save result of ((data ^ crc) & 1)

    mov     r25, r24    ; data lsb
    eor     r25, r20    ; ^crc msb
    and     r25, r18

; crc >>= 1 in either case

    lsr     r23
    ror     r22
    ror     r21
    ror     r20

; If result of previous calculation was '1', accumulate feedback

    cp      r25, r18
    brne    _else

; crc ^= 0xEB31D82E; CRC-32K polynomial, 1 + x**1 + ... + x**30 (+ x**32)

    ldi     r25, 0xEB
    eor     r23, r25
    ldi     r25, 0x31
    eor     r22, r25
    ldi     r25, 0xD8
    eor     r21, r25
    ldi     r25, 0x2E
    eor     r20, r25

_else:
; data >>= 1;

    lsr     r24

    add     r19, r18    ; b++;
    cpi     r19, 0x08
    brlt    _for        ; b < 8;

    movw    r24,r22     ; copy 32-bit return value to r25-r22
    movw    r22,r20
    ret
```

G.3.3 Parallel Implementation of the CRC-32K

Other implementations of the CRC-32K algorithm are possible. It can be seen that the new CRC-32K value may be factored into the exclusive OR of two terms. One term is the most significant octet of the prior CRC-32K value. The other term is a function of the least significant octet of the prior CRC-32K exclusive OR'ed with the data value. A lookup table with 256 elements may be used to quickly determine the value of the second term, using the least significant octet of the prior CRC-32K exclusive OR'ed with the data value as an index. This term may then be exclusive OR'ed with the most significant octet of the prior CRC-32K value to form the new CRC-32K value. The contents of the table may be computed using an implementation similar to the one shown in Clause G.3.1. A sample implementation appears below.

```
#include <stdint.h>

void
CreateCRC32Table()
```

```
{
  uint16_t data;
  uint32_t crc;
  uint16_t b;

  printf( "static const uint32_t CRC32Table[256] = {" );
  for (data = 0; ; ) {
    if (data % 8 == 0)
      printf("\n");

    crc = data & 0xFF;
    for (b = 0; b < 8; b++) {
      if (crc & 1) {
          crc >>= 1;
          crc ^= 0xEB31D82E;
        } else {
          crc >>= 1;
        }
    }
    printf( "0x%08lX", crc );

    if (++data == 256)
      break;
    printf( ", " );
  }
  printf( "\n};\n" );
}


/* Update running "crcValue" with "dataValue"
 * The crcValue must be initialized to all ones.
 *
 * For transmission, the returned value must be complemented and then
 * sent low-order octet first (i.e., right to left).
 *
 * On reception, if Data ends with a correct CRC, the returned value
 * will be 0x0843323B (0000 1000 0100 0011 0011 0010 0011 1011).
 */
uint32_t
LookupCRC32K(uint8_t dataValue, uint32_t crcValue)
{
  crcValue = CRC32Table[(crcValue ^ dataValue) & 0xFF] ^ (crcValue >> 8);
  return (crcValue);
}
```

[Insert new **Annex X**]

ANNEX X - COBS (CONSISTENT OVERHEAD BYTE STUFFING) FUNCTIONS (INFORMATIVE)

(This Annex is not part of this standard but is included for informative purposes only.)

MS/TP did not originally specify a method for escaping preamble sequences that might appear in the Data or Data CRC fields. In certain cases, this might result in dropped frames due to loss of frame synchronization. The SKIP_DATA state was subsequently added to the Receive Frame state machine to mitigate this issue (see subclause 9.5.4.7), but did not resolve the problem for earlier implementations. Encoded MS/TP frames eliminate this problem by using Consistent Overhead Byte Stuffing to remove preamble sequences from the transmitted MSDU fields.

MS/TP implementations that support encoded frames use COBS to encode the Encoded Data and Encoded CRC-32K fields before transmission and decode these fields upon reception. COBS is a reversible run-length encoding method that by design removes X'00' octet values from its input. The encoding overhead is at least one octet per

field and at most one octet per 254 octets of input, or less than 0.4% (as described in subclause 9.10). A selected octet value may be removed by first passing the input stream through the COBS encoder and then XOR'ing the output with the specified octet. In the case of MS/TP, the X'55' preamble octet is specified for removal.

X.1 Preparing a COBS-Encoded MS/TP Frame for Transmission

Encoding proceeds in two passes over the client data. First, the data is passed through the COBS encoder. Then each octet of the encoded output stream is XOR'd with the MS/TP preamble octet X'55' in order to remove all occurrences of this value from the resulting Encoded Data field (any X'55' octet is transformed into a X'00' octet, which is not a preamble value). Observing that the worst-case overhead for COBS-encoding is one octet per 254 octets of input (or six octets for a 1497 octet NPDU), it is possible to set the location of the output buffer just six octets before the location of the input buffer in memory and perform the encoding nearly in place.

The CRC-32K is then calculated over the Encoded Data field, prepared for transmission as described in Annex G.3, COBS-encoded (which adds one octet of overhead), and the resulting octets are each XOR'd with X'55' to produce the Encoded CRC-32K field. The combined length of the Encoded Data and Encoded CRC-32K, minus two octets for compatibility with legacy MS/TP devices, is placed in the MS/TP header Length field before transmission. Observing that the length of the Encoded CRC-32K field is always five octets, the Length field may be computed after encoding the data (as the length of the Encoded Data field plus three octets) and the CRC-32K field may then be computed and in parallel with data transmission.

As an example, the frame encoding of the null-terminated C string "Hello World\n" is shown below. Before encoding, the client data is:

```
0000: 48 65 6C 6C 6F 20 57 6F 72 6C 64 0A 00                "Hello World.."
```

After COBS encoding (shown here for clarity), the output stream is:

```
0000: 0D 48 65 6C 6C 6F 20 57 6F 72 6C 64 0A 01             ".Hello World.."
```

Each octet in the COBS-encoded output stream is XOR'ed with X'55':

```
0000: 58 1D 30 39 39 3A 75 02 3A 27 39 31 5F 54             "X.099:u.:'91_T"
```

The length of the resulting Encoded Data field is 14 octets. After adding the constant five octet length of the Encoded CRC-32K field and subtracting two octets for compatibility with legacy MS/TP devices, the resulting Length field is 17 octets. At this point data transmission may begin and each octet in the Encoded Data field is accumulated in the CRC-32K before it is sent.

The resulting CRC-32K value (shown here for clarity) is:

```
000E: B3 8F 28 CA                                           "..(."
```

After taking the ones' complement and arranging in LSB order (see Annex G.3), the value becomes:

```
000E: 35 D7 70 4C                                           "5.pL"
```

After COBS-encoding and XOR'ing each octet in the output stream with X'55', the Encoded CRC-32K field is ready for transmission:

```
000E: 50 60 82 25 19                                        "P`.%."
```

An example C implementation that combines these two passes is shown below. This algorithm is presented as an example and is not intended to restrict the vendor's implementation of COBS. Since the worst-case overhead of the COBS encoding for a maximum size NPDU is six octets, the output pointer `to` in the example below may be set to `(uint8_t *)(from - 6)` and the encoding performed nearly in place.

```
#include <stddef.h>
#include <stdint.h>

#define CRC32K_INITIAL_VALUE (0xFFFFFFFF)
#define MSTP_PREAMBLE_X55 (0x55)

/*
 * Encodes 'length' octets of data located at 'from' and
 * writes one or more COBS code blocks at 'to', removing
 * any 0x55 octets that may present be in the encoded data.
 * Returns the length of the encoded data.
 */
size_t
cobs_encode (uint8_t *to, const uint8_t *from, size_t length, uint8_t mask)
{
  size_t code_index = 0;
  size_t read_index = 0;
  size_t write_index = 1;
  uint8_t code = 1;
  uint8_t data, last_code;

  while (read_index < length) {
    data = from[read_index++];
    /*
     * In the case of encountering a non-zero octet in the data,
     * simply copy input to output and increment the code octet.
     */
    if (data != 0) {
      to[write_index++] = data ^ mask;
      code++;
      if (code != 255)
        continue;
    }
    /*
     * In the case of encountering a zero in the data or having
     * copied the maximum number (254) of non-zero octets, store
     * the code octet and reset the encoder state variables.
     */
    last_code = code;
    to[code_index] = code ^ mask;
    code_index = write_index++;
    code = 1;
  }
  /*
   * If the last chunk contains exactly 254 non-zero octets, then
   * this exception is handled above (and returned length must be
   * adjusted).  Otherwise, encode the last chunk normally, as if
   * a "phantom zero" is appended to the data.
   */
  if ((last_code == 255) && (code == 1))
    write_index--;
  else
    to[code_index] = code ^ mask;

  return write_index;
}


/*
 * Encodes ’length’ octets of client data located at 'from' and writes
 * the COBS-encoded Encoded Data and Encoded CRC-32K fields at 'to'.
 * Returns the combined length of these encoded fields.
 */
```

```
size_t
frame_encode (uint8_t *to, const uint8_t *from, size_t length)
{
  size_t cobs_data_len, cobs_crc_len;
  uint32_t crc32K;
  int i;

  /*
   * Prepare the Encoded Data field for transmission.
   */
  cobs_data_len = cobs_encode(to, from, length, MSTP_PREAMBLE_X55);
  /*
   * Calculate CRC-32K over the Encoded Data field.
   * NOTE: May be done as each octet is transmitted to reduce latency.
   */
  crc32K = CRC32K_INITIAL_VALUE;
  for (i = 0; i < cobs_data_len; i++) {
    crc32K = CalcCRC32K(to[i], crc32K);   /* See Annex G.3.1 */
  }
  /*
   * Prepare the Encoded CRC-32K field for transmission.
   * NOTE: Assumes a little-endian CPU (otherwise order the
   * octets least-significant first before encoding).
   */
  crc32K = ~crc32K;
  cobs_crc_len = cobs_encode((uint8_t *)(to + cobs_data_len),
                             (const uint8_t *)&crc32K, sizeof(uint32_t),
                             MSTP_PREAMBLE_X55);
  /*
   * Return the combined length of the Encoded Data and Encoded CRC-32K
   * fields. NOTE: Subtract two before use as the MS/TP frame Length field.
   */
  return cobs_data_len + cobs_crc_len;
}
```

X.2 Decoding an Extended MS/TP Frame upon Reception

The frame_decode() function is the inverse of the frame_encode() function shown in the previous section. Note that the octets of the Encoded Data field are accumulated by the CalcCRC32K() function before decoding. The Encoded CRC-32K field is then decoded and the resulting CRC-32K octets are accumulated by the CalcCRC32K() function. If the result is the expected residue value, then the frame was received correctly.

```
#include <stddef.h>
#include <stdint.h>

#define CRC32K_INITIAL_VALUE (0xFFFFFFFF)
#define CRC32K_RESIDUE (0x0843323B)
#define MSTP_PREAMBLE_X55 (0x55)

/*
 * Decodes 'length' octets of data located at 'from' and
 * writes the original client data at 'to', restoring any
 * 'mask' octets that may present in the encoded data.
 * Returns the length of the encoded data or zero if error.
 */
size_t
cobs_decode (uint8_t *to, const uint8_t *from, size_t length, uint8_t mask)
{
  size_t read_index = 0;
  size_t write_index = 0;
  uint8_t code, last_code;
```

```
  while (read_index < length) {
    code = from[read_index] ^ mask;
    last_code = code;
    /*
     * Sanity check the encoding to prevent the while() loop below
     * from overrunning the output buffer.
     */
    if (read_index + code > length) {
      return 0;
    }
    read_index++;

    while (--code > 0) {
      to[write_index++] = from[read_index++] ^ mask;
    }
    /*
     * Restore the implicit zero at the end of each decoded block
     * except when it contains exactly 254 non-zero octets or the
     * end of data has been reached.
     */
    if ((last_code != 255) && (read_index < length)) {
      to[write_index++] = 0;
    }
  }
  return write_index;
}

#define ADJ_FOR_ENC_CRC (5)   /* Set to 3 if passing MS/TP Length field */
#define SIZEOF_ENC_CRC (5)

/*
 * Decodes Encoded Data and Encoded CRC-32K fields at 'from' and
 * writes the decoded client data at 'to'.  Assumes 'length' contains
 * the actual combined length of these fields in octets (that is, the
 * MS/TP header Length field plus two).
 * Returns length of decoded Data in octets or zero if error.
 * NOTE: Safe to call with 'output' <= 'input' (decodes in place).
 */
size_t
frame_decode (uint8_t *to, const uint8_t *from, size_t length)
{
  size_t data_len, crc_len;
  uint32_t crc32K;
  int i;

  /*
   * Calculate the CRC32K over the Encoded Data octets before decoding.
   * NOTE: Adjust 'length' by removing size of Encoded CRC-32K field.
   */
  data_len = length - ADJ_FOR_ENC_CRC;
  crc32K = CRC32K_INITIAL_VALUE;

  for (i = 0; i < data_len; i++) {
    crc32K = CalcCRC32K(from[i], crc32K);   /* See Annex G.3.1 */
  }
  data_len = cobs_decode(to, from, data_len, MSTP_PREAMBLE_X55);
  /*
   * Decode the Encoded CRC-32K field and append to data.
   */
  crc_len = cobs_decode((uint8_t *)(to + data_len),
                        (uint8_t *)(from + length - ADJ_FOR_ENC_CRC),
                        SIZEOF_ENC_CRC,
                        MSTP_PREAMBLE_X55);
```

```
  /*
   * Sanity check length of decoded CRC32K.
   */
  if (crc_len != sizeof(uint32_t)) {
    return 0;
  }
  /*
   * Verify CRC32K of incoming frame.
   */
  for (i = 0; i < crc_len; i++) {
    crc32K = CalcCRC32K((to + data_len)[i], crc32K);
  }
  if (crc32K == CRC32K_RESIDUE) {
    return data_len;
  } else {
    return 0;
  }
}
```

X.4 Example COBS-Encoded Frame - Who-Has Service

This section shows the header, NPDU, and APDU fields of a simple (Who-Has) BACnet request with a very long name consisting of 19 "A" characters, followed by 19 "B" characters, etc., through 19 "Z" characters. The hexadecimal dump of the corresponding COBS-encoded frame follows.

X'55FF'	MS/TP Preamble
X'21'	Frame Type BACnet Extended Data Not Expecting Reply = 33
X'FF'	Destination Address = 255 (broadcast)
X'01'	Source Address = 1
X'0200'	Length = 512
X'4E'	Header CRC = 78
X'01'	Version=1
X'20'	Control Bit 5: 1 = DNET, DLEN, and Hop Count Present
X'FFFF'	DNET = 65535 (Global broadcast)
X'00'	DLEN = 0 (A value of 0 indicates a broadcast on the destination network)
X'FF'	Hop Count = 255
X'10'	PDU Type=1 (Unconfirmed-Service-Request-PDU)
X'07'	Service Choice=7 (Who-Has-Request)
X'3D'	SD Context Tag 3 (Object Name, L>4)
X'FE'	Extended Length > 253
X'01EF'	Extended Length = 495
X'00'	ISO 10646 (UTF-8) Encoding
X'41' ... '5A'	"A ... Z" (494 octets of Object Name data)
X'ACF483BD'	CRC-32K

```
0000  55 FF 21 FF 01 02 00 4E  50 54 75 AA AA 5D AA 45
0010  52 68 AB 54 BA AA 14 14  14 14 14 14 14 14 14 14
0020  14 14 14 14 14 14 14 14  14 17 17 17 17 17 17 17
0030  17 17 17 17 17 17 17 17  17 17 17 17 16 16 16 16
0040  16 16 16 16 16 16 16 16  16 16 16 16 16 16 16 11
0050  11 11 11 11 11 11 11 11  11 11 11 11 11 11 11 11
0060  11 11 10 10 10 10 10 10  10 10 10 10 10 10 10 10
0070  10 10 10 10 10 13 13 13  13 13 13 13 13 13 13 13
0080  13 13 13 13 13 13 13 13  12 12 12 12 12 12 12 12
0090  12 12 12 12 12 12 12 12  12 12 12 1D 1D 1D 1D 1D
```

