ANSI/ASHRAE Addendum af to
ANSI/ASHRAE Standard 135-2010

ASHRAE ADDENDA

BACnet®

A Data Communication Protocol for Building Automation and Control Networks

Approved by the ASHRAE Standards Committee on June 25, 2011; by the ASHRAE Board of Directors on June 29, 2011; and by the American National Standards Institute on June 30, 2011.

This addendum was 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 Web site (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 Web site (www.ashrae.org) or from ASHRAE Customer Service, 1791 Tullie Circle, NE, Atlanta, GA 30329-2305. E-mail: orders@ashrae.org. Fax: 404-321-5478. Telephone: 404-636-8400 (worldwide), or toll free 1-800-527-4723 (for orders in US and Canada). For reprint permission, go to www.ashrae.org/permissions.

ISSN 1041-2336

Approved American National Standard
ANSI
www.ansi.org

American Society of Heating, Refrigerating and Air-Conditioning Engineers, Inc.
1791 Tullie Circle NE, Atlanta, GA 30329
www.ashrae.org

ASHRAE Standing Standard Project Committee 135
Cognizant TC: TC 1.4, Control Theory and Application
SPLS Liaison: Richard L. Hall

ASHRAE STANDARDS COMMITTEE 2010–2011

SPECIAL NOTE

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

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

ASHRAE Standards are prepared by a Project Committee appointed specifically for the purpose of writing the Standard. The Project Committee Chair and Vice-Chair must be members of ASHRAE; while other 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.

[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 135-2010 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.

INTRODUCTION

ASHRAE SSPC-135 has addressed a number of issues in the BACnet standard related to events and alarms over the years. Numerous inconsistencies have been identified and clarified through the use of interpretation requests, but have not been comprehensively addressed in the standard. A larger initiative in the form of three three-day "Alarm Summit" meetings and many conference calls over the course of 18 months has led to this Alarm Addendum that proposes a comprehensive resolution of these issues.

This introduction provides some background information on the history and development of the Alarm Addendum. But as its main purpose, the reader will be introduced to the changes proposed and guided in reading the addendum.

CONTENTS

1. Alarm Addendum History

In the recent years, many change proposals and interpretation requests related to alarms and events have been brought before the BACnet committee. These documents were seeking for clarifications, extensions and changes to the event mechanisms of BACnet. Many of the change proposals were stalled at the point of recognizing the significant amount of work required to propose comprehensive modifications to the event mechanism specification of BACnet.

In Germantown in April 2008, an interpretation request was discussed that asked for the possibility for a more reasonable notification of fault events. This was the ultimate trigger for the committee to start an initiative called "Alarm Summit" that was dedicated to address all pending issues and to revise the specification in a comprehensive and meaningful way.

In June 2008 during the Salt Lake City meeting, a first Alarm Summit meeting was agreed upon and scheduled for fall 2008, right before the regular committee and working group interim meetings in Atlanta. After two other Alarm Summit meetings, many hours of Objects & Services Working Group (OS-WG) meetings and many telephone conferences, the OS-WG has prepared this draft of the Alarm Addendum for review by the committee and the public.

2. Initial List of Issues

During the first Alarm Summit meeting a list of issues was compiled. The issues discussed include:

- **Clarification and improvement of fault reporting**
- **Revise the standard to use a more formal language for the alarm and event specification**
- **Freedom to choose the event algorithm used for intrinsic reporting by a particular object type**
- **Support of generic alarm notifications, not bound to an algorithm**
- **Extended capabilities to inhibit the event algorithm**
- **Addition of a standard mechanism to forward event notifications**
- **Relaxing the event algorithms to allow additional transitions**
- **Improvements of enrollment and addition of subscription mechanism for temporary clients**
- **Clarifications of alarm summarization mechanisms**
- **Alarm acknowledgment improvements**
- **Improvement of alarm configuration security**
- **Multiple alarm algorithms per object, for different kinds of events**

- Extensions of event notification messages
- Addition of an alarm escalation mechanism
- Alarm grouping and aggregation
- Additional properties in alarm generating objects for better visibility of alarm conditions
- Alarm classification improvements
- Other minor issues

Not all of these issues made it through the debates, and some were rearranged while others were added. In addition, it turned out that some of the issues can only be resolved reasonably by some more radical approaches. Such approaches were considered risky, time consuming and contentious. It became clear that a prioritization is required. As a result, only those issues that are urgent, are fixing major flaws, or are required to pave the ground for the future have been addressed in this addendum as a first step.

As a guiding principle, changes were crafted carefully to preserve backward compatibility with existing definitions to the greatest extent possible.

3. Alarm Addendum Overview

The Alarm Addendum, as completed for its first public review, contains 32 parts, identified by numbers. In order to address the objects and properties of the 135-2010 version of the standard, and those of addenda that are either published or may be in public review at the same time as this addendum, a more generic language is used to identify the applicable objects and properties.

As an overview, the changes proposed in these parts can be summarized as follows.

Section 1 to Section 15: Paving the Ground for the Future
These changes will improve further maintenance of the standard. Also, they add clarity and consistency to definitions that will become relevant in future changes, in particular for those changes that will be needed to go forward with issues not yet addressed.

Section 1: Remove Annex C and Annex D
Annex C and Annex D are being removed from the standard because they do not provide information that is not already clear from other parts of the standard and have proven to be problematic in respect to maintenance of the standard.

Section 2 to Section 15: Clarifications on Property Presence
Language on presence requirements of properties is made clear. In general, presence requirements are modified so that it is clear under what conditions properties are allowed to be present.

Section 32: General Revision of Alarm and Event Concepts
Although not being the first part of the addendum, the concept and language revision of event reporting is to be found in part 32. This part entirely replaces existing clauses 13.2, 13.3 and 13.4. The changes in functionality are minimal. However, these clauses have been completely rewritten to gain clarity in the definition of event reporting. The former content has been rearranged and some parts were reassigned to more relevant clauses of the standard. This addendum part is also the key to better understanding the more particular changes made to objects and properties related to event reporting.

One of the main drivers for this general revision is to ensure that event reporting works consistently between intrinsic and algorithmic reporting, including fault reporting.

Included in this section is a change to the Load Control object to generate CHANGE_OF_STATE notifications instead of COMMAND_FAILURE notifications.

New Clause 13.2: Event Reporting Concepts
The new language for the concepts provides clear identification, separation and general description of event state detection, event notification distribution, alarm acknowledgement and event summarization. Also, it clearly separates the fault evaluation for the determination of the Reliability from event state detection.

New Clause 13.3: Event Algorithms
The event algorithms are now described using more formal language, a formal method, and a style template. The algorithms are generally applicable for both algorithmic and intrinsic reporting, through a defined interface. The event algorithms have been extended to allow additional transitions, while preserving backward compatibility.

New Clause 13.4: Fault Algorithms
Fault algorithms have been added using the same formal language, formal method, and a style template as for the event algorithms. The fault algorithms are used in the evaluation of reliability, not the event state.

Section 16 to Section 28: Particular Changes
These changes complement object and service specifications to realize the event reporting concept, to add particular features and to clarify details of event reporting.

Section 16: Add in the Ability to Configure Event Message Text
A new property is added to all event generating objects that allows configuring the event message text to be conveyed with event notifications.

Section 17: Add in an Intrinsic Reporting Enable / Disable Property
To allow the intrinsic reporting in an object to be enabled / disabled, and to allow the indication of whether or not an object is configured for intrinsic reporting, the property Event_Detection_Enable is added to all objects types that support intrinsic reporting.

Section 18: Add in the Ability to Dynamically Suppress Event Detection
To allow the detection of offnormal conditions to be dynamically suppressed, two properties are added to all event-initiating objects. Suppression of OFFNORMAL event states can be controlled either locally, or by some value in another object of the device.

Section 19: Add in the Ability to Specify a Different Time Delay for TO-NORMAL Transitions
To allow a different time delay to be used for TO-NORMAL event transitions, the Time_Delay_Normal property is added to all event-initiating objects. The Time_Delay_Normal property is added to the Event Enrollment object as a property instead of as a field in the BACnetEventParameter production in order to keep backward compatibility.

Section 20: Add in the Ability to Inhibit the Evaluation of Fault Conditions
To inhibit the detection and reporting of Fault conditions, the Reliability_Evaluation_Inhibit property is added to all objects types that contain a Reliability property.