```
00A0  1D 1D 1D 1D 1D 1D 1D 1D  1D 1D 1D 1D 1D 1D 1C 1C
00B0  1C 1C 1C 1C 1C 1C 1C 1C  1C 1C 1C 1C 1C 1C 1C 1C
00C0  1C 1F 1F 1F 1F 1F 1F 1F  1F 1F 1F 1F 1F 1F 1F 1F
00D0  1F 1F 1F 1F 1E 1E 1E 1E  1E 1E 1E 1E 1E 1E 1E 1E
00E0  1E 1E 1E 1E 1E 1E 1E 19  19 19 19 19 19 19 19 19
00F0  19 19 19 19 19 19 19 19  19 19 18 18 18 18 18 18
0100  18 18 18 18 18 18 18 18  18 18 18 18 18 1B 1B 1B
0110  1B 1B 1B 1B A4 1B 1B 1B  1B 1B 1B 1B 1B 1B 1B 1B
0120  1B 1A 1A 1A 1A 1A 1A 1A  1A 1A 1A 1A 1A 1A 1A 1A
0130  1A 1A 1A 1A 05 05 05 05  05 05 05 05 05 05 05 05
0140  05 05 05 05 05 05 05 04  04 04 04 04 04 04 04 04
0150  04 04 04 04 04 04 04 04  04 04 07 07 07 07 07 07
0160  07 07 07 07 07 07 07 07  07 07 07 07 07 06 06 06
0170  06 06 06 06 06 06 06 06  06 06 06 06 06 06 06 06
0180  01 01 01 01 01 01 01 01  01 01 01 01 01 01 01 01
0190  01 01 01 00 00 00 00 00  00 00 00 00 00 00 00 00
01A0  00 00 00 00 00 00 03 03  03 03 03 03 03 03 03 03
01B0  03 03 03 03 03 03 03 03  03 02 02 02 02 02 02 02
01C0  02 02 02 02 02 02 02 02  02 02 02 02 0D 0D 0D 0D
01D0  0D 0D 0D 0D 0D 0D 0D 0D  0D 0D 0D 0D 0D 0D 0D 0C
01E0  0C 0C 0C 0C 0C 0C 0C 0C  0C 0C 0C 0C 0C 0C 0C 0C
01F0  0C 0C 0F 0F 0F 0F 0F 0F  0F 0F 0F 0F 0F 0F 0F 0F
0200  0F 0F 0F 0F 0F 50 F9 A1  D6 E8
```

135-2012*an*-2 Add Procedure for Determining Maximum Conveyable APDU

Rationale

Add the description of a method for determining the maximum conveyable APDU size to a remote network.

[Insert new **Clause 19.X**, p. 613]

19.X Determining Maximum Conveyable APDU

This clause describes a method for determining the maximum conveyable APDU size to a remote network. This size may be affected by the existence of routers in the path to the remote network that are incapable of handling the maximum APDU size of the end nodes. If this method is used by numerous devices, it consumes considerable communications bandwidth and should be used sparingly since normal communications may be disrupted when used excessively. The method should only be performed when necessary and preferably after a randomly chosen delay in order to reduce the quantity of devices that may be using the method simultaneously. Once a node has determined the maximum conveyable APDU length for a remote network, it shall cache the determined value.

During this test, it is important that the client not generate any other messages to the remote network that are larger than the currently outstanding test message being sent. This will ensure that any Reject-Message-To-Network with a "Reject Reason" of 4 (MESSAGE_TOO_LONG) for the destination network applies to the outstanding test message.

A device on the remote network is selected which is capable of receiving a large APDU. It is a local matter as to how the client determines which device on the remote network to perform the test with.

The client first reads the remote peer's Max_APDU_Length_Accepted property. This step verifies that the remote peer is online and reachable from the client. If no response is received from the remote peer, a different remote peer needs to be selected and the test restarted.

The client then generates a request of a size equal to the smaller of the local maximum APDU the client supports and the maximum APDU length supported by the remote peer. The preferred message to send is a ConfirmedPrivateTransfer-Request with a "Vendor ID" of 0, a "Service Number" of 0 and "Service Parameters" containing a single application tagged OCTET_STRING with N data octets, where N is the number of octets required to generate an APDU of the desired size. This message is reserved by ASHRAE for this use and as such will not result in a change of state of the remote peer (it will have no net effect on the remote peer's operation).

Table 19.X: Calculating Test Message OCTET STRING Size

Desired_APDU_Size	Number of OCTET STRING Data Octets (excluding tag octets)
Up to 263 octets	Desired_APDU_Size - 12
Over 263 octets	Desired_APDU_Size - 14

19.X.1 Example ConfirmedPrivateTransfer Service

This is an example test message using a BACnet confirmed service.

Service = ConfirmedPrivateTransfer
'VendorID'= 0
'ServiceNumber'=0
'ServiceParameters'= (An OCTET STRING with length calculated according to Table 19.X, e.g. 1476 - 14 = 1462)

19.X.2 Encoding for Example

X'00' PDU Type=0 (BACnet-Confirmed-Request-PDU, SEG=0, MOR=0, SA=0)
X'05' Maximum APDU Size Accepted=1476 octets
X'55' Invoke ID=85
X'12' Service Choice=18 (ConfirmedPrivateTransfer)

X'09' SD Context Tag 0 (Vendor ID, L=1)
X'00' 0 (ASHRAE)
X'19' SD Context Tag 1 (Service Number, L=1)
X'00' 0 (ASHRAE Max Conveyable APDU Test Message)
X'2E' PD Opening Tag 2 (Service Parameters)
    X'65' Application Tag 6 (Octet String, L>4)
    X'FE' Extended Length > 253
    X'05B6' Extended Length = 1462
    X'00' ... '00' (1462 octets of octet string data)
X'2F' PD Closing Tag 2 (Service Parameters)

19.X.3 Procedure

Upon sending such a request, the BACnet device can expect one of the following situations to occur:

1) The remote peer responds. Any response by the remote peer, be it a positive or negative response, indicates that the request was successfully conveyed through the internetwork to the peer.
2) A Reject-Message-To-Network with a "Reject Reason" of 4 for the destination network is received. This indicates that the message cannot be conveyed through the internetwork to the peer because it is too large.
3) A Reject-Message-To-Network with a "Reject Reason" of a value other than 4 for the destination network is received indicating that some other failure is stopping completion of the test.
4) No response is received from the remote peer.

If situation 3 occurs, the test cannot continue and the maximum conveyable APDU cannot be determined at this time.

If situation 2 or 4 occurs, the message was too large. A shorter ConfirmedPrivateTransfer should be sent. Given that BACnet networks have specific maximum conveyable APDU sizes, the next size to check should be the next lower maximum conveyable APDU for the standard BACnet data link types. See clause 20.1.2.5 for a list of suggested APDU lengths to test.

If situation 4 repeatedly occurs, regardless of the size of test message used, either the remote peer is now offline, the network has become disjoint, or some other catastrophic failure is occurring. The test should be restarted with a different selected remote peer device.

Clients that send messages to determine the maximum conveyable APDU length for remote networks shall cache the determined values so as to not have to repeat the test whenever the information is needed.

[Add a new entry to **History of Revisions**, p. 1027]

(This History of Revisions is not part of this standard. It is merely informative and does not contain requirements necessary for conformance to the standard.)

HISTORY OF REVISIONS

…	…	**…**
1	16	**Addendum *an* to ANSI/ASHRAE 135-2012** Approved by the ASHRAE Standards Committee June 28, 2014; by the ASHRAE Board of Directors July 7, 2014; and by the American National Standards Institute July 3, 2014. 1. Add Extended Length MS/TP Frames, p. 3 2. Add Procedure for Determining Maximum Conveyable APDU

[This foreword and the "rationales" on the following pages are not part of this standard. They are merely informative and do not contain requirements necessary for conformance to the standard.]

FOREWORD

The purpose of this addendum is to present a proposed change for public review. These modifications are the result of change proposals made pursuant to the ASHRAE continuous maintenance procedures and of deliberations within Standing Standard Project Committee 135. The proposed changes are summarized below.

135-2012*at*-1 Add Interface_Value Property, p. 3

In the following document, language to be added to existing clauses of ANSI/ASHRAE 135-2012 and Addenda is indicated through the use of *italics*, while deletions are indicated by ~~strikethrough~~. Where entirely new subclauses are proposed to be added, plain type is used throughout. Only this new and deleted text is open to comment at this time. All other material in this addendum is provided for context only and is not open for public review comment except as it relates to the proposed changes.

135-2012*at*-1 Add Interface_Value Property

Rationale

For Output objects:
HOA (Hand Off Auto) switches are common in control equipment, and permit a local override of output values. Assuming the HOA control cannot be "trumped" by the controller, there is nothing the controller can do to change the value of the output. There is no other process that can change the value of its output. However, the actual output value is of interest (especially to workstations), but are not reportable via a standard mechanism in BACnet, assuming the value of the HOA switch is known by the controller.

For Input objects:
Manual override of inputs by placing the object out of service is common in control equipment, in cases where a sensor is broken. Assuming that the error condition is fixed, the input should be reconnected to the measured value, but there is no way to verify that the measured value is valid, prior to reconnecting the input to the measured value.
This change adds in a property which allows a controller to expose the actual value of the physical input or output in a standard manner.

[Modify **Table 12.2**, **Analog Input Object Type**, p. 157]

Table 12-2. Properties of the Analog Input Object Type

Property Identifier	Property Datatype	Conformance Code
...	...	...
Interface_Value	*BACnetOptionalREAL*	*O*
...	...	...

[Insert **Clause 12.2.X, Analog Input Object Type**, p. 161]

12.2.X Interface_Value

This read-only property, of type BACnetOptionalREAL, indicates the value, in engineering units, of the physical input. If the BACnet device is not capable of knowing the value of the physical input, then the value of this property shall be NULL.

[Modify **Table 12-3**, **Analog Output Object Type**, p. 162]

Table 12-3. Properties of the Analog Output Object Type

Property Identifier	Property Datatype	Conformance Code
...	...	...
Interface_Value	*BACnetOptionalREAL*	*O*
...	...	...

[Insert **Clause 12.3.X, Analog Output Object Type**, p. 166]

12.3.X Interface_Value

This read-only property, of type BACnetOptionalREAL, indicates the value, in engineering units, of the physical output. If the BACnet device is not capable of knowing the value of the physical output, then the value of this property shall be NULL.

[Modify **Table 12-6**, **Binary Input Object Type**, p. 175]

Table 12-6. Properties of the Binary Input Object Type

Property Identifier	Property Datatype	Conformance Code
...	...	...
Interface_Value	*BACnetOptionalBinaryPV*	*O*
...	...	...

[Insert **Clause 12.6.X, Binary Input Object Type**, p. 180]

12.6.X Interface_Value

This read-only property, of type BACnetOptionalBinaryPV, reflects the logical state of the physical input. If the BACnet device is not capable of knowing the logical state of the physical input, then the value of this property shall be NULL. Otherwise, the logical state of this property shall be either INACTIVE or ACTIVE. The relationship between this property and the physical state of the input is determined by the Polarity property. The possible states are summarized in Table 12-7.

[Modify **Table 12-8**, **Binary Output Object Type**, p. 181]

Table 12-8. Properties of the Binary Output Object Type

Property Identifier	Property Datatype	Conformance Code
...	...	...
Interface_Value	*BACnetOptionalBinaryPV*	*O*
...	...	...

[Insert **Clause 12.7.X, Binary Output Object Type**, p. 186]

12.7.X Interface_Value

This read-only property, of type BACnetOptionalBinaryPV, reflects the logical state of the physical output. If the BACnet device is not capable of knowing the logical state of the physical output, then the value of this property shall be NULL. Otherwise, the logical state of this property shall be either INACTIVE or ACTIVE. The relationship between this property and the physical state of the output is determined by the Polarity property. The possible states are summarized in Table 12-9.

[Modify **Table 12-21**, **Multi-state Input Object Type**, p. 242]

Table 12-21. Properties of the Multi-state Input Object Type

Property Identifier	Property Datatype	Conformance Code
...	...	...
Interface_Value	*BACnetOptionalUnsigned*	*O*
...	...	...

[Insert **Clause 12.18.X, Multi-state Input Object Type**, p. 246]

12.18.X Interface_Value

This read-only property, of type BACnetOptionalUnsigned, reflects the logical state of the physical input. If the BACnet device is not capable of knowing the logical state of the physical input, then the value of this property shall be NULL. Otherwise, the logical state of this property shall be one of 'n' states, where 'n' is the number of

states defined in the Number_Of_States property. The Interface_Value property shall always have a value greater than zero, if not NULL.

[Modify **Table 12-22**, **Multi-state Output Object Type**, p. 247]

Table 12-22. Properties of the Multi-state Output Object Type

Property Identifier	Property Datatype	Conformance Code
...	...	...
Interface_Value	*BACnetOptionalUnsigned*	*O*
...	...	...

[Insert **Clause 12.19.X, Multi-state Output Object Type**, p. 251]

12.19.X Interface_Value

This read-only property, of type BACnetOptionalUnsigned, reflects the logical state of the physical output. If the BACnet device is not capable of knowing the logical state of the physical output, then the value of this property shall be NULL. Otherwise, the logical state of this property shall be one of 'n' states, where 'n' is the number of states defined in the Number_Of_States property. The Interface_Value property shall always have a value greater than zero, if not NULL.

[Modify production **BACnetPropertyIdentifier**, in **Clause**, p. 695]

```
BACnetPropertyIdentifier ::= ENUMERATED { -- see below for numerical order
...
    integral-constant-units         (50),
    interface-value                 (387),
    interval-offset                 (195),
...
    -- see egress-active            (386),
    -- see interface-value          (387)
    ...
    }
```

[Add new productions to **Clause 21**, p. 694]
[Note that a BACnetOptionalREAL production is added identically in Addendum 135-2012*aw*]

```
BACnetOptionalREAL ::= CHOICE {
    null                NULL,
    real                REAL
    }

BACnetOptionalUnsigned ::= CHOICE {
    null                NULL,
    unsigned            Unsigned
    }

BACnetOptionalBinaryPV ::= CHOICE {
    null                NULL,
    binary-pv           BACnetBinaryPV
    }
```

[Add a new entry to **History of Revisions**, p. 1027]

(This History of Revisions is not part of this standard. It is merely informative and does not contain requirements necessary for conformance to the standard.)

HISTORY OF REVISIONS

…	…	**…**
1	16	**Addendum *at* to ANSI/ASHRAE 135-2012** Approved by the ASHRAE Standards Committee June 28, 2014; by the ASHRAE Board of Directors July 7, 2014; and by the American National Standards Institute July 3, 2014. 1. Add Interface_Value Property

[This foreword and the "rationales" on the following pages are not part of this standard. They are merely informative and do not contain requirements necessary for conformance to the standard.]

FOREWORD

The purpose of this addendum is to present a proposed change for public review. These modifications are the result of change proposals made pursuant to the ASHRAE continuous maintenance procedures and of deliberations within Standing Standard Project Committee 135. The proposed changes are summarized below.

135-2012*au*-1 Clarify Authentication Factor Value Encoding Rules, p. 3
135-2012*au*-2 Clarify Coercion Support Requirements, p. 5

In the following document, language to be added to existing clauses of ANSI/ASHRAE 135-2012 and Addenda is indicated through the use of *italics*, while deletions are indicated by ~~strikethrough~~. Where entirely new subclauses are proposed to be added, plain type is used throughout. Only this new and deleted text is open to comment at this time. All other material in this addendum is provided for context only and is not open for public review comment except as it relates to the proposed changes.

135-2012*au*-1 Clarify Authentication Factor Value Encoding Rules

Rationale

A number of issues have been identified in ANNEX P:

1. One of the BCD Authentication Factor formats is ambiguous.
2. The month and year are specified in reverse order for ABA Track 2 format.
3. The data format for ABA Track 2 is clarified in regard to ANSI/ISO BCD (a 5 bit code).
4. Clarify that big-endian is used for encoding for multi-byte values.
5. Require that values that use less than the specified number of octets be padded with leading zeros.
6. Specify that optional fields which are not specified shall be set to '0'.
7. Fixed error in FASC_N_LARGE_BCD encoding.

[Change **Table P-1**, p. 956]

Table P-1: Authentication Factor Value Encoding Rules

Format Type (BACnetAuthenticationFactorType)	Authentication Factor Format Description	Authentication Factor Value Encoding[1,3]
…	…	…
ABA_TRACK2	Magnetic stripe card format (*ANSI/ISO* BCD[2] format) as developed by the banking industry (ABA).	Octet String Size = ~~15~~ *13* Octet[1..10 (MS nibble)] = primary account number (19 digits) Octet[10 (LS nibble) - 12(MS nibble)] = 4 digit expiration date in form ~~"MMYY"~~ *"YYMM"* Octet[12 (LS nibble)..13] = 3 digit service code ~~Octet[14.. 15] = discretionary data (4 digits)~~ *Note: Only the ANSI/ISO BCD data bits (4 bits) are stored with this format.*
…	…	…
FASC_N_LARGE_BCD	Federal Agency Smart Credential - Number. (BCD[2] format) Includes all FASC-N data fields excluding start sentinel, end sentinel, field separators and LRC.	Octet String Size = 16 Octet[1..2] = agency-code (4-digit BCD number) Octet[3..4] = system-site code (4-digit BCD number) Octet[5..7] = credential number (6-digit BCD number) Octet[8 (MS nibble)] = series code (1-digit BCD number) Octet[8 (LS nibble)] = credential code (1-digit BCD number) Octet[9..13] = ~~credential number~~ *person identifier* (10-digit BCD number) Octet[14 (MS nibble)] = organizational category (1 digit BCD number) Octet[14 (LS nibble)..16(MS nibble)] = organizational identifier (4 digit BCD number)

Format Type (BACnetAuthenticationFactorType)	Authentication Factor Format Description	Authentication Factor Value Encoding[1,3]
		Octet[16 (LS nibble)] = association category (1 digit BCD number) -- refer to NIST technical implementation Guidance document for more details
…	…	…

[1] Multi-octet fields shall be conveyed with the most significant octet first *(i.e., big-endian encoding)*
[2] In BCD (binary coded decimal) format, each octet holds two 4-bit BCD encoded decimal digits. Bits 7 to 4 convey the most significant digit, while Bits 3 to 0 convey the least significant digit.
[3] Data fields specified for an encoding which are not contained on the credential shall be set to zero for unsigned values or all zeros for BCD values. BCD values which use less than the allocated space shall be padded with leading zeros in the most significant nibbles as necessary.

135-2012*au*-2 Clarify Coercion Support Requirements

Rationale

The WriteGroup service was added in 135-2010*aa*, but the DS-WG-I-B and DS-WG-E-B BIBBs do not rule on datatype support, implying that DS-WG-I-B devices must support coercions for which they have no actual objects that would use them.

[Modify **Clause K.1.22**, p. 883]

K.1.22 BIBB - Data Sharing-WriteGroup-Internal-B (DS-WG-I-B)

The B device shall contain one or more Channel objects that may be influenced by WriteGroup service requests from device A.

BACnet Service	Initiate	Execute
WriteGroup		x

Devices claiming conformance to DS-WG-I-B shall support configuration of Channel object BACnetDeviceObjectPropertyReference values that contain references to objects inside of device B only.

When performing datatype coercion of values to be written to objects internal to device B, the B device shall only be required to support coercions to datatypes actually used by objects implemented in device B.

[Modify **Clause K.1.23**, p. 883]

K.1.23 BIBB - Data Sharing-WriteGroup-External-B (DS-WG-E-B)

The B device shall contain one or more Channel objects that may be influenced by WriteGroup service requests from device A.

BACnet Service	Initiate	Execute
WriteGroup		x
WriteProperty	x	

Devices claiming conformance to DS-WG-E-B shall also support DS-WG-I-B and DS-WP-A. The B device shall also support configuration of Channel object BACnetDeviceObjectPropertyReference values that contain Device Instances outside of device B, and shall be capable of initiating WriteProperty and optionally WritePropertyMultiple.

When performing datatype coercion of values to be written to objects by device B, the B device shall support coercions to all the datatypes shown in Table 12-63 – Datatype Coercion Rules.

[Add a new entry to **History of Revisions**, p. 1027]

(This History of Revisions is not part of this standard. It is merely informative and does not contain requirements necessary for conformance to the standard.)

HISTORY OF REVISIONS

…	…	**…**
1	16	**Addendum *au* to ANSI/ASHRAE 135-2012** Approved by the ASHRAE Standards Committee June 28, 2014; by the ASHRAE Board of Directors July 7, 2014; and by the American National Standards Institute July 3, 2014. 1. Clarify Authentication Factor Value Encoding Rules 2. Clarify Coercion Support Requirements

[This foreword and the "rationales" on the following pages are not part of this standard. They are merely informative and do not contain requirements necessary for conformance to the standard.]

FOREWORD

The purpose of this addendum is to present a proposed change for public review. These modifications are the result of change proposals made pursuant to the ASHRAE continuous maintenance procedures and of deliberations within Standing Standard Project Committee 135. The proposed changes are summarized below.

135-2012*av*-1 Deprecate Execution of GetAlarmSummary, p. 3
135-2012*av*-2 Deprecate Execution of GetEnrollmentSummary, p. 4

In the following document, language to be added to existing clauses of ANSI/ASHRAE 135-2012 and Addenda is indicated through the use of *italics*, while deletions are indicated by ~~strikethrough~~. Where entirely new subclauses are proposed to be added, plain type is used throughout. Only this new and deleted text is open to comment at this time. All other material in this addendum is provided for context only and is not open for public review comment except as it relates to the proposed changes.

135-2012*av*-1 Deprecate Execution of GetAlarmSummary

Rationale

The execution of the GetAlarmSummary service is being deprecated, since addendum 135-2010*af* made the GetEventInformation service execution required for all devices that support event reporting.

For backward compatibility with older devices which support GetAlarmSummary execution as the only alarm-summarization service, the initiation (client or A-side) remains part of the standard and is required to be supported by devices that claim conformance to the BIBB AE-AS-A.

[Change **Clause 13.2.4**, p. 470]

13.2.4 Event-Summarization

...

Table 13-2: Event-Summarization Services

...

Notification-servers are required to support execution of the GetEventInformation service. Support for the execution of the GetAlarmSummary and GetEnrollmentSummary services is ~~optional,~~ *recommended to not be implemented in devices*. *The specification of this service is retained for historical reference so that implementations of client devices have guidance on how to interoperate with older server devices.*

...

[Change **Clause 13.10**, p. 519]

13.10 GetAlarmSummary Service

The specification of this service is retained for historical reference so that implementations of client devices have guidance on how to interoperate with older server devices. Otherwise, it is deprecated. It is recommended that execution of this service not be implemented in server devices.

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.~~

...

[Change **Clause K.2.7**, p. 885]

K.2.7 BIBB - Alarm and Event-Alarm Summary-B (AE-ASUM-B)

The functionality covered by this BIBB has been deprecated. This BIBB is included solely for historical purposes.

Device B provides summaries of alarms *to* device A.

...

135-2012*av*-2 Deprecate Execution of GetEnrollmentSummary

Rationale

In contrast to the GetEventInformation service, GetEnrollmentSummary is a very complex service and does not provide enough information to acknowledge an event. In addition with the publication of Addendum 135-2010*af*, the execution of the GetEnrollmentSummary service was made optional.

Due to these deficiencies the execution of the GetEnrollmentSummary service is being deprecated. For backward compatibility of alarm clients with devices that support GetEnrollmentSummary execution as the only alarm-summarizations service, the initiation (client or A-side) remains part of the standard and required to be supported for the BIBB AE-AS-A (required for B-AWS and B-OWS).

The execution of the GetEnrollmentSummary service (AE-ESUM-B) is currently required for the device profile B-BC. This is not consistent with Clause 13.2.4. The BIBB AE-ESUM-B is removed from the B-BC device profile.

[Change **Clause 13.11**, p. 521]

13.11 GetEnrollmentSummary Service

The specification of this service is retained for historical reference so that implementations of client devices have guidance on how to interoperate with older server devices. Otherwise, it is deprecated. It is recommended that execution of this service not be implemented in server devices.

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.~~

...

[Change **Clause K.2.9**, p. 885]

K.2.9 BIBB - Alarm and Event-Enrollment Summary-B (AE-ESUM-B)

The functionality covered by this BIBB has been deprecated. This BIBB is included solely for historical purposes.

Device B provides event enrollments to device A.
...

[Change **Clause L.7**, p. 911]

L.7 Profiles of the Standard BACnet Devices

The following tables indicate which BIBBs shall be supported by each device type for each interoperability area.

...

Alarm & Event Management

B-AWS	B-OWS	B-OD	B-BC	B-AAC	B-ASC	B-SA	B-SS
AE-N-A	AE-N-A		AE-N-I-B	AE-N-I-B			
AE-ACK-A	AE-ACK-A		AE-ACK-B	AE-ACK-B			
			AE-INFO-B	AE-INFO-B			
			AE-ESUM-B[3]				
AE-AS-A	AE-AS-A						
AE-AVM-A	AE-VM-A						
AE-AVN-A	AE-VN-A	AE-VN-A					
AE-ELVM-A[2]							

...

[1] Not required if the device is a BACnet MS/TP Slave.

[2] Not required for devices claiming conformance to a Protocol_Revision less than 7.

[3] *Not required for devices claiming conformance to Protocol_Revision 13 or greater.*

[Add a new entry to **History of Revisions**, p. 1027]

(This History of Revisions is not part of this standard. It is merely informative and does not contain requirements necessary for conformance to the standard.)

HISTORY OF REVISIONS

…	…	**…**
1	16	**Addendum *av* to ANSI/ASHRAE 135-2012** Approved by the ASHRAE Standards Committee June 28, 2014; by the ASHRAE Board of Directors July 7, 2014; and by the American National Standards Institute July 3, 2014. 1. Deprecate Execution of GetAlarmSummary 2. Deprecate Execution of GetEnrollmentSummary

[This foreword and the "rationales" on the following pages are not part of this standard. They are merely informative and do not contain requirements necessary for conformance to the standard.]

FOREWORD

The purpose of this addendum is to present a proposed change for public review. These modifications are the result of change proposals made pursuant to the ASHRAE continuous maintenance procedures and of deliberations within Standing Standard Project Committee 135. The proposed changes are summarized below.

In the following document, language to be added to existing clauses of ANSI/ASHRAE 135-2012 and Addenda is indicated through the use of *italics*, while deletions are indicated by ~~strikethrough~~. Where entirely new subclauses are proposed to be added, plain type is used throughout. Only this new and deleted text is open to comment at this time. All other material in this addendum is provided for context only and is not open for public review comment except as it relates to the proposed changes.

135-2012*aw*-1 Extend the CHANGE-OF-STATE Event Algorithm for All Discrete Types

Rationale

The discrete data types possible to monitor by the CHANGE_OF_STATE event algorithm are defined through the data type BACnetPropertyStates.

BACnetPropertyStates is extended to include data type INTEGER to enable CHANGE_OF_STATE to monitor values of data type INTEGER.

[Change **BACnetPropertyStates** production, **Clause 21**, p. 708]

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.

...

lighting-transition [40] BACnetLightingTransition,
signed-value *[41] INTEGER,*
...
}
-- 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.

135-2012*aw*-2 Add a New Event Algorithm CHANGE_OF_DISCRETE_VALUE

Rationale

The event-algorithm CHANGE_OF_VALUE is restricted to data types REAL and BIT STRING. It is not possible to use the Event Enrollment object to generate CHANGE_OF_VALUE event notifications for discrete type values, such as the changes of the Present_Value of a Multi-state Output object.

A new event-algorithm CHANGE_OF_DISCRETE_VALUE is added to support 'change of value events' for the discrete primitive data types BOOLEAN, Unsigned, INTEGER, ENUMERATED, CharacterString, OCTET STRING, Date, Time and BACnetObjectIdentifier, and the complex type BACnetDateTime.

[Change **Table 12-15,** p. 210]

12.12.7 Event_Parameters

...

Table 12-15. Event Algorithm, Event Parameters and Event Algorithm Parameters

Event Algorithm	Event Parameters	Event Algorithm Parameters
NONE	none	none
...	...	...
CHANGE_OF_DISCRETE_VALUE	*Time_Delay*	*pTimeDelay*

[Change **Table 13-7**, p. 475]

13.3 Event Algorithms

Table 13-7 lists the event algorithms that are specified in this standard. The event algorithms are indicated by the BACnetEventType value of the same name.

Table 13-7. Standardized Event Algorithms

Event Algorithm	Clause
...	...
UNSIGNED_RANGE	13.3.9
CHANGE_OF_DISCRETE_VALUE	*13.3.X*

[Add new **Clause 13.3.X**, p. 503]

13.3.X CHANGE_OF_DISCRETE_VALUE Event Algorithm

The CHANGE_OF_DISCRETE_VALUE event algorithm, for monitored discrete values of datatype BOOLEAN, Unsigned, INTEGER, ENUMERATED, CharacterString, OCTET STRING, Date, Time, BACnetObjectIdentifier and BACnetDateTime, detects changes to the monitored value.

For detection of change, the value of the monitored value when a transition to NORMAL is indicated shall be used in evaluation of the conditions until the next transition to NORMAL is indicated. The initialization of the value used in evaluation before the first transition to NORMAL is indicated is a local matter.

The parameters of this event algorithm are:

pCurrentState	This parameter, of type BACnetEventState, represents the current value of the Event_State property of the object that applies the event algorithm.
pMonitoredValue	This parameter, of type BOOLEAN, Unsigned, INTEGER, ENUMERATED, CharacterString, OCTET STRING, Date, Time, BACnetObjectIdentifier or BACnetDateTime, represents the current value of the monitored property.
pStatusFlags	This parameter, of type BACnetStatusFlags, represents the current value of the Status_Flags property of the object containing the property that provides the value of the pMonitoredValue parameter. If no value is available for this parameter, then it takes on the value {FALSE, FALSE, FALSE, FALSE}.
pTimeDelay	This parameter, of type Unsigned, represents the time, in seconds, that the offnormal conditions must exist before an offnormal event state is indicated.
pTimeDelayNormal	This parameter, of type Unsigned, represents the time, in seconds, that the Normal conditions must exist before a NORMAL event state is indicated. If no value is available for this parameter, then it takes on the value of the pTimeDelay parameter.

The conditions evaluated by this event algorithm, for a monitored value of type BOOLEAN, Unsigned, INTEGER, ENUMERATED, CharacterString, OCTET STRING, Date, Time, BACnetObjectIdentifier or BACnetDateTime, are:

(a) If pCurrentState is NORMAL, and the value of pMonitoredValue changes and remains changed for pTimeDelayNormal, then indicate a transition to the NORMAL event state.

Figure 13-Z depicts those transitions of Figure 13-X3 that this event algorithm may indicate:

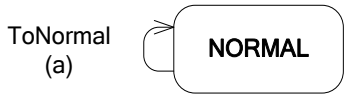


Figure 13-Z. Transitions indicated by CHANGE_OF_DISCRETE_VALUE algorithm

The notification parameters of this algorithm are:

New_Value	This notification parameter, of type BOOLEAN, Unsigned, INTEGER, ENUMERATED, CharacterString, OCTET STRING, Date, Time, BACnetObjectIdentifier or BACnetDateTime, conveys the value of pMonitoredValue.
Status_Flags	This notification parameter, of type BACnetStatusFlags, conveys the value of pStatusFlags.

[Change **Clause 21, BACnetEventParameter** production, p. 680]

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.

...
    none   [20] NULL,

```
        change-of-discrete-value   [21] SEQUENCE {
                                        time-delay              [0] Unsigned
                                        }
        }
...
```

[Change **Clause 21, BACnetEventType** production, p. 683]

```
    BACnetEventType ::= ENUMERATED {
        ...
        none                            (20),
        change-of-discrete-value        (21),
        ...
        }
```

[Change **Clause 21**, **BACnetNotificationParameters** production, p. 688]