Section 21: Separate the Detection of Fault Conditions from Intrinsic Reporting
In object types where some fault conditions are defined by properties in the object (i.e. Fault_Values), the properties associated with reliability-evaluation are separated from intrinsic reporting. This allows an object to detect fault conditions using these properties when the object does not support intrinsic reporting.

Section 22: Specify the Content of the Message Text in Acknowledgement Notifications
When acknowledging an event notification, the acknowledging device is given the opportunity to provide the acknowledgment source. To allow for the sharing of the acknowledgement source, this change mandates that the source be distributed as the Message Text parameter of acknowledgement notifications.

Section 23: Ensure that Event Notifications are not Ignored due to Character Set Issues
This change ensures that event notifications are processed even when the Message Text parameter is in an unknown character set.

Section 24: Make the Event Reporting Property Descriptions Consistent
The unification of the property descriptions helps to ensure that event algorithms and event reporting properties are specified and applied consistently throughout the standard.

The property descriptions for event reporting properties that are the parameters to an object's event algorithm are changed to rely on the description of the event algorithm, thereby not replicating it.

Section 25: Identify the Property in each Object that is monitored by Intrinsic Reporting
In extracting the algorithm descriptions from the object types, the identification of the value(s) that are monitored by the event algorithm is lost. This section adds back in the identification, and links the property to the event algorithm parameter for intrinsic reporting.

Section 26: Change the description of the Reliability Property
This allows objects to use any of the Reliability values that are appropriate as defined by the central table of Reliability values in the Clause 12 overview.

Section 27: Improve Fault Detection in Event Enrollment Objects
This improves the fault detection in the Event Enrollment objects so that faults in the monitored object and faults in the Event Enrollment object can be reported and distinguished.

Section 28: Provide for Object Types that only Send Fault Notifications
The Schedule, Program, and Credential Data Input object types are able to detect faults but are not currently able to send notifications indicating the faults. This change adds the ability for those objects to do so. Other access control objects may also have a Reliability property, but will become unreliable only to indicate configuration errors. Adding intrinsic reporting was considered too heavy-weight for these objects.

Section 29 and Section 30: Notification Forwarding
Notification forwarding is a new feature added to the event notification distribution.

Section 29: Add a Notification Forwarder Object Type
This part adds a new Notification Forwarder object. This new object adds important features in notification distribution. It allows reducing the number of recipient lists, and supports simple devices with limited capabilities to participate in sophisticated distribution scenarios. In addition to the rather static enrollment for events, it supports dynamic subscription for events.

Section 30: Reduce the Requirements on Notification-Servers
To reduce the level of effort required to implement basic event reporting, the notification distribution requirements are reduced. This change works in concert with the addition of the Notification Forwarder object that may be in another device.

Section 31: Add an Alert Enrollment Object Type
This new object allows any object to indicate the occurrence of interesting events that are unrelated to the object's state, event state or event algorithm. The events reported through this new mechanism are considered stateless events, not related to an event state machine. There is neither acknowledgement nor return from event concepts. The event identification and event parameters are determined by the vendor indicated in the notifications.

4 Acknowledgments
The committee wishes to thank all contributors to this addendum:

Carl Neilson, Delta Controls
Clifford H. Copass, Johnson Controls
René Kälin, Siemens
Bernhard Isler, Siemens
Christoph Zeller, Sauter
Roland Laird, Reliable Controls
Craig Gemmill, Tridium
Bill Swan, Alerton / Honeywell
Attendees of Alarm Summit Meetings, OS-WG Meetings and Teleconferences
Public Review Commenters

135-2010*af*-1. Remove Annex C and Annex D.

Rationale

Annex C and Annex D are being removed from the standard because they do not provide information that is not already clear from other parts of the standard, and they have proven to be problematic with respect to maintenance of the standard.

[Replace Annex C, p. 698]

ANNEX C - FORMAL DESCRIPTION OF OBJECT TYPE STRUCTURES (INFORMATIVE)

This annex was removed from the standard by Addendum 135-2010af.

[Replace Annex D, p. 723]

ANNEX D - EXAMPLES OF STANDARD OBJECT TYPES (INFORMATIVE)

This annex was removed from the standard by Addendum 135-2010af.

135-2010*af*-2. Clarify Optionality of Properties Related to Intrinsic Event Reporting.

Rationale

The standard is not clear on whether or not properties that are related to intrinsic event reporting are allowed to be included in object instances that do not support intrinsic reporting.

The majority of these properties are not allowed except when intrinsic reporting is supported. The properties that are allowed to be present in an object that does not support intrinsic reporting are:

Adjust_Value, Fault_Values, Feedback_Value, Limit_Monitoring_Interval, Pulse_Rate, Occupancy_Count, Occupancy_Count_Enable.

The changes are applied to all objects types defined in the standard and all addenda that support intrinsic reporting.

Note that Event Enrollment and Notification Class objects do not support intrinsic reporting (they play other roles in event reporting) and as such this section does not apply to these object types.

[Add Footnote to every "Table 12-X. Properties of..." for object types that allow support of intrinsic reporting, and apply footnote to every property that is currently footnoted as required for intrinsic reporting, except: Adjust_Value, Fault_Values, Feedback_Value, Limit_Monitoring_Interval, Occupancy_Count, Occupancy_Count_Enable, Pulse_Rate.]

[Note that each object type that supports intrinsic reporting already has a footnote x as shown below, or one of similar wording, and this change adds footnote y.]
[Note that Event Enrollment and Notification Class objects do support intrinsic reporting (they play other roles in event reporting) and as such this section does not apply to those object types.]

Property Identifier	Property Datatype	Conformance Code
Event_Enable	BACnetEventTransitionBits	O[x,y]

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

[Change Footnote 4 to Table 12-36, , Access Point properties, p. 313]

[4]These properties are required ~~to be present~~ if the object supports intrinsic reporting.

[Change Footnote 3 to Table 12-31, Event Log properties", p. 287]

[3]These properties are required ~~to be present~~ if the object supports intrinsic reporting.

[Change Footnote 4 to Table 12-35, Trend Log Multiple properties, p. 305]

[3]These properties are required ~~to be present~~ if the object supports intrinsic reporting.

135-2010*af*-3. Clarify Optionality of Properties Related to Change of Value Reporting.

Rationale

The standard is not clear on whether or not properties that are related to change of value reporting are allowed to be included in object instances that do not support change of value reporting.

This change clarifies that none of these properties are allowed in an object that does not support COV reporting.

The changes are applied to all objects types defined in the standard and all addenda that support COV reporting through the SubscribeCOV service.

[Change Footnote x to every "Table 12-X. Properties of..." for object types that identify properties required for COV reporting.]

[x] This property is required if*, and shall be present only if,* the object supports COV reporting.

135-2010*af*-4. Ensure that Pulse_Rate and Limit_Monitoring_Interval are Always Together.

Rationale

Limit_Monitoring_Interval is required in order to calculate Pulse_Rate but has no useful purpose if Pulse_Rate is missing. This change ensures that, if one is present, then the other will also be present.

[Add Footnote to Table 12-1, p. 147]

Table 12-1. Properties of the Accumulator Object

Property Identifier	Property Datatype	Conformance Code
…		
Pulse_Rate	Unsigned	O[1,4,x]
…		
Limit_Monitoring_Interval	Unsigned	O[4,x]
…		

[1] This property is required to be writable when Out_Of_Service is TRUE.
…
[4] These properties are required if the object supports intrinsic reporting.
[x] *If one of these properties is present, then both shall be present.*
…

[Change Clause 12.1.23, p. 154]

12.1.23 Limit_Monitoring_Interval

This property, of type Unsigned, specifies the monitoring period in seconds for determining the value of Pulse_Rate. The use of a fixed or sliding time window for detecting pulse rate is a local matter. ~~This property is required if this object supports intrinsic reporting.~~

135-2010*af*-5. Clarify when Priority_Array and Relinquish_Default are allowed to be Present.

Rationale

Clarify that the presence of Priority_Array and Relinquish_Default indicate commandability of a value object.

[Change Footnote x to every "Table 12-X. Properties of..." containing Priority_Array and Relinquish_Default properties]

Property Identifier	Property Datatype	Conformance Code
…		
Priority_Array	BACnetPriorityArray	O[x]
Relinquish_Default	REAL	O[x]
…		

[x] ***These properties are required if, and shall be present only if,*** ~~**If**~~ **Present_Value is commandable~~, then both of these properties shall be present~~.**

[Change Clauses "12.X.Y1 Priority_Array" property description in all value objects.]