```
    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-status-flags [18] SEQUENCE {
                                    present-value           [0] ABSTRACT-SYNTAX.&Type OPTIONAL,
                                                            -- depends on referenced property
                                    referenced-flags        [1] BACnetStatusFlags
                                    } },
                                -- context tag [20] is not used, see note below
        change-of-discrete-value    [21] SEQUENCE {
                                    new-value               [0] CHOICE {
                                                                boolean         BOOLEAN,
                                                                unsigned        Unsigned,
                                                                signed          INTEGER,
                                                                enumerated      ENUMERATED,
                                                                characterString CharacterString,
                                                                octetString     OCTET STRING,
                                                                date            Date,
                                                                time            Time,
                                                                objectid        BACnetObjectIdentifier,
                                                                datetime        [0] BACnetDateTime
                                                                },
                                    status-flags            [1] BACnetStatusFlags
                                    }
        }
```

135-2012*aw*-3 Add a New Fault Algorithm FAULT_OUT_OF_RANGE

Rationale

The standard does not specify a standard algorithm for reporting an out of range fault condition at an object capable of originating events that may report OVER_RANGE or UNDER_RANGE reliability.

A new fault algorithm FAULT_OUT_OF_RANGE is added. The Accumulator, Analog Input, Analog Value, Large Analog Value, Integer Value and Positive Integer Value object types are enabled to use this new fault algorithm, the Event Enrollment object type is extended to allow supporting it.

[Change **Clause 12.1, Accumulator Object Type**, p. 148]

12.1 Accumulator Object Type

...

Accumulator objects that support intrinsic reporting shall apply the UNSIGNED_RANGE event algorithm. *For reliability-evaluation, the FAULT_OUT_OF_RANGE fault algorithm may be applied.*

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
...	...	...
Fault_High_Limit	*Unsigned*	*O*[6]
Fault_Low_Limit	*Unsigned*	*O*[6]
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.
[5] This property, if present, is required to be read only.
[6] *These properties are required if the object supports the FAULT_OUT_OF_RANGE fault algorithm.*

[Change **Clause 12.1.9, Accumulator Object Type**, p. 151]

12.1.9 Reliability

...

If a fault algorithm is applied, then this property shall be the pCurrentReliability parameter for the object's fault algorithm.

[Change **Clause 12.1.20, Accumulator Object Type**, p. 154]

12.1.20 Pulse_Rate

...

If the object supports the FAULT_OUT_OF_RANGE fault algorithm, then this property shall be present and shall be the pMonitoredValue for the FAULT_OUT_OF_RANGE fault algorithm.

[Add new **Clauses 12.1.x1 and 12.1.x2, Accumulator Object Type**, p. 156]

12.1.x1 Fault_High_Limit

This property, of type Unsigned, shall specify a limit that the Pulse_Rate must exceed before a fault event is generated by the FAULT_OUT_OF_RANGE fault algorithm.

If the object supports the FAULT_OUT_OF_RANGE fault algorithm, then this property shall be present and shall be the pMaximumNormalValue for the FAULT_OUT_OF_RANGE fault algorithm.

12.1.x2 Fault_Low_Limit

This property, of type Unsigned, shall specify a limit that the Pulse_Rate must fall below before a fault event is generated by the FAULT_OUT_OF_RANGE fault algorithm.

If the object supports the FAULT_OUT_OF_RANGE fault algorithm, then this property shall be present and shall be the pMinimumNormalValue for the FAULT_OUT_OF_RANGE fault algorithm.

[Change **Clause 12.2, Analog Input Object Type**, p. 157]

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.

Analog Input objects that support intrinsic reporting shall apply the OUT_OF_RANGE event algorithm. *For reliability-evaluation, the FAULT_OUT_OF_RANGE fault algorithm may be applied.*

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
...	...	...
Fault_High_Limit	*REAL*	*O*[5]
Fault_Low_Limit	*REAL*	*O*[5]
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.
[4] This property, if present, is required to be read only.
[5] *These properties are required if the object supports the FAULT_OUT_OF_RANGE fault algorithm.*

[Change **Clause 12.2.4, Analog Input Object Type**, p. 158]

12.2.4 Present_Value

…

If the object supports the FAULT_OUT_OF_RANGE fault algorithm, then this property shall be the pMonitoredValue for the FAULT_OUT_OF_RANGE fault algorithm.

[Change **Clause 12.2.9, Analog Input Object Type**, p. 159]

12.2.9 Reliability

...

If a fault algorithm is applied, then this property shall be the pCurrentReliability parameter for the object's fault algorithm.

[Add new **Clauses 12.2.x1 and 12.2.x2, Analog Input Object Type**, p. 161]

12.2.x1 Fault_High_Limit

This property, of type REAL, shall specify a limit that the Present_Value must exceed before a fault event is generated by the FAULT_OUT_OF_RANGE fault algorithm.

If the object supports the FAULT_OUT_OF_RANGE fault algorithm, then this property shall be present and shall be the pMaximumNormalValue for the FAULT_OUT_OF_RANGE fault algorithm.

12.2.x2 Fault_Low_Limit

This property, of type REAL, shall specify a limit that the Present_Value must fall below before a fault event is generated by the FAULT_OUT_OF_RANGE fault algorithm.

If the object supports the FAULT_OUT_OF_RANGE fault algorithm, then this property shall be present and shall be the pMinimumNormalValue for the FAULT_OUT_OF_RANGE fault algorithm.

[Change **Clause 12.4, Analog Value Object Type**, p. 167]

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.

Analog Value objects that support intrinsic reporting shall apply the OUT_OF_RANGE event algorithm. *For reliability-evaluation, the FAULT_OUT_OF_RANGE fault algorithm may be applied.*

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
...	...	...
Fault_High_Limit	*REAL*	*O*[9]
Fault_Low_Limit	*REAL*	*O*[9]
Profile_Name	CharacterString	O

[1] These properties are required if, and shall be present only if, Present_Value is commandable.
[2] This property is required if, and shall be present only 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.
[5] This property, if present, is required to be read-only.

[6] These properties shall be present only if the object supports intrinsic reporting.
[7] Event_Algorithm_Inhibit shall be present if Event_Algorithm_Inhibit_Ref is present.
[8] If this property is present, then the Reliability property shall be present.
[9] *These properties are required if the object supports the FAULT_OUT_OF_RANGE fault algorithm.*

[Change **Clause 12.4.4, Analog Value Object Type**, p. 168]

12.4.4 Present_Value

…

If the object supports the FAULT_OUT_OF_RANGE fault algorithm, then this property shall be the pMonitoredValue for the FAULT_OUT_OF_RANGE fault algorithm.

[Change **Clause 12.4.8, Analog Value Object Type**, p. 169]

12.4.8 Reliability

...

If a fault algorithm is applied, then this property shall be the pCurrentReliability parameter for the object's fault algorithm.

[Add new **Clauses 12.4.x1 and 12.4.x2, Analog Value Object Type**, p. 171]

12.4.x1 Fault_High_Limit

This property, of type REAL, shall specify a limit that the Present_Value must exceed before a fault event is generated by the FAULT_OUT_OF_RANGE fault algorithm.

If the object supports the FAULT_OUT_OF_RANGE fault algorithm, then this property shall be present and shall be the pMaximumNormalValue for the FAULT_OUT_OF_RANGE fault algorithm.

12.4.x2 Fault_Low_Limit

This property, of type REAL, shall specify a limit that the Present_Value must fall below before a fault event is generated by the FAULT_OUT_OF_RANGE fault algorithm.

If the object supports the FAULT_OUT_OF_RANGE fault algorithm, then this property shall be present and shall be the pMinimumNormalValue for the FAULT_OUT_OF_RANGE fault algorithm.

[Change **Clause 12.39, Large Analog Value Object Type**, p. 381]

12.39 Large Analog Value Object Type

The Large Analog Value object type defines a standardized object whose properties represent the externally visible characteristics of a named data value in a BACnet device. A BACnet device can use a Large Analog Value object to make any kind of double-precision data value accessible to other BACnet devices. The mechanisms by which the value is derived are not visible to the BACnet client.

Large Analog Value objects that support intrinsic reporting shall apply the DOUBLE_OUT_OF_RANGE event algorithm. *For reliability-evaluation, the FAULT_OUT_OF_RANGE fault algorithm may be applied.*

Table 12-46. Properties of the Large Analog Value Object

Property Identifier	Property Datatype	Conformance Code
...	...	...
Fault_High_Limit	*Double*	*O*[9]
Fault_Low_Limit	*Double*	*O*[9]
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] These properties are required if, and shall be present only if, Present_Value is commandable.
[3] This property is required if, and shall be present only if, the object supports COV reporting.
[4] These properties are required if the object supports intrinsic reporting.
[5] This property, if present, is required to be read-only.
[6] These properties shall be present only if the object supports intrinsic reporting.
[7] Event_Algorithm_Inhibit shall be present if Event_Algorithm_Inhibit_Ref is present.
[8] If this property is present, then the Reliability property shall be present.
[9] *These properties are required if the object supports the FAULT_OUT_OF_RANGE fault algorithm.*

[Change **Clause 12.39.5, Large Analog Value Object Type**, p. 382]

12.39.5 Present_Value

…

If the object supports the FAULT_OUT_OF_RANGE fault algorithm, then this property shall be the pMonitoredValue for the FAULT_OUT_OF_RANGE fault algorithm.

[Change **Clause 12.39.8, Large Analog Value Object Type**, p. 383]

12.39.8 Reliability

...

If a fault algorithm is applied, then this property shall be the pCurrentReliability parameter for the object's fault algorithm.

[Add new **Clauses 12.39.x1 and 12.39.x2, Large Analog Value Object Type**, p. 385]

12.39.x1 Fault_High_Limit

This property, of type Double, shall specify a limit that the Present_Value must exceed before a fault event is generated by the FAULT_OUT_OF_RANGE fault algorithm.

If the object supports the FAULT_OUT_OF_RANGE fault algorithm, then this property shall be present and shall be the pMaximumNormalValue for the FAULT_OUT_OF_RANGE fault algorithm.

12.39.x2 Fault_Low_Limit

This property, of type Double, shall specify a limit that the Present_Value must fall below before a fault event is generated by the FAULT_OUT_OF_RANGE fault algorithm.

If the object supports the FAULT_OUT_OF_RANGE fault algorithm, then this property shall be present and shall be the pMinimumNormalValue for the FAULT_OUT_OF_RANGE fault algorithm.

[Change **Clause 12.43, Integer Value Object Type**, p. 397]

12.43 Integer Value Object Type

The Integer Value object type defines a standardized object whose properties represent the externally visible characteristics of a named data value in a BACnet device. A BACnet device can use an Integer Value object to make any kind of signed integer data value accessible to other BACnet devices. The mechanisms by which the value is derived are not visible to the BACnet client.

Integer Value objects that support intrinsic reporting shall apply the SIGNED_OUT_OF_RANGE event algorithm. *For reliability-evaluation, the FAULT_OUT_OF_RANGE fault algorithm may be applied.*

Table 12-50. Properties of the Integer Value Object

Property Identifier	Property Datatype	Conformance Code
...	...	...
Fault_High_Limit	*INTEGER*	*O*[9]
Fault_Low_Limit	*INTEGER*	*O*[9]
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] These properties are required if, and shall be present only if, Present_Value is commandable.
[3] This property is required if, and shall be present only if, the object supports COV reporting.
[4] These properties are required if the object supports intrinsic reporting.
[5] This property, if present, is required to be read-only.
[6] These properties shall be present only if the object supports intrinsic reporting.
[7] Event_Algorithm_Inhibit shall be present if Event_Algorithm_Inhibit_Ref is present.
[8] If this property is present, then the Reliability property shall be present.
[9] *These properties are required if the object supports the FAULT_OUT_OF_RANGE fault algorithm.*

[Change **Clause 12.43.5, Integer Value Object Type**, p. 398]

12.43.5 Present_Value

…

If the object supports the FAULT_OUT_OF_RANGE fault algorithm, then this property shall be the pMonitoredValue for the FAULT_OUT_OF_RANGE fault algorithm.

[Change **Clause 12.43.8, Integer Value Object Type**, p. 399]

12.43.8 Reliability

...

If a fault algorithm is applied, then this property shall be the pCurrentReliability parameter for the object's fault algorithm.

[Add new **Clauses 12.43.x1 and 12.43.x2, Integer Value Object Type**, p. 401]

12.43.x1 Fault_High_Limit

This property, of type INTEGER, shall specify a limit that the Present_Value must exceed before a fault event is generated by the FAULT_OUT_OF_RANGE fault algorithm.

If the object supports the FAULT_OUT_OF_RANGE fault algorithm, then this property shall be present and shall be the pMaximumNormalValue for the FAULT_OUT_OF_RANGE fault algorithm.

12.43.x2 Fault_Low_Limit

This property, of type INTEGER, shall specify a limit that the Present_Value must fall below before a fault event is generated by the FAULT_OUT_OF_RANGE fault algorithm.

If the object supports the FAULT_OUT_OF_RANGE fault algorithm, then this property shall be present and shall be the pMinimumNormalValue for the FAULT_OUT_OF_RANGE fault algorithm.

[Change **Clause 12.44, Positive Integer Value Object Type**, p. 402]

12.44 Positive Integer Value Object Type

The Positive Integer Value object type defines a standardized object whose properties represent the externally visible characteristics of a named data value in a BACnet device. A BACnet device can use a Positive Integer Value object to make any kind of unsigned data value accessible to other BACnet devices. The mechanisms by which the value is derived are not visible to the BACnet client.

Positive Integer Value objects that support intrinsic reporting shall apply the UNSIGNED_OUT_OF_RANGE event algorithm. *For reliability-evaluation, the FAULT_OUT_OF_RANGE fault algorithm may be applied.*

Table 12-51. Properties of the Positive Integer Value Object

Property Identifier	Property Datatype	Conformance Code
…	…	…
Fault_High_Limit	*Unsigned*	*O*[9]
Fault_Low_Limit	*Unsigned*	*O*[9]
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] These properties are required if, and shall be present only if, Present_Value is commandable.
[3] This property is required if, and shall be present only if, the object supports COV reporting.
[4] These properties are required if the object supports intrinsic reporting.
[5] This property, if present, is required to be read-only.
[6] These properties shall be present only if the object supports intrinsic reporting.
[7] Event_Algorithm_Inhibit shall be present if Event_Algorithm_Inhibit_Ref is present.
[8] If this property is present, then the Reliability property shall be present.
[9] *These properties are required if the object supports the FAULT_OUT_OF_RANGE fault algorithm.*

[Change **Clause 12.44.5, Positive Integer Value Object Type**, p. 403]

12.44.5 Present_Value

…

If the object supports the FAULT_OUT_OF_RANGE fault algorithm, then this property shall be the pMonitoredValue for the FAULT_OUT_OF_RANGE fault algorithm.

[Change **Clause 12.44.8, Positive Integer Value Object Type**, p. 404]

12.44.8 Reliability

...

If a fault algorithm is applied, then this property shall be the pCurrentReliability parameter for the object's fault algorithm.

[Add new **Clauses 12.44.x1 and 12.44.x2, Positive Integer Value Object Type**, p. 406]

12.44.x1 Fault_High_Limit

This property, of type Unsigned, shall specify a limit that the Present_Value must exceed before a fault event is generated by the FAULT_OUT_OF_RANGE fault algorithm.

If the object supports the FAULT_OUT_OF_RANGE fault algorithm, then this property shall be present and shall be the pMaximumNormalValue for the FAULT_OUT_OF_RANGE fault algorithm.

12.44.x2 Fault_Low_Limit

This property, of type Unsigned, shall specify a limit that the Present_Value must fall below before a fault event is generated by the FAULT_OUT_OF_RANGE fault algorithm.

If the object supports the FAULT_OUT_OF_RANGE fault algorithm, then this property shall be present and shall be the pMinimumNormalValue for the FAULT_OUT_OF_RANGE fault algorithm.

[Change **Clause 21**, p. 695]

BACnetPropertyIdentifier ::= ENUMERATED { -- see below for numerical order

```
...
failed-attempts-time            (274),
fault-high-limit                (388),
fault-low-limit                 (389),
fault-parameters                (358),
...
-- see egress-active            (386),
-- see fault-high-limit         (388),
-- see fault-low-limit          (389),
...
}
```

[Change **Clause 12.12.8, Event Enrollment Object Type**, p. 211]

12.12.8 Object_Property_Reference

...

Depending on the fault algorithm, the values of additional properties of the monitored object are used for particular fault algorithm parameters, as specified by Table 12-15.2.

Table 12-15.2. Additional Monitored Object Properties by Fault Algorithm

Fault Algorithm	Additional Monitored Object Properties	Fault Algorithm Parameters
NONE	none	none
FAULT_CHARACTERSTRING	none	none
FAULT_EXTENDED	Defined by vendor	Defined by vendor
FAULT_LIFE_SAFETY	none	none
FAULT_STATE	none	none
FAULT_STATUS_FLAGS	none	none
FAULT_OUT_OF_RANGE	*none*	*none*

[Change **Clause 12.12.23, Event Enrollment Object Type**, p. 214]

12.12.23 Fault_Parameters

This property, of type BACnetFaultParameter, determines the fault algorithm used to monitor the referenced object and provides the parameter values needed for this fault algorithm. The mapping to the fault algorithm parameter values is defined in Table 12-15.3.

If the Event Enrollment object does not apply a fault algorithm, then the fault parameter choice NONE shall be set in this property.

Table 12-15.3. Fault Algorithm, Fault Parameters and Fault Algorithm Parameters

Fault Algorithm	Fault Parameters	Fault Algorithm Parameters
NONE	none	none
FAULT_CHARACTERSTRING	List_Of_Fault_Values	pFaultValues
FAULT_EXTENDED	Vendor_Id Extended_Fault_Type Parameters	pVendorId pFaultType pParameters
FAULT_LIFE_SAFETY	List_Of_Fault_Values Mode_Property_Reference	pFaultValues Referent's value is pMode
FAULT_STATE	List_Of_Fault_Values	pFaultValues
FAULT_STATUS_FLAGS	Status_Flags_Property_Reference	pMonitoredValue
FAULT_OUT_OF_RANGE	*Min_Normal_Value* *Max_Normal_Value*	*pMinimumNormalValue* *pMaximumNormalValue*

[Change **Clause 21**, **BACnetFaultParameter** production, p. 683]

```
BACnetFaultParameter ::= CHOICE {
    none                [0] NULL,

    …

    fault-status-flags  [5] SEQUENCE {
                            status-flags-reference      [0]  BACnetDeviceObjectPropertyReference
                            },
    fault-out-of-range  [6] SEQUENCE {
                            min-normal-value            [0]  CHOICE {
```

```
                                                    REAL,
                                                    Unsigned,
                                                    Double,
                                                    INTEGER
                                                    },
                        max-normal-value        [1] CHOICE {
                                                    REAL,
                                                    Unsigned,
                                                    Double,
                                                    INTEGER
                                                    },
                        }
    }
```

[Change **Clause 21**, **BACnetFaultType** production, p. 684]