[Note that the word "optional" may or may not be present. If present, it shall be stricken.]

12.X.Y1 Priority_Array

This [~~optional~~] property is a read-only array that contains prioritized commands that are in effect for this object. See Clause 19 for a description of the prioritization mechanism. ~~If either the Priority_Array property or the Relinquish_Default property are present, then both of them shall be present. If Present_Value is commandable, then Priority_Array and Relinquish_Default shall both be present.~~

[Change Clauses "12.X.Y2 Relinquish_Default" property description in all value objects.]

[Note that there are currently 2 different forms of text for Relinquish_Default property descriptions. The two different forms are shown below identifying the resulting changes. In the second case, the datatype varies depending on the object type and the datatype shown is only an example. This changes unifies the language for all Relinquish_Default properties.]

[Note that the word "optional" may or may not be present. If present, it shall be stricken.]

[version 1]

12.X.Y2 Relinquish_Default

This [~~optional~~] property is the default value to be used for the Present_Value property when all command priority values in the Priority_Array property have a NULL value. See Clause 19. ~~If either the Relinquish_Default property or the Priority_Array property are present, then both of them shall be present. If Present_Value is commandable, then Priority_Array and Relinquish_Default shall both be present.~~

[version 2]

12.X.Y2 Relinquish_Default

This [~~optional~~] property~~, of type BACnetDateTime,~~ is the default value to be used for the Present_Value property when all command priority values in the Priority_Array property have a NULL value. See Clause 19. ~~If either the Relinquish_Default property or the Priority_Array property is present, then both of them shall be present. If Present_Value is commandable, then Priority_Array and Relinquish_Default shall both be present.~~

135-2010*af*-6. Clarify when Segmentation Related Properties are Allowed to be Present.

Rationale

Clarify that Max_Segments_Accepted and APDU_Segment_Timeout shall only be present if the device supports segmentation.

[Change Footnote 1 to Table 12-13, p. 196]

[1] ~~**Required if**~~ ***These properties are required if, and shall be present only if,*** **segmentation of any kind is supported.**

[Change Clause 12.11.27, p 200]

12.11.27 APDU_Segment_Timeout

The APDU_Segment_Timeout property, of type Unsigned, shall indicate the amount of time in milliseconds between retransmission of an APDU segment. A suggested value for this property is 5000 milliseconds. This value shall be non-zero if the Device object property called Number_Of_APDU_Retries is non-zero. See *Clause* 5.3. ~~**If segmentation of any kind is supported, then the APDU_Segment_Timeout property shall be present.**~~

In order to achieve reliable communication, it is recommended that the values of the APDU_Segment_Timeout properties of the Device objects of all intercommunicating devices should contain the same value.

135-2010*af*-7. Clarify when Virtual Terminal Related Properties are Allowed to be Present.

Rationale

Clarify that VT_Classes_Supported and Active_VT_Sessions shall be present only if the device supports Virtual Terminal services.

[Change Footnote 2 to Table 12-13, p. 196]

[2] ~~**If one of the properties VT_Classes_Supported or Active_VT_Sessions is present, then both of these properties shall be present. Both properties are required, if support for VT Services is indicated in the PICS.**~~ ***These properties are required if, and shall be present only if, the VT Services are supported.***

[Change Clause 12.11.21, p 199]

12.11.21 VT_Classes_Supported

The VT_Classes_Supported property is a List of BACnetVTClass each of which is an enumeration indicating a particular set of terminal characteristics. A given BACnet Device may support multiple typ es of behaviors for differing types of terminals or differing types of operator interface programs. At a minimum, such devices shall support the "Default-terminal" VT-class defined in *Clause* 17.5.

~~**If one of the properties VT_Classes_Supported or Active_VT_Sessions is present, then both of these properties shall be present. Both properties are required if support for VT Services is indicated in the PICS.**~~

[Change Clause 12.11.22, p 199]

12.11.22 Active_VT_Sessions

The Active_VT_Sessions property is a List of BACnetVTSession each of which consists of a Local VT Session Identifier, a Remote VT Session Identifier, and Remote VT Address. This property provides a network-visible indication of those virtual terminal sessions (VT-Sessions) that are active at any given time. Whenever a virtual terminal session is created with the VT-Open service, a new entry is added to the Active_VT_Sessions list. Similarly, whenever a VT-session is terminated, the corresponding entry shall be removed from the Active_VT_Sessions list.

~~**If one of the properties VT_Classes_Supported or Active_VT_Sessions is present, then both of these properties shall be present. Both properties are required if support for VT Services is indicated in the PICS.**~~

135-2010*af*-8. Clarify when Time Synchronization Interval Properties are Allowed to be Present.

Rationale

Clarify that Time_Synchronization_Interval, Align_Intervals and Interval_Offset shall be present only if the device supports one of the time synchronization recipient list properties.

[Change Footnote 5 and Footnote 14 of Table 12-13, p. 196]

[5] ~~**If this property is present, then Time_Synchronization_Interval, Align_Intervals and Interval_Offset shall be present.**~~ **If present, this property shall be writable.**

...

[14] ~~**If**~~ ***These properties are required if, and shall be present only if,*** ~~**either**~~ **Time_Synchronization_Recipients or UTC_Time_Synchronization_Recipients is present.** ~~**, then this property shall be present. and**~~ ***If present, these properties shall be*** **writable.**

135-2010*af*-9. Clarify when Backup and Restore Properties are Allowed to be Present.

Rationale

Clarify that Configuration_Files, Last_Restore_Time, Backup_Failure_Timeout, Backup_Preparation_Time, Restore_Preparation_Time, Restore_Completion_Time and Backup_And_Restore_State shall only be present if the device supports the Backup and Restore procedures.

[Change Footnote 7 and Footnote 8 to Table 12-13, p. 196]

[7] **These properties are required if*, and shall be present only if,* the device supports execution of the backup and restore procedures.**

[8] **This property ~~shall be present~~ *is required if, and shall be present only if,* ~~and writable if~~ the device supports *execution of* the backup and restore procedures. *If present, this property shall be writable.***

[Add footnote to entry in Table 12-13, p. 196]

Table 12-13. Properties of the Device Object Type

Property Identifier	Property Datatype	Conformance Code
…		
Backup_And_Restore_State	BACnetBackupState	~~O~~*O*[7]
…		

[Add Footnote x to Table 12-13, p. 196]

Table 12-13. Properties of the Device Object Type

Property Identifier	Property Datatype	Conformance Code
…		
Backup_Preparation_Time	Unsigned16	O[x]
Restore_Preparation_Time	Unsigned16	O[x]
Restore_Completion_Time	Unsigned16	O[x]
…		

[x] ***These properties are required if, and shall be present only if, the device supports execution of the backup and restore procedures as described in Clause 19.1 and cannot respond to subsequent communications within the minimum value it will accept in its APDU_Timeout property.***

[Change Clause 12.11.37, p. 201]

12.11.37 Last_Restore_Time

This ~~optional~~ property, of type BACnetTimeStamp, is the time at which the device's image was last restored as described in Clause 19.1. ~~This property must be supported if the device supports the BACnet backup and restore procedures as described in Clause 19.1.~~

[Change Clauses 12.11.51 through 12.11.54, p. 203]

12.11.51 Backup_Preparation_Time

This ~~optional~~ property, of type Unsigned16, indicates the amount of time in seconds that the device might remain unresponsive after the sending of a ReinitializeDevice-ACK at the start of a backup procedure. The device that initiated the backup shall either wait the period of time specified by this property or be prepared to encounter communication timeouts during this period. The use of this value is described in more detail in Clause 19.1.

~~**This property is required if the device supports execution of the backup and restore procedures and cannot complete backup preparation within the minimum value it will accept in its APDU_Timeout property.**~~

12.11.52 Restore_Preparation_Time

This ~~optional~~ property, of type Unsigned16, indicates the amount of time in seconds that the device is allowed to remain unresponsive after the sending of a ReinitializeDevice-ACK at the start of a restore procedure. The restoring device shall either wait or be prepared to encounter communication timeouts during this period. The use of this value is described in more detail in Clause 19.1.

~~**This property is required if the device supports execution of the backup and restore procedures and cannot complete restore preparation within the minimum value it will accept in its APDU_Timeout property.**~~

12.11.53 Restore_Completion_Time

This ~~optional~~ property, of type Unsigned16, indicates the amount of time in seconds that the device is allowed to remain unresponsive after the sending of a ReinitializeDevice-ACK at the end of a restore procedure. The restoring device shall either wait or be prepared to encounter communication timeouts during this period. The use of this value is described in more detail in Clause 19.1.