```
BACnetFaultType ::= ENUMERATED {
    none                                (0),
    fault-characterstring               (1),
    fault-extended                      (2),
    fault-life-safety                   (3),
    fault-state                         (4),
    fault-status-flags                  (5),
    fault-out-of-range                  (6)
    }
```

[Change **Table 13-8**, p. 504]

Table 13-8. Standardized Fault Algorithms

Fault Algorithm	Clause
NONE	13.4.1
FAULT_CHARACTERSTRING	13.4.2
FAULT_EXTENDED	13.4.3
FAULT_LIFE_SAFETY	13.4.4
FAULT_STATE	13.4.5
FAULT_STATUS_FLAGS	13.4.6
FAULT_OUT_OF_RANGE	*13.4.X*

[Add new **Clause 13.4.X**, p 508]

13.4.X FAULT_OUT_OF_RANGE Fault Algorithm

The FAULT_OUT_OF_RANGE fault algorithm detects whether the monitored value is outside the range of values considered to be normal for the object. At the vendor's discretion, evaluation of the algorithm may be delayed by verification or filtering mechanisms local to the object that applies the fault algorithm which are used to insure that the pMonitoredValue is correct.

The parameters of this fault algorithm are:

pMinimumNormalValue	This parameter, of type REAL, Unsigned, Double or INTEGER, represents the minimum normal value.

pMaximumNormalValue	This parameter, of type REAL, Unsigned, Double or INTEGER, represents the maximum normal value.
pMonitoredValue	This parameter, of type REAL, Unsigned, Double or INTEGER, is the value monitored by this algorithm.
pCurrentReliability	This parameter, of type BACnetReliability, represents the current value of the Reliability property of the object that applies the fault algorithm.

For any single instance of the FAULT_OUT_OF_RANGE fault algorithm, all of the numeric parameters, pMinimumNormalValue, pMaximumNormalValue, and pMonitoredValue shall have the same data type and shall match the data type of the object properties assigned to the parameters.

The conditions evaluated by this fault algorithm are:

(a) If pCurrentReliability is NO_FAULT_DETECTED, and pMonitoredValue is less than pMinimumNormalValue, then indicate a transition to the UNDER_RANGE reliability.

(b) If pCurrentReliability is NO_FAULT_DETECTED, and pMonitoredValue is greater than pMaximumNormalValue, then indicate a transition to the OVER_RANGE reliability.

(c) If pCurrentReliability is UNDER_RANGE, and pMonitoredValue is greater than pMaximumNormalValue, then indicate a transition to the OVER_RANGE reliability.

(d) If pCurrentReliability is OVER_RANGE, and pMonitoredValue is less than pMinimumNormalValue, then indicate a transition to the UNDER_RANGE reliability.

(e) If pCurrentReliability is UNDER_RANGE, and pMonitoredValue is greater than or equal to pMinimumNormalValue, and pMonitoredValue is less than or equal to pMaximumNormalValue, then indicate a transition to the NO_FAULT_DETECTED reliability.

(f) If pCurrentReliability is OVER_RANGE, and pMonitoredValue is greater than or equal to pMinimumNormalValue, and pMonitoredValue is less than or equal to pMaximumNormalValue, then indicate a transition to the NO_FAULT_DETECTED reliability.

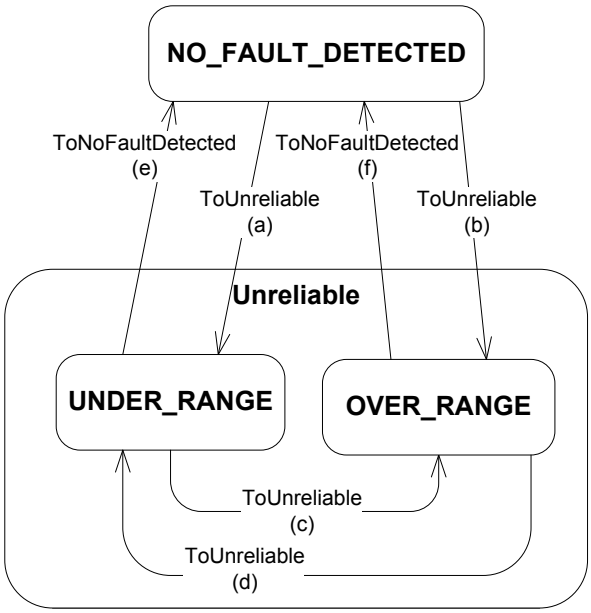


Figure 13-X. Transitions indicated by the FAULT_OUT_OF_RANGE algorithm

135-2012*aw*-4 Extend the Loop Object Type to Support Specific Low and High Error Limits

Rationale

Extend the Loop object type to support dedicated low and high error limits for its FLOATING_LIMIT event algorithm.

[Change **Clause 12.17, Loop Object Type**, p. 234]

12.17 Loop Object Type

...

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
...	...	...
Event_Message_Texts	BACnetARRAY[3] of CharacterString	O[6]
Low_Diff_Limit	*BACnetOptionalREAL*	*O*
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.
[6] This property, if present, is required to be read only.

[Change **Clause 12.17.32**, p. 239]

12.17.32 Error_Limit

This property is both the pLowDiffLimit and pHighDiffLimit parameter for the object's event algorithm *if the property Low_Diff_Limit is not present or has a value of NULL*.

If the property Low_Diff_Limit is present and has a value other than NULL, then Error_Limit is the pHighDiffLimit parameter for the object's event algorithm.

See 13.3 for event algorithm parameter descriptions.

[Add new **Clause 12.17.x**, p. 240]

12.17.x Low_Diff_Limit

This property, of type BACnetOptionalREAL, is the pLowDiffLimit parameter for the object's FLOATING_LIMIT event algorithm if it has a value other than NULL.

If this property has the value NULL, then the value of the Error_Limit property is used as the pLowDiffLimit parameter for the object's FLOATING_LIMIT event algorithm.

See 13.3 for event algorithm parameter descriptions.

[Add new production **BACnetOptionalREAL to Clause 21**, p. 694]
[Note that a BACnetOptionalREAL production is added identically in Addendum 135-2012*at*]

BACnetOptionalREAL ::= CHOICE {
    null   NULL,
    real   REAL
    }

[Change Clause 21, BACnetPropertyIdentifier production, p. 695]

BACnetPropertyIdentifier ::= ENUMERATED { -- see below for numerical order
...
    logging-type   (197),
    low-diff-limit   *(390),*
    low-limit   (59),
...
    -- see egress-active   (386),
    -- *see low-diff-limit*   *(390),*
    ...
    }

135-2012*aw*-5 Add the Ability to Report Faults to Date and Time Related Value Objects

Rationale

Several date and time related value object types are able to detect and indicate faults, through properties Status_Flags, Event_State, and Reliability. These object types are currently not specified to be able to support intrinsic reporting of faults.

This change adds the ability of intrinsically reporting faults to the primitive value object types Time Value, Date Value, DateTime Value, Time Pattern Value, Date Pattern Value and DateTime Pattern Value.

[Change **Clauses 12.38 DateTime Value Object Type, 12.42 Time Value Object Type, 12.45 Date Value Object Type, 12.46 DateTime Pattern Value Object Type, 12.47 Time Pattern Value Object Type** and **12.48 Date Pattern Object Type.** The respective object type name is replaced by "<TYPE>" below]

…

<TYPE> objects may optionally support intrinsic reporting to facilitate the reporting of fault conditions. <TYPE> objects that support intrinsic reporting shall apply the NONE event algorithm.

The <TYPE> object type and its standardized properties are summarized in Table 12-X and described in detail in this subclause.

Table 12-X. Properties of the <TYPE> Object Type

. . .	. . .	. . .
Event_Detection_Enable	*BOOLEAN*	*O*[x,y]
Notification_Class	*Unsigned*	*O*[x,y]
Event_Enable	*BACnetEventTransitionBits*	*O*[x,y]
Acked_Transitions	*BACnetEventTransitionBits*	*O*[x,y]
Notify_Type	*BACnetNotifyType*	*O*[x,y]
Event_Time_Stamps	*BACnetARRAY[3] of BACnetTimeStamp*	*O*[x,y]
Event_Message_Texts	*BACnetARRAY[3] of CharacterString*	*O*[y]
Event_Message_Texts_Config	*BACnetARRAY[3] of CharacterString*	*O*[y]
Profile_Name	CharacterString	O

…

[x] *These properties are required if the object supports intrinsic reporting.*
[y] *These properties shall be present only if the object supports intrinsic reporting.*

[Add new Clauses **12.38.z1..., 12.42.z1..., 12.45.z1..., 12.46.z1..., 12.47.z1...,** and **12.48.z1....** The respective second level sub-clause number is replaced by "X" below.]

12.X.z1 Event_Detection_Enable
This property, of type BOOLEAN, indicates whether (TRUE) or not (FALSE) intrinsic reporting is enabled in the object and controls whether (TRUE) or not (FALSE) the object will be considered by event summarization services.

This property is expected to be set during system configuration and is not expected to change dynamically.

When this property is FALSE, Event_State shall be NORMAL, and the properties Acked_Transitions, Event_Time_Stamps, and Event_Message_Texts shall be equal to their respective initial conditions.

12.X.z2 Notification_Class

This property, of type Unsigned, shall specify the instance of the Notification Class object to use for event-notification-distribution.

12.X.z3 Event_Enable
This property, of type BACnetEventTransitionBits, shall convey three flags that separately enable and disable the distribution of TO_OFFNORMAL, TO_FAULT, and TO_NORMAL notifications (see Clause 13.2.5). A device is allowed to restrict the set of supported values for this property but shall support (T, T, T) at a minimum.

12.X.z4 Acked_Transitions
This read-only property, of type BACnetEventTransitionBits, shall convey three flags that separately indicate the acknowledgment state for TO_OFFNORMAL, TO_FAULT, and TO_NORMAL events (see Clause 13.2.2.1.5). Each flag shall have the value TRUE if no event of that type has ever occurred for the object.

12.X.z5 Notify_Type
This property, of type BACnetNotifyType, shall convey whether the notifications generated by the object should be Events or Alarms. The value of the property is used as the value of the 'Notify Type' service parameter in event notifications generated by the object.

12.X.z6 Event_Time_Stamps
This read-only property, of type BACnetARRAY[3] of BACnetTimeStamp, shall convey the times of the last TO_OFFNORMAL, TO_FAULT, and TO_NORMAL events (see 13.2.2.1). 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 of that type has ever occurred for the object.

12.X.z7 Event_Message_Texts
This read-only property, of type BACnetARRAY[3] of CharacterString, shall convey the message text values of the last TO_OFFNORMAL, TO_FAULT, and TO_NORMAL events, respectively (see Clause 13.2.2.1). If a particular type of event has yet to occur, an empty string shall be stored in the respective array element.

12.X.z8 Event_Message_Texts_Config
This property, of type BACnetARRAY[3] of CharacterString, contains the character strings which are the basis for the 'Message Text' parameter for the event notifications of TO_OFFNORMAL, TO_FAULT, and TO_NORMAL events, respectively, generated by this object. The character strings may optionally contain proprietary text substitution codes to incorporate dynamic information such as date and time or other information.

135-2012*aw*-6 Add the Ability to Report Faults to the Command, Device and Notification Class Objects

Rationale

The Command, Device and Notification Class object types are currently not able to indicate and notify fault conditions.

This change adds the ability of detecting, indicating and intrinsically reporting faults to the Command, Device and Notification Class object types.

[Change Clauses **12.10 Command Object Type, Clause 12.11 Device Object Type** and **12.21 Notification Class Object Type.** The respective object type is replaced by "<TYPE>" below]

…

<TYPE> objects may optionally support intrinsic reporting to facilitate the reporting of fault conditions. <TYPE> objects that support intrinsic reporting shall apply the NONE event algorithm.

The <TYPE> object type and its standardized properties are summarized in Table 12-X and described in detail in this subclause.

Table 12-X. Properties of the <TYPE> Object Type

. . .	. . .	. . .
Status_Flags	*BACnetStatusFlags*	O^{x}
Event_State	*BACnetEventState*	O^{x}
Reliability	*BACnetReliability*	O^{x}
Event_Detection_Enable	*BOOLEAN*	$O^{x,y}$
Notification_Class	*Unsigned*	$O^{x,y}$
Event_Enable	*BACnetEventTransitionBits*	$O^{x,y}$
Acked_Transitions	*BACnetEventTransitionBits*	$O^{x,y}$
Notify_Type	*BACnetNotifyType*	$O^{x,y}$
Event_Time_Stamps	*BACnetARRAY[3] of BACnetTimeStamp*	$O^{x,y}$
Event_Message_Texts	*BACnetARRAY[3] of CharacterString*	O^{y}
Event_Message_Texts_Config	*BACnetARRAY[3] of CharacterString*	O^{y}
Reliability_Evaluation_Inhibit	*BOOLEAN*	O^{z}
Profile_Name	CharacterString	O

…

[x] *These properties are required if the object supports intrinsic reporting.*
[y] *These properties shall be present only if the object supports intrinsic reporting.*
[z] *If this property is present, then the Reliability property shall be present.*

[Add new **Clauses 12.10.y1 to 12.10.y3 to the Command Object Type**, p. 197]

12.10.y1 Status_Flags
This property, of type BACnetStatusFlags, represents four Boolean flags that indicate the general "health" of the Command object. One flag is 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 TRUE (1) if the Event_State property is present and does not have a value of NORMAL, otherwise 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	The value of this flag shall be logical FALSE (0).
OUT_OF_SERVICE	The value of this flag shall be logical FALSE (0).

12.10.y2 Event_State
The Event_State property, of type BACnetEventState, is included in order to provide a way to determine whether this object has an active event state associated with it (see Clause 13.2.2.1). If the object supports event reporting, then the Event_State property shall indicate the event state of the object. If the object does not support event reporting then the value of this property shall be NORMAL.

12.10.y3 Reliability
The Reliability property, of type BACnetReliability, provides an indication of whether the Command object is properly configured and is able to process write requests received. Failing property write attempts initiated by this object shall not cause this property to indicate a fault condition. Such error conditions shall be indicated in respective other properties.

[Add new **Clauses 12.11.y1 to 12.11.y3 to the Device Object Type**, p. 206]

12.11.y1 Status_Flags
This property, of type BACnetStatusFlags, represents four Boolean flags that indicate the general "health" of the Device object. One flag is 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 TRUE (1) if the Event_State property is present and does not have a value of NORMAL, otherwise 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	The value of this flag shall be logical FALSE (0).
OUT_OF_SERVICE	The value of this flag shall be logical FALSE (0).

12.11.y2 Event_State
The Event_State property, of type BACnetEventState, is included in order to provide a way to determine whether this object has an active event state associated with it (see Clause 13.2.2.1). If the object supports event reporting, then the Event_State property shall indicate the event state of the object. If the object does not support event reporting then the value of this property shall be NORMAL.

12.11.y3 Reliability
The Reliability property, of type BACnetReliability, provides an indication of whether the Device object is properly configured, is able to perform the functionality it represents and to maintain properties that indicate status information.

[Add new **Clauses 12.21.y1 to 12.21.y3 to the Notification Class Object Type**, p. 259]

12.21.y1 Status_Flags
This property, of type BACnetStatusFlags, represents four Boolean flags that indicate the general "health" of the Notification Class object. One flag is 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 TRUE (1) if the Event_State property is present and does not have a value of NORMAL, otherwise 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	The value of this flag shall be logical FALSE (0).
OUT_OF_SERVICE	The value of this flag shall be logical FALSE (0).

12.21.y2 Event_State
The Event_State property, of type BACnetEventState, is included in order to provide a way to determine whether this object has an active event state associated with it (see Clause 13.2.2.1). If the object supports event reporting, then the Event_State property shall indicate the event state of the object. If the object does not support event reporting then the value of this property shall be NORMAL.

12.21.y3 Reliability
The Reliability property, of type BACnetReliability, provides an indication of whether the Notification Class object is properly configured and is able to perform the functionality it represents. Failing event notifications initiated by this object shall not cause this property to indicate a fault condition.

[Add new **Clauses 12.10.z1..., 12.11.z1...,** and **12.21.z1....** The respective second level clause number is replaced by X below.]

12.X.z1 Event_Detection_Enable
This property, of type BOOLEAN, indicates whether (TRUE) or not (FALSE) intrinsic reporting is enabled in the object and controls whether (TRUE) or not (FALSE) the object will be considered by event summarization services.

This property is expected to be set during system configuration and is not expected to change dynamically.

When this property is FALSE, Event_State shall be NORMAL, and the properties Acked_Transitions, Event_Time_Stamps, and Event_Message_Texts shall be equal to their respective initial conditions.

12.X.z2 Notification_Class
This property, of type Unsigned, shall specify the instance of the Notification Class object to use for event-notification-distribution.

12.X.z3 Event_Enable
This property, of type BACnetEventTransitionBits, shall convey three flags that separately enable and disable the distribution of TO_OFFNORMAL, TO_TO_FAULT, and TO_NORMAL notifications (see Clause 13.2.5).

A device is allowed to restrict the set of supported values for this property but shall support (T, T, T) at a minimum.

12.X.z4 Acked_Transitions
This read-only property, of type BACnetEventTransitionBits, shall convey three flags that separately indicate the acknowledgment state for TO_OFFNORMAL, TO_FAULT, and TO_NORMAL events (see Clause 13.2.2.1.5). Each flag shall have the value TRUE if no event of that type has ever occurred for the object.

12.X.z5 Notify_Type
This property, of type BACnetNotifyType, shall convey whether the notifications generated by the object should be Events or Alarms. The value of the property is used as the value of the 'Notify Type' service parameter in event notifications generated by the object.

12.X.z6 Event_Time_Stamps
This read-only property, of type BACnetARRAY[3] of BACnetTimeStamp, shall convey the times of the last TO_OFFNORMAL, TO_FAULT, and TO_NORMAL events (see 13.2.2.1). 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 of that type has ever occurred for the object.

12.X.z7 Event_Message_Texts
This read-only property, of type BACnetARRAY[3] of CharacterString, shall convey the message text values of the last TO_OFFNORMAL, TO_FAULT, and TO_NORMAL events, respectively (see Clause 13.2.2.1). If a particular type of event has yet to occur, an empty string shall be stored in the respective array element.

12.X.z8 Event_Message_Texts_Config
This property, of type BACnetARRAY[3] of CharacterString, contains the character strings which are the basis for the 'Message Text' parameter for the event notifications of TO_OFFNORMAL, TO_FAULT, and TO_NORMAL events, respectively, generated by this object. The character strings may optionally contain proprietary text substitution codes to incorporate dynamic information such as date and time or other information.

12.X.z9 Reliability_Evaluation_Inhibit
This property, of type BOOLEAN, indicates whether (TRUE) or not (FALSE) reliability-evaluation is disabled in the object. This property is a runtime override that allows temporary disabling of reliability-evaluation.

When reliability-evaluation is disabled, the Reliability property shall have the value NO_FAULT_DETECTED unless Out_Of_Service is TRUE and an alternate value has been written to the Reliability property.

[Add a new entry to **History of Revisions**, p. 1027]

(This History of Revisions is not part of this standard. It is merely informative and does not contain requirements necessary for conformance to the standard.)

HISTORY OF REVISIONS

…	…	**…**
1	16	**Addendum *aw* to ANSI/ASHRAE 135-2012** Approved by the ASHRAE Standards Committee June 28, 2014; by the ASHRAE Board of Directors July 7, 2014; and by the American National Standards Institute July 3, 2014. 1. Extend the CHANGE-OF-STATE Event Algorithm for All Discrete Types 2. Add a New Event Algorithm CHANGE_OF_DISCRETE_VALUE 3. Add a New Fault Algorithm FAULT_OUT_OF_RANGE 4. Extend the Loop Object Type to Support Specific Low and High Error Limits 5. Add the Ability to Report Faults to Date and Time Related Value Objects 6. Add the Ability to Report Faults to the Command, Device and Notification Class Objects

[This foreword and the "rationales" on the following pages are not part of this standard. They are merely informative and do not contain requirements necessary for conformance to the standard.]

FOREWORD

The purpose of this addendum is to present a proposed change for public review. These modifications are the result of change proposals made pursuant to the ASHRAE continuous maintenance procedures and of deliberations within Standing Standard Project Committee 135. The proposed changes are summarized below.

In the following document, language to be added to existing clauses of ANSI/ASHRAE 135-2012 and Addenda is indicated through the use of *italics*, while deletions are indicated by ~~strikethrough~~. Where entirely new subclauses are proposed to be added, plain type is used throughout. Only this new and deleted text is open to comment at this time. All other material in this addendum is provided for context only and is not open for public review comment except as it relates to the proposed changes.

135-2012*ax*-1 Remove Incorrect Recipient_List Requirement to be Non-empty

Rationale

Make the language on number of entries of the Recipient_List property of the Notification Class object consistent with the language of the Recipient_List property of the Notification Forwarder object.

[Change **Clause 12.21.8,** p. 258]

12.21.8 Recipient_List

This property, of type BACnetList of BACnetDestination, shall convey a list of ~~one~~ *zero* or more recipient destinations to which notifications shall be sent when event-initiating objects using this class detect the occurrence of an event. These recipient destinations are intended to be relatively permanent, do not expire, and shall be maintained through a power failure or device "restart." The destinations themselves define a structure of parameters that is summarized in Table 12-25.

[Change **Clause 12.51.8,** p. 433]

12.51.8 Recipient_List

This property, of type BACnetLIST of BACnetDestination, shall convey a list of *zero or more* recipient destinations to which notifications shall be sent when events are processed by the Notification Forwarder object. These recipient destinations are intended to be relatively permanent, do not expire and shall be maintained through a power failure or device "restart". If not writable, the Recipient_List must be configurable by some other means. The destinations themselves define a structure of parameters that is summarized in Table 12-25.

135-2012*ax*-2 Section Removed

135-2012*ax*-3 Extend the Allowable BACnetPropertyStates Enumeration Range

Rationale

The space reserved in BACnetPropertyStates for standard enumerations is quickly diminishing. This change provides a mechanism that will allow more enumerations to be added while not breaking existing decoders or violating the space reserved for vendor extensions.

[Change BACnetPropertyStates production in **Clause 21**, p. 707]

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,
...	
backup-state	[36] BACnetBackupState,
extended-value	*[63] Unsigned32,*
-- example-one	*[256] BACnetExampleTypeOne,*
-- example-two	*[257]BACnetExampleTypeTwo,*
...	
}	

-- Tag values greater than 254 are not encoded as ASN context tags. In these cases, the tag value is multiplied
-- by 100000 and is added to the enumeration value and the sum is encoded using context tag 63, the extended-value
-- choice.
-- Tag values 0-63 *and 256-42949* 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.

135-2012*ax*-4 Specifically Disallow Duplicate Time Entries in Schedules

Rationale

The standard does not discuss how to deal with duplicate time entries in lists of Time/Value pairs in schedules. Rather than provide support for an obscure use case, this change explicitly disallows duplicate time entries.

[Change **Clause 12.24.7**, p. 275]

12.24.7 Weekly_Schedule

...

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. *If the Weekly_Schedule property is written with a value containing a duplicate time within any single day, the device shall return a Result(-) response specifying an 'Error Class' of PROPERTY and an 'Error Code' of DUPLICATE_ENTRY.*

[Change **Clause 12.24.8**, p. 275]

12.24.8 Exception_Schedule

...

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. *If the Exception_Schedule property is written with a BACnetSpecialEvent containing a duplicate time in any particular SEQUENCE OF TimeValue, the device shall return a Result(-) response specifying an 'Error Class' of PROPERTY and an 'Error Code' of DUPLICATE_ENTRY.*

[Insert into **Clause 18.3**, p. 596]

18.3 Error Class - PROPERTY

...

Duplicate_Entry
An attempt has been made to write to a SEQUENCE OF, with a value that has duplicate entries in a property that does not accept duplicates, such as duplicate times within a single SEQUENCE OF TimeValue within a BACnetSpecialEvent or in a BACnetDailySchedule array element.

...

[Change Error production in **Clause 21**, p. 655]

Error ::= SEQUENCE {
    ...
    error-code ENUMERATED {
        ...
        duplicate-object-id (49),
        duplicate-entry (137),
        dynamic-creation-not-supported (4),
        ...
        -- see abort-security-error (136),
        -- see *duplicate-entry* *(137),*

...

135-2012*ax*-5 Non-BBMD Responses to BBMD BVLL Requests

Rationale

The standard does not specifically state what a non-BBMD should do when it receives a BBMD-related BVLL request.

[Change **Clause J.2.1**, p. 617]

J.2.1 BVLC-Result: Purpose

This message provides a mechanism to acknowledge the result of those BVLL service requests that require an acknowledgment, whether successful (ACK) or unsuccessful (NAK). These are: Write-Broadcast-Distribution-Table (ACK, NAK); Read-Broadcast-Distribution-Table (NAK only); Register-Foreign-Device (ACK, NAK); Read-Foreign-Device-Table (NAK only); Delete-Foreign-Device-Table-Entry (ACK, NAK); and Distribute-Broadcast-To-Network (NAK only).

All non-BBMD B/IP devices shall return the appropriate NAK upon receipt of a unicast BVLL service request intended for BBMDs only (Write-Broadcast-Distribution-Table; Read-Broadcast-Distribution-Table; Register-Foreign-Device; Read-Foreign-Device-Table; Delete-Foreign-Device-Table-Entry; and Distribute-Broadcast-To-Network).

[Add a new entry to **History of Revisions**, p. 1027]