~~**This property is required if the device supports execution of the backup and restore procedures and cannot respond to subsequent communications within the minimum value it will accept in its APDU_Timeout property.**~~

12.11.54 Backup_And_Restore_State

This ~~optional~~ property, of type BACnetBackupState, indicates a server device's backup and restore state. The use of this value is described in more detail in Clause 19.1.

135-2010*af*-10. Clarify when the Active_COV_Subscriptions Property is Allowed to be Present

Rationale

Clarify that Active_COV_Subscriptions shall only be present if the device supports execution of SubscribeCOV or SubscribeCOVProperty.

[Change Footnote 9 to Table 12-13, p. 196]

[9] This property is required if*, and shall be present only if,* the device supports execution of either the SubscribeCOV or SubscribeCOVProperty service.

[Change Clause 12.11.39, p. 201]

12.11.39 Active_COV_Subscriptions

The Active_COV_Subscriptions property is a List of BACnetCOVSubscription, each of which consists of a Recipient, a Monitored Property Reference, an Issue Confirmed Notifications flag, a Time Remaining value and an optional COV Increment. This property provides a network-visible indication of those COV subscriptions that are active at any given time. Whenever a COV Subscription is created with the SubscribeCOV or SubscribeCOVProperty service, a new entry is added to the Active_COV_Subscriptions list. Similarly, whenever a COV Subscription is terminated, the corresponding entry shall be removed from the Active_COV_Subscriptions list.

~~**This property is required if the device supports execution of either SubscribeCOV or SubscribeCOVProperty service.**~~

135-2010*af*-11. Clarify when the Slave Proxy Properties are Allowed to be Present.

Rationale

Clarify that Slave_Proxy_Enable, Manual_Slave_Address_Binding, Auto_Slave_Discovery and Slave_Address_Binding shall only be present if the device is capable of being a Slave-Proxy.

[Change Footnotes to Table 12-13, p. 196]

Table 12-13. Properties of the Device Object Type

Property Identifier	Property Datatype	Conformance Code
…		
Slave_Proxy_Enable	BACnetARRAY[N] of BOOLEAN	O[10]
Manual_Slave_Address_Binding	List of BACnetAddressBinding	O[10,12]
Auto_Slave_Discovery	BACnetARRAY[N] of BOOLEAN	O[10,11]
Slave_Address_Binding	List of BACnetAddressBinding	O[10,12]
…		

…

[10] This property ~~shall be present~~ *is required if, and shall be present only if,* ~~and writable if~~ the device is capable of being a Slave-Proxy device.

[11] This property ~~shall be present if~~ *is required if, and shall be present only if,* the device is capable of being a Slave-Proxy device that implements automatic discovery of slaves.

[12] ~~This property shall be present if the device is capable of being a Slave-Proxy device~~*This property shall be writable if the device is directly connected to an MS/TP network.*

…

[Change Clause 12.11.40, p. 201]

12.11.40 Slave_Proxy_Enable

This property, of type BACnetArray of BOOLEAN, is an indication whether (TRUE) or not (FALSE) the device will perform Slave-Proxy functions for each of the MS/TP ports represented by each array element. ~~This property shall be present if the device is capable of performing the functions of a Slave-Proxy device.~~ The value of this property shall be retained over a device reset.

[Change Clause 12.11.41, p. 201]

12.11.41 Manual_Slave_Address_Binding

This property, of type List of BACnetAddressBinding, describes the manually configured set of slave devices for which this device is acting as a Slave Proxy as described in *Clause* 16.10.2. ~~This property shall be present if the device is capable of performing the functions of a Slave-Proxy device. If present, and the device is directly attached to an MS/TP network, then this property shall be writable.~~

This property is used to manually configure a set of slave devices for which this device will be a proxy. This property allows a Slave Proxy that does not support automatic slave discovery be configured with a set of slaves for which this device will be a proxy. It also allows a Slave-Proxy device to be a proxy for Slave devices that do not support the special object instance of 4194303 as described in Clause 12. The value of this property shall be retained over a device reset. When enabled, the Slave-Proxy device shall periodically check each device that is in this list, and not in the Slave_Address_Binding list, by reading the device's Protocol_Services_Supported property from the device's Device object using the ReadProperty service. If the device responds and indicates that it does not execute the Who-Is service, it shall be added to the Slave_Address_Binding property. The period at which the devices are checked is a local matter.