(This History of Revisions is not part of this standard. It is merely informative and does not contain requirements necessary for conformance to the standard.)

HISTORY OF REVISIONS

…	…	**…**
1	16	**Addendum *ax* to ANSI/ASHRAE 135-2012** Approved by the ASHRAE Standards Committee June 28, 2014; by the ASHRAE Board of Directors July 7, 2014; and by the American National Standards Institute July 3, 2014. 1. Remove Incorrect Recipient_List Requirement to be Non-empty 2. Section Removed 3. Extend the Allowable BACnetPropertyStates Enumeration Range 4. Specifically Disallow Duplicate Time Entries in Schedules 5. Non-BBMD Responses to BBMD BVLL Requests

[This foreword and the "rationales" on the following pages are not part of this standard. They are merely informative and do not contain requirements necessary for conformance to the standard.]

FOREWORD

The purpose of this addendum is to present a proposed change for public review. These modifications are the result of change proposals made pursuant to the ASHRAE continuous maintenance procedures and of deliberations within Standing Standard Project Committee 135. The proposed changes are summarized below.

135-2012*az*-1 Add Binary Lighting Output Object Type, p. 3
135-2012*az*-2 Setting Non-zero Values to Change_Of_State_Count and Elapsed_Active_Time, p. 14

In the following document, language to be added to existing clauses of ANSI/ASHRAE 135-2012 and Addenda is indicated through the use of *italics*, while deletions are indicated by ~~strikethrough~~. Where entirely new subclauses are proposed to be added, plain type is used throughout. Only this new and deleted text is open to comment at this time. All other material in this addendum is provided for context only and is not open for public review comment except as it relates to the proposed changes.

135-2012*az*-1 Add Binary Lighting Output Object Type

Rationale

The Lighting Output object as defined in BACnet 135-2012 Clause 12.54 defines a BACnet lighting output that is analog in nature. It has a present value defined as a REAL with a range of 0%...100% and includes special lighting functions which are typical in analog lighting such as ramping, fading, and stepping.

It has been suggested that the Lighting Output object can also be used for binary lighting applications by utilizing a threshold where values over the threshold are considered to be 'on' and values under the threshold are considered to be 'off'. While this could be made to work, it suffers from a number of complexities:

1. If the Present_Value is between 0%...100%, it is not possible to determine if the binary lighting output is on or off by simply reading the Present_Value, as the binary threshold is not defined in a standard manner.

2. The binary threshold is not exposed as a standard BACnet property. This may make it difficult for a third-party client to ever determine if the physical lighting is on or off.

3. The Lighting Output object defines special functions for ramping, fading, and stepping that must be supported by all instances of the object. It is not clear how a physical binary lighting output would be affected by these analog lighting commands.

Clearly the BACnet Lighting Output object is not ideal for controlling a binary lighting output. The solution to these issues is to define a lighting output specific for binary lighting that does not have the complexities of the analog solution.

The object type proposed in this document defines a BACnet Binary Lighting Output object type. It is based on the (analog) Lighting Output object and incorporates the same properties and functionality where possible. The object is significantly less complex than the analog version but retains a similar feel.

[Change **Clause 12**, p. 146]

12. MODELING CONTROL DEVICES AS A COLLECTION OF OBJECTS

...

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.

…

TRIPPED	The end device, such as an actuator, is not responding to commands, prevented by a tripped condition or by being mechanically held open.
LAMP_FAILURE	*Indicates that the lamp has failed in a physical lighting device.*

[Add new **Clause 12.X**, p. 459]

12.X Binary Lighting Output Object Type

The Binary Lighting Output object type defines a standardized object whose properties represent the externally visible characteristics of a binary lighting output and includes dedicated functionality specific to lighting control that would otherwise require explicit programming.

This object is binary in nature and can be either in a logical on or off state. The object is commanded by writing to the Present_Value property. This property is commandable and uses a priority array to arbitrate between multiple writers to the binary lighting output.

The binary lighting output also provides a blink-warn mechanism to notify room occupants that the lights are about to turn off. The blink-warning mechanism is internal to the object and may cause the physical lights to blink on and off or issue a notification in some other manner. A blink-warn command is issued by writing a special value to the Present_Value. The blink-warn command comes in three different variants. The WARN command causes a blink-warn notification but then leaves the level of the lights unaffected. The WARN_RELINQUISH command executes the blink-warn notification and then keeps the lights ON for a predetermined amount of time (egress time) and then relinquishes the value at the given priority. The WARN_OFF command executes the blink-warn notification and then keeps the lights ON and then writes OFF at the given priority.

The following example illustrates how a Binary Lighting Output object may be used in a typical office scenario. Prior to 7:00 am the lights are off as the Lighting Output object is being controlled at the relinquish default value (OFF). At 7:00 am a scheduler (e.g., a BACnet Schedule object or other automated process) turns the physical lights on by writing ON to the Present_Value property at priority 9. At 6:00 pm a WARN_RELINQUISH lighting command is executed at priority 9, which causes an immediate blink-warn notification to occur but leaves the lights on until the egress timer has expired. Assuming a 10 minute (600 seconds) egress time is specified, the value at priority 9 is relinquished at 6:10 pm. This scenario is shown in figure 12-X1.

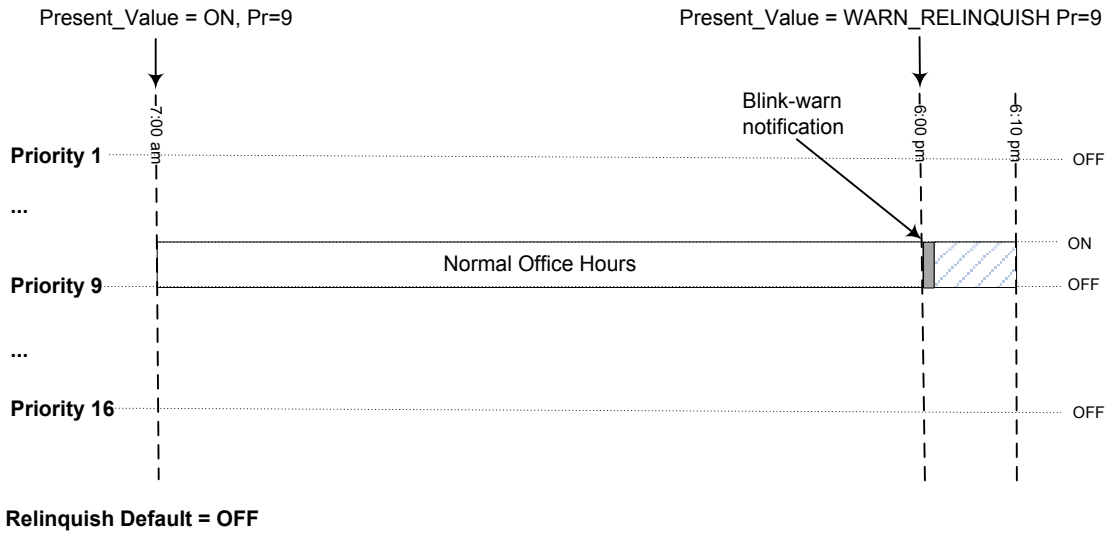


Figure 12-X1. Daily Schedule with Blink-Warn Example

The object and its properties are summarized in Table 12-X1 and described in detail in this subclause.

Table 12-X1. Properties of the Binary Lighting Output Object

Property Identifier	Property Datatype	Conformance Code
Object_Identifier	BACnetObjectIdentifier	R
Object_Name	CharacterString	R
Object_Type	BACnetObjectType	R
Present_Value	BACnetBinaryLightingPV	W
Description	CharacterString	O
Status_Flags	BACnetStatusFlags	R
Reliability	BACnetReliability	O
Out_Of_Service	BOOLEAN	R
Blink_Warn_Enable	BOOLEAN	R
Egress_Time	Unsigned	R
Egress_Active	BOOLEAN	R
Feedback_Value	BACnetBinaryLightingPV	O
Priority_Array	BACnetPriorityArray	R
Relinquish_Default	BACnetBinaryLightingPV	R
Power	REAL	O
Polarity	BACnetPolarity	O
Elapsed_Active_Time	Unsigned32	O[1]
Time_Of_Active_Time_Reset	BACnetDateTime	O[1]
Strike_Count	Unsigned	O[2]
Time_Of_Strike_Count_Reset	BACnetDateTime	O[2]
Event_Detection_Enable	BOOLEAN	O[3,4]
Notification_Class	Unsigned	O[3,4]
Event_Enable	BACnetEventTransitionBits	O[3,4]
Acked_Transitions	BACnetEventTransitionBits	O[3,4]
Notify_Type	BACnetNotifyType	O[3,4]
Event_Time_Stamps	BACnetARRAY[3] of BACnetTimeStamp	O[3,4]
Event_Message_Texts	BACnetARRAY[3] of CharacterString	O[4]
Event_Message_Texts_Config	BACnetARRAY[3] of CharacterString	O[4]
Reliability_Evaluation_Inhibit	BOOLEAN	O[5]
Property_List	BACnetARRAY[N] of BACnetPropertyIdentifier	R
Profile_Name	CharacterString	O

[1] 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.
[2] If one of the optional properties Strike_Count or Time_Of_Strike_Count_Reset is present, then both of these properties shall be present.
[3] These properties are required if the object supports intrinsic reporting.
[4] These properties shall be present only if the object supports intrinsic reporting.
[5] If this property is present, then the Reliability property shall be present.

12.X.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.X.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.X.3 Object_Type

This property, of type BACnetObjectType, indicates membership in a particular object type class. The value of this property shall be BINARY_LIGHTING_OUTPUT.

12.X.4 Present_Value (Commandable)

This property, of type BACnetBinaryLightingPV, reflects the logical state of the Binary Lighting Output. The logical state of the output shall be either ON or OFF.

The Present_Value also supports special values to provide blink-warn functionality. Each special value written corresponds to a specific operation which has an effect on the present value; however, the special value itself is not written to the priority array. The special values of the Present_Value are summarized in table 12-X2.

Table 12-X2. Special Values of the Present_Value Property

Value	Description
WARN	Executes a blink-warn notification at the specified priority. After the blink-warn notification has been executed the value at the specified priority remains ON. The blink-warn notification shall not occur if any of the following conditions occur: (a) The specified priority is not the highest active priority, or (b) The value at the specified priority is OFF, or (c) Blink_Warn_Enable is FALSE. (see Clause 12.X.9.1 Blink-Warn Behavior)
WARN_OFF	Executes a blink-warn notification at the specified priority and then writes the value OFF to the specified slot in the priority array after a delay of Egress_Time seconds. The blink-warn notification shall not occur and the value OFF written at the specified priority immediately if any of the following conditions occur: (a) The specified priority is not the highest active priority, or (b) The Present_Value is OFF, or (c) Blink_Warn_Enable is FALSE. (see Clause 12.X.9.1 Blink-Warn Behavior).
WARN_RELINQUISH	Executes a blink-warn notification at the specified priority and then relinquishes the value at the specified priority slot after a delay of Egress_Time seconds. The blink-warn notification shall not occur, and the value at the specified priority shall be relinquished immediately if any of the following conditions occur: (a) The specified priority is not the highest active priority, or (b) The value at the specified priority is OFF or NULL, or (c) The value of the next highest non-NULL priority, including Relinquish_Default, is ON, or (d) Blink_Warn_Enable is FALSE.

	(See Clause 12.X.9.1 Blink-Warn Behavior).
STOP	Cancels any WARN_RELINQUISH or WARN_OFF operation in progress at the specified priority and cancels the blink-warn egress timer. The value in the priority array at the specified priority remains unchanged. If there is no warn operation currently executing at the specified priority, then this value is ignored.

The physical lighting output shall be updated whenever the Present_Value is commanded. However, when the device starts up or is reset, it is a local matter as to whether the physical lighting output is updated with the current value of Present_Value or whether the value of the physical output before startup or reset is retained.

12.X.4.1 Halting Warn Operation in Progress

The WARN_OFF and WARN_RELINQUISH operations are executed over a period of time. While a lighting operation, at a specific priority, is in progress it shall be halted when the Present_Value is written at a higher priority than the operation in progress.

When a WARN_RELINQUISH operation currently in progress is halted, the blink-warn egress timer is immediately expired and the corresponding value of the priority array is relinquished.

When a WARN_OFF operation currently in progress is halted, the egress timer is immediately expired and the value OFF is written to the priority array at the specified priority.

A WARN_OFF or WARN_RELINQUISH operation which is in progress is implicitly halted when it is overwritten by a write to the Present_Value at the same priority.

12.X.5 Description

This property, of type CharacterString, is a string of printable characters whose content is not restricted.

12.X.6 Status_Flags

This property, of type BACnetStatusFlags, represents four Boolean flags that indicate the general "health" of a Binary Lighting Output 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 — Always 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 output has been overridden by some mechanism local to the BACnet Device, otherwise logical FALSE (0). 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.

OUT_OF_SERVICE — Logical TRUE (1) if the Out_Of_Service property has a value of TRUE, otherwise logical FALSE (0).

12.X.7 Reliability

This 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.

12.X.8 Out_Of_Service

This property, of type BOOLEAN, indicates 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 lighting 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 lighting 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 lighting 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.X.9 Blink_Warn_Enable

This property, of type BOOLEAN, specifies whether a blink-warn is executed (TRUE) or not (FALSE) when a WARN, WARN_RELINQUISH, or WARN_OFF value is written to Present_Value. When this property is FALSE and a warn operation is written, a blink-warn notification shall not occur, and the effect of the operation shall occur immediately without an egress delay.

12.X.9.1 Blink-Warn Behavior

The WARN, WARN_RELINQUISH, and WARN_OFF special values to the Present_Value property, cause a blink-warn notification to occur at the specified priority. A blink-warn notification is used to warn the occupants that the lights are about to turn off, giving the occupants the opportunity to exit the space or to override the lights for a period of time.

The actual blink-warn notification mechanism shall be a local matter. The physical lights may blink once, multiple times, or repeatedly. They may also go bright, go dim, or signal a notification through some other means. In some circumstances, no blink-warn notification will occur at all. The blink-warn notification shall not be reflected in the priority array.

The WARN_RELINQUISH and WARN_OFF operations include an egress time in which the lights are held ON until the egress time expires or the operation is halted. The number of seconds for egress is specified in the Egress_Time property. The egress timer shall start when the WARN_RELINQUISH or WARN_OFF operation is written. While the egress timer is active, the property Egress_Active shall be set to TRUE. When the egress timer expires or the operation is halted, then Egress_Active shall be set to FALSE. There may only be one priority slot with an active egress timer at any time.

If the Blink_Warn_Enable property has the value FALSE, then the blink-warn notification shall not occur, and the effect of the operation shall occur immediately without an egress delay.

The relationship between Egress_Time and the Egress_Active property is shown in Figure 12-X2.

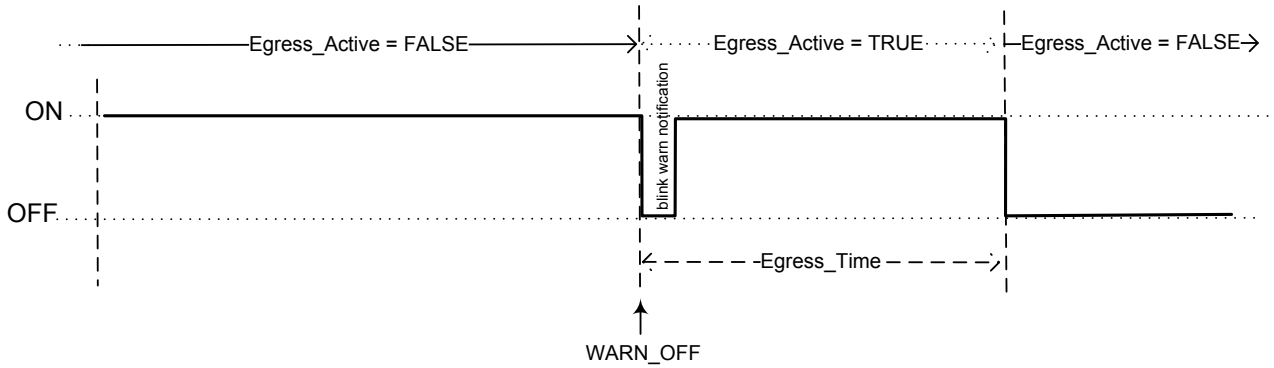


Figure 12-X2. Blink-warn and Egress Time

12.X.10 Egress_Time

This property, of type Unsigned, specifies the egress time in seconds when a WARN_RELINQUISH or WARN_OFF value is written to the Present_Value property. The egress time is the time in which the light level is held at its current level before turning off.

12.X.11 Egress_Active

This property, of type BOOLEAN, shall be TRUE whenever the Egress_Time for a WARN_RELINQUISH or WARN_OFF lighting operation is in effect and FALSE otherwise.

12.X.12 Feedback_Value

This property, of type BACnetBinaryLightingPV, shall indicate the actual value of the physical lighting output and shall have the value ON (i.e., light is physically on) or OFF (i.e., light is physically off).

If this property is writable an attempt to write a value other than ON or OFF shall be denied and a Result(-) specifying an 'Error Class' of PROPERTY and an 'Error Code' of VALUE_OUT_OF_RANGE shall be returned.

The manner by which the Feedback_Value is determined shall be a local matter.

12.X.13 Priority_Array

This property is a read-only array of prioritized values. See Clause 19 for a description of the prioritization mechanism.

12.X.14 Relinquish_Default

This property, of type BACnetBinaryLightingPV, 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. This property shall have the value ON or OFF.

If this property is writable an attempt to write a value other than ON or OFF shall be denied and a Result(-) specifying an 'Error Class' of PROPERTY and an 'Error Code' of VALUE_OUT_OF_RANGE shall be returned.

See Clause 19.

12.X.15 Power

This property, of type REAL, is the nominal power consumption of the load(s) controlled by this object when the physical lighting output is on. The units shall be kilowatts.

12.X.16 Polarity

This property, of type BACnetPolarity, indicates the relationship between the physical state of the lighting output and the logical state represented by the Present_Value property. If the Polarity property is NORMAL, then the ON state of the Present_Value property is also the ON state of the physical output as long as Out_Of_Service is FALSE. If the Polarity property is REVERSE, then the ON state of the Present_Value property is the OFF state of the physical output as long as Out_Of_Service is FALSE.

If Out_Of_Service is TRUE, then the Polarity property shall have no effect on the physical output state.

12.X.17 Elapsed_Active_Time

This property, of type Unsigned32, represents the accumulated number of seconds that the Present_Value property or the Feedback_Value Property has had the value ON since the date and time indicated by the Time_Of_Active_Time_Reset property. Whether Present_Value or Feedback_Value is used as the indicator for the calculation of the Elapsed_Active_Time is a local matter.

When this property is set to zero, the Time_Of_Active_Time_Reset property shall be set to the current date and time. When this property is set to a non-zero value, the value of the Time_Of_Active_Time_Reset property shall not change. If this property is writable, it shall be a local matter whether the property accepts writes of zero only.

12.X.18 Time_Of_Active_Time_Reset

This property, of type BACnetDateTime, indicates the date and time at which the accumulation of active time, indicated by the Elapsed_Active_Time property, has been started.

If the Elapsed_Active_Time property accepts writes of non-zero values, then the Time_Of_Active_Time_Reset property shall be writeable, otherwise it shall be read-only.

12.X.19 Strike_Count

This property, of type Unsigned, represents the number of times that the Present_Value property or Feedback_Value property has transitioned from OFF to ON since the date and time indicated by the Time_Of_Strike_Count_Reset property. Whether Present_Value or Feedback_Value is used as the indicator for the calculation of the Strike_Count is a local matter.

When this property is set to zero, the Time_Of_Strike_Count_Reset property shall be set to the current date and time. When this property is set to a non-zero value, the value of the Time_Of_Strike_Count_Reset property shall not change. If this property is writable, it shall be a local matter whether the property accepts writes of zero only.

The Strike_Count property shall have a range of 0-65535 or greater.

12.X.20 Time_Of_Strike_Count_Reset

This property, of type BACnetDateTime, indicates the date and time at which the counting of lamp strikes, indicated by the Strike_Count property, has been started.

If the Strike_Count property accepts writes of non-zero values, then the Time_Of_Strike_Count_Reset property shall be writeable, otherwise it shall be read-only.

12.X.21 Event_Detection_Enable

This property, of type BOOLEAN, indicates whether (TRUE) or not (FALSE) intrinsic reporting is enabled in the object and controls whether (TRUE) or not (FALSE) the object will be considered by event summarization services.

This property is expected to be set during system configuration and is not expected to change dynamically.

When this property is FALSE, Event_State shall be NORMAL, and the properties Acked_Transitions, Event_Time_Stamps, and Event_Message_Texts shall be equal to their respective initial conditions.

12.X.22 Notification_Class

This property, of type Unsigned, shall specify the instance of the Notification Class object to use for event-notification-distribution.

12.X.23 Event_Enable

This property, of type BACnetEventTransitionBits, shall convey three flags that separately enable and disable the distribution of TO_OFFNORMAL, TO_FAULT, and TO_NORMAL notifications (see Clause 13.2.5). A device is allowed to restrict the set of supported values for this property but shall support (T, T, T) at a minimum.

12.X.24 Acked_Transitions

This read-only property, of type BACnetEventTransitionBits, shall convey three flags that separately indicate the acknowledgment state for TO_OFFNORMAL, TO_FAULT, and TO_NORMAL events (see Clause 13.2.2.1.5). Each flag shall have the value TRUE if no event of that type has ever occurred for the object.

12.X.25 Notify_Type

This property, of type BACnetNotifyType, shall convey whether the notifications generated by the object should be Events or Alarms. The value of the property is used as the value of the 'Notify Type' service parameter in event notifications generated by the object.

12.X.26 Event_Time_Stamps

This read-only property, of type BACnetARRAY[3] of BACnetTimeStamp, shall convey the times of the last TO_OFFNORMAL, TO_FAULT, and TO_NORMAL events (see Clause 13.2.2.1). Timestamps of type Time or Date shall have X'FF' in each octet and Sequence Number timestamps shall have the value 0 if no event of that type has ever occurred for the object.

12.X.27 Event_Message_Texts

This read-only property, of type BACnetARRAY[3] of CharacterString, shall convey the message text values of the last TO_OFFNORMAL, TO_FAULT, and TO_NORMAL events (see Clause 13.2.2.1). If a particular type of event has yet to occur, an empty string shall be stored in the respective array element.

12.X.28 Event_Message_Texts_Config

This property, of type BACnetARRAY[3] of CharacterString, contains the character strings which are the basis for the 'Message Text' parameter for the event notifications of TO_OFFNORMAL, TO_FAULT, and TO_NORMAL events, respectively, generated by this object. The character strings may optionally contain proprietary text substitution codes to incorporate dynamic information such as date and time or other information.

12.X.29 Reliability_Evaluation_Inhibit

This property, of type BOOLEAN, indicates whether (TRUE) or not (FALSE) reliability-evaluation is disabled in the object. This property is a runtime override that allows temporary disabling of reliability-evaluation.

When reliability-evaluation is disabled, the Reliability property shall have the value NO_FAULT_DETECTED unless Out_Of_Service is TRUE and an alternate value has been written to the Reliability property.

12.X.30 Property_List

This read-only property is a BACnetARRAY of property identifiers, one property identifier for each property that exists within the object. The Object_Name, Object_Type, Object_Identifier, and Property_List properties are not included in the list.

12.X.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.

[Change **Table 13-1**, p. 462]

Table 13-1. Standardized Objects That May Support COV Reporting

Object Type	Criteria	Properties Reported
....	...	..
Binary Input, Binary Output, Binary Value, Life Safety Point, Life Safety Zone, Multi-state Input, Multi-state Output, Multi-state Value, OctetString Value, CharacterString Value, Time Value, DateTime Value, Date Value, Time Pattern Value, Date Pattern Value, DateTime Pattern Value, *Binary Lighting Output*	If Present_Value changes at all or Status_Flags changes at all	Present_Value, Status_Flags
...		

[Change **Clause 19.2.1.1**, p. 610]

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
Access Door	Present_Value
BitString Value	Present_Value
CharacterString Value	Present_Value
Date Value	Present_Value
Date Pattern Value	Present_Value
DateTime Value	Present_Value
DateTime Pattern Value	Present_Value
Large Analog Value	Present_Value
OctetString Value	Present_Value
Integer Value	Present_Value
Time Value	Present_Value
Time Pattern Value	Present_Value
Positive Integer Value	Present_Value
Channel	Present_Value (see 19.2.1.6)
Lighting Output	Present_Value
Binary Lighting Output	*Present_Value*

The designated properties of the Analog Output, Binary Output, Multi-state Output, Access Door, ~~and~~ Lighting Output~~,~~ *and Binary Lighting Output* objects are commandable (prioritized) by definition. The designated properties of the Analog Value, Binary Value, Multi-state Value, BitString Value, CharacterString Value, Date Value, Date Pattern Value, DateTime Value, DateTime Pattern Value, Large Analog Value, OctetString Value, Integer Value, Time Value, Time Pattern Value, and Positive Integer 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. . The Channel object is a special exception, see Clause 19.2.1.6.

[Add new productions to **Clause 21**, p. 661]

BACnetBinaryLightingPV ::= ENUMERATED {
off (0),
on (1),
warn (2),
warn-off (3),
warn-relinquish (4),
stop (5) ,
... }

-- 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.

[Change BACnetObjectType production in **Clause 21**, p. 690]

BACnetObjectType ::= ENUMERATED { -- see below for numerical order
...
binary-input (3),
binary-lighting-output (n),
...
-- see lighting-output (54),
-- see binary-lighting-output (55),
}

[Change BACnetObjectTypesSupported production in **Clause 21**, p. 692]

BACnetObjectTypesSupported ::= BIT STRING {
…
lighting-output (54),
binary-lighting-output (55)
}

[Change BACnetPropertyStates production **Clause 21**, p. 708]

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,
...
lighting-transition [40] BACnetLightingTransition,
binary-lighting-value [42] BACnetBinaryLightingPV
...

}

[Change BACnetPropertyIdentifer production **Clause 21**, p. 694]

BACnetPropertyIdentifier ::= ENUMERATED { -- see below for numerical order

```
    ...
    structured-object-list                  (209),
    strike-count                            (391),
    subordinate-annotations                 (210),
    ...
    time-of-state-count-reset               (115),
    time-of-strike-count-reset              (392),
    time-synchronization-interval           (204),
    ...
    -- -numerical order reference
    ...
    -- see egress-active                    (386),
    -- see strike-count                     (391),
    -- see time-of-strike-count-reset       (392),
    ...
    }
```

[Change BACnetReliability production **Clause 21**, p. 709]

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),
    -- enumeration value 11 is reserved for a future addendum
    communication-failure       (12),
    member-fault                (13),
    monitored-object-fault      (14),
    tripped                     (15),
    lamp-failure                (16),
    ...
    }
-- 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.
```

[Change **Table 23-1**, p. 718]

Table 23-1. Extensible Enumerations

Enumeration Name	Reserved Range	Maximum Value
…	…	…

Enumeration Name	Reserved Range	Maximum Value
BACnetLightingTransition	0-63	255
BACnetBinaryLightingPV	*0-63*	*255*

135-2012*az*-2 Setting Non-zero Values to Change_Of_State_Count and Elapsed_Active_Time

Rationale

In case when a machine or appliance is replaced by a revised "second hand" product, the values of the properties Change_Of_State_Count and Elapsed_Active_Time should be set to the accumulated active time or state changes of the replacement product, rather than to zero.

In addition, the language of the properties is unified to be the same in all binary objects. The proposed changes remove language which is redundant, or move language to other places. The data type of the Change_Of_State_Count property of the Binary Value object type is changed from Unsigned32 to Unsigned, for consistency.

[Change **Clause 12.6.4,** page 176]

12.6.4 Present_Value

...

The Present_Value property shall be writable when Out_Of_Service is TRUE. *When Out_Of_Service is TRUE, changes of the Polarity property shall not cause changes of Present_Value. When Out_Of_Service is FALSE, a change of the Polarity property shall alter Present_Value accordingly.*

If the object supports event reporting, then this property shall be the pMonitoredValue parameter for the object's event algorithm. See Clause 13.3 for event algorithm parameter descriptions.

[Change **Clauses 12.6.14, 12.6.15, 12.6.16, 12.6.17,** and **12.6.18**, page 178]

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 date and time indicated by the Time_Of_State_Count_Reset property.*

When this property is set to zero, the Time_Of_State_Count_Reset property shall be set to the current date and time. When this property is set to a non-zero value, the value of the Time_Of_State_Count_Reset property shall not change. If this property is writable, it shall be a local matter whether the property accepts writes of zero only.

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~~ *indicates* the date and time at which ~~the Change_Of_State_Count property was most recently set to a zero value~~ *the counting of state changes, indicated by the Change_Of_State_Count property, has been started*. ~~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.~~

If the Change_Of_State_Count property accepts writes of non-zero values, then the Time_Of_State_Count_Reset property shall be writeable, otherwise it shall be read-only.

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~~ *the date and time indicated by the Time_Of_Active_Time_Reset property.*

When this property is set to zero, the Time_Of_Active_Time_Reset property shall be set to the current date and time. When this property is set to a non-zero value, the value of the Time_Of_Active_Time_Reset property shall not change. If this property is writable, it shall be a local matter whether the property accepts writes of zero only.

~~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~~ *indicates* the date and time at which ~~the Elapsed_Active_Time property was most recently set to a zero value~~ *the accumulation of active time, indicated by the Elapsed_Active_Time property, has been started*. ~~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.~~

If the Elapsed_Active_Time property accepts writes of non-zero values, then the Time_Of_Active_Time_Reset property shall be writeable, otherwise it shall be read-only.

[Change **Clauses 12.7.14, 12.7.15, 12.7.16, 12.7.17,** and **12.7.18**, page 184]

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 date and time indicated by the Time_Of_State_Count_Reset property.*

When this property is set to zero, the Time_Of_State_Count_Reset property shall be set to the current date and time. When this property is set to a non-zero value, the value of the Time_Of_State_Count_Reset property shall not change. If this property is writable, it shall be a local matter whether the property accepts writes of zero only.

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~~ *indicates* the date and time at which ~~the Change_Of_State_Count property was most recently set to a zero value~~ *the counting of state changes, indicated by the Change_Of_State_Count property, has been started*. ~~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.~~

If the Change_Of_State_Count property accepts writes of non-zero values, then the Time_Of_State_Count_Reset property shall be writeable, otherwise it shall be read-only.

12.7.17 Elapsed_Active_Time

This property, of type Unsigned32, represents the accumulated number of seconds that the Present_Value property or the Feedback_Value property has had the value ACTIVE since ~~this property was most recently set to a zero value~~ *the date and time indicated by the Time_Of_Active_Time_Reset property.* Whether Present_Value or Feedback_Value is used as the indicator for the calculation of the Elapsed_Active_Time is a local matter.

When this property is set to zero, the Time_Of_Active_Time_Reset property shall be set to the current date and time. When this property is set to a non-zero value, the value of the Time_Of_Active_Time_Reset property shall not change. If this property is writable, it shall be a local matter whether the property accepts writes of zero only.

~~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~~ *indicates* the date and time at which~~ the Elapsed_Active_Time property was most recently set to a zero value~~ *the accumulation of active time, indicated by the Elapsed_Active_Time property, has been started*. ~~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.~~

If the Elapsed_Active_Time property accepts writes of non-zero values, then the Time_Of_Active_Time_Reset property shall be writeable, otherwise it shall be read-only.

[Change **Table 12-10**, page 187]

12.8 Binary Value Object Type

...

Table 12-10. Properties of the Binary Value Object Type

Property Identifier	Property Datatype	Conformance Code
...		
Change_Of_State_Time	BACnetDateTime	O[3]
Change_Of_State_Count	~~Unsigned32~~ *Unsigned*	O[3]
Time_Of_State_Count_Reset	BACnetDateTime	O[3]
...		

...

[Change **Clauses 12.8.12, 12.8.13, 12.8.14, 12.8.15, and 12.8.16**, page 189]

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~~

Present_Value property. ~~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~~ *Unsigned*, represents the number of times that the ~~state of the Binary Value~~ *Present_Value property* has changed *state* since ~~this property was most recently set to a zero value~~ *the date and time indicated by the Time_Of_State_Count_Reset property.*

When this property is set to zero, the Time_Of_State_Count_Reset property shall be set to the current date and time. When this property is set to a non-zero value, the value of the Time_Of_State_Count_Reset property shall not change. If this property is writable, it shall be a local matter whether the property accepts writes of zero only.

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~~ *indicates* the date and time at which ~~the Change_Of_State_Count property was most recently set to a zero value~~ *the counting of state changes, indicated by the Change_Of_State_Count property, has been started*. ~~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.~~

If the Change_Of_State_Count property accepts writes of non-zero values, then the Time_Of_State_Count_Reset property shall be writeable, otherwise it shall be read-only.

12.8.15 Elapsed_Active_Time

This property, of type Unsigned32, represents the accumulated number of seconds that the Present_Value property or the Feedback_Value property has had the value ACTIVE since ~~this property was most recently set to a zero value~~ *the date and time indicated by the Time_Of_Active_Time_Reset property.* Whether Present_Value or Feedback_Value is used as the indicator for the calculation of the Elapsed_Active_Time is a local matter.

When this property is set to zero, the Time_Of_Active_Time_Reset property shall be set to the current date and time. When this property is set to a non-zero value, the value of the Time_Of_Active_Time_Reset property shall not change. If this property is writable, it shall be a local matter whether the property accepts writes of zero only.

~~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~~ *indicates* the date and time at which ~~the Elapsed_Active_Time property was most recently set to a zero value~~ *the accumulation of active time, indicated by the Elapsed_Active_Time property, has been started*. ~~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.~~

If the Elapsed_Active_Time property accepts writes of non-zero values, then the Time_Of_Active_Time_Reset property shall be writeable, otherwise it shall be read-only.

[Add a new entry to **History of Revisions**, p. 1027]

(This History of Revisions is not part of this standard. It is merely informative and does not contain requirements necessary for conformance to the standard.)

HISTORY OF REVISIONS

…	…	**…**
1	16	**Addendum *az* to ANSI/ASHRAE 135-2012** Approved by the ASHRAE Standards Committee June 28, 2014; by the ASHRAE Board of Directors July 7, 2014; and by the American National Standards Institute July 3, 2014. 1. Add Binary Lighting Output Object Type 2. Setting Non-zero Values to Change_Of_State_Count and Elapsed_Active_Time

POLICY STATEMENT DEFINING ASHRAE'S CONCERN FOR THE ENVIRONMENTAL IMPACT OF ITS ACTIVITIES

ASHRAE is concerned with the impact of its members' activities on both the indoor and outdoor environment. ASHRAE's members will strive to minimize any possible deleterious effect on the indoor and outdoor environment of the systems and components in their responsibility while maximizing the beneficial effects these systems provide, consistent with accepted standards and the practical state of the art.

ASHRAE's short-range goal is to ensure that the systems and components within its scope do not impact the indoor and outdoor environment to a greater extent than specified by the standards and guidelines as established by itself and other responsible bodies.

As an ongoing goal, ASHRAE will, through its Standards Committee and extensive technical committee structure, continue to generate up-to-date standards and guidelines where appropriate and adopt, recommend, and promote those new and revised standards developed by other responsible organizations.

Through its *Handbook*, appropriate chapters will contain up-to-date standards and design considerations as the material is systematically revised.

ASHRAE will take the lead with respect to dissemination of environmental information of its primary interest and will seek out and disseminate information from other responsible organizations that is pertinent, as guides to updating standards and guidelines.

The effects of the design and selection of equipment and systems will be considered within the scope of the system's intended use and expected misuse. The disposal of hazardous materials, if any, will also be considered.

ASHRAE's primary concern for environmental impact will be at the site where equipment within ASHRAE's scope operates. However, energy source selection and the possible environmental impact due to the energy source and energy transportation will be considered where possible. Recommendations concerning energy source selection should be made by its members.