[Change Clause 12.11.42, p. 202]

12.11.42 Auto_Slave_Discovery

This property, of type BACnetArray of BOOLEAN, is an indication whether (TRUE) or not (FALSE) the device will perform automatic slave detection functions for each of the MS/TP ports represented by each array element. ~~This property shall be present if the device is capable of performing the functions of a Slave-Proxy device.~~ The value of this property shall be retained over a device reset.

Slave detection shall be accomplished by the proxy device using ReadProperty services to read, at a minimum, the Device object's Protocol_Services_Supported property for each MAC address on each port where Auto_Slave_Discovery for that port is TRUE. The ReadProperty service shall use the special object instance of 4194303 as described in Clause 12. If the device is found to support execution of the Who-Is service, it is ignored; otherwise, the device shall be added to the Slave_Address_Binding property. The slave detection algorithm shall be repeated periodically. The period at which it is repeated is a local matter.

[Change Clause 12.11.43, p. 202]

12.11.43 Slave_Address_Binding

This property, of type List of BACnetAddressBinding, describes the set of slave devices for which this device is acting as a Slave-Proxy as described in *Clause* 16.10.2. ~~This property shall be present if the device is capable of performing the functions of a Slave-Proxy device. If present, and the device is directly attached to an MS/TP network, then this property shall be writable.~~

The set of devices described by the Slave_Address_Binding property consists of those devices described in the Manual_Slave_Address_Binding and those devices that are automatically discovered. When enabled, the Slave-Proxy device shall periodically check each device in this list by reading the device's Protocol_Services_Supported property from the device's Device object using the ReadProperty service. If the device fails to respond, or indicates that it executes Who-Is, it shall be removed from the list. The period at which the devices are checked is a local matter.

135-2010*af*-12. Clarify when the Restart Related Properties are Allowed to be Present.

Rationale

Clarify that Last_Restart_Reason, Time_Of_Device_Restart, Restart_Notification_Recipients shall only be present if the device supports execution of SubscribeCOV or SubscribeCOVProperty.

[Change Table 12-13, p. 196]

Table 12-13. Properties of the Device Object Type

Property Identifier	Property Datatype	Conformance Code
…		
Last_Restart_Reason	BACnetRestartReason	O[13]
Time_Of_Device_Restart	BACnetTimeStamp	O[13]
Restart_Notification_Recipients	List of BACnetRecipient	O[13,x]
…		

…

[13] These properties are required if the device supports *execution of* the restart procedure as described in Clause 19.3.

[x] *This property is required if, and shall be present only if, the device supports execution of the restart procedure as described in Clause 19.3.*

…

[Change Clauses 12.11.44, .. 12.11.46, p. 202]

12.11.44 Last_Restart_Reason

This property, of type BACnetRestartReason, indicates the reason for the last device restart. ~~This property shall be present if the device supports the BACnet restart procedure as described in Clause 19.3.~~ *See clause 19.3 for a description of the restart procedure.* The possible values for this property are:

UNKNOWN	The device cannot determine the cause of the last reset.
COLDSTART	A ReinitializeDevice request was received with a 'Reinitialized State of Device' of COLDSTART or the device was made to COLDSTART by some other means.
WARMSTART	A ReinitializeDevice request was received with a 'Reinitialized State of Device' of WARMSTART or the device was made to WARMSTART by some other means.
DETECTED_POWER_LOST	The device detected that incoming power was lost.
DETECTED_POWERED_OFF	The device detected that its power switch was turned off.
HARDWARE_WATCHDOG	The hardware watchdog timer reset the device.
SOFTWARE_WATCHDOG	The software watchdog timer reset the device.
SUSPENDED	The device was suspended. How the device was suspended or what it means to be suspended is a local matter.

12.11.45 Time_Of_Device_Restart

This property, of type BACnetTimeStamp, is the time at which the device was last restarted. ~~This property shall be present if the device supports the BACnet restart procedure as described in Clause 19.3.~~ *See Clause 19.3 for a description of the restart procedure.*

12.11.46 Restart_Notification_Recipients

The Restart_Notification_Recipients property is used to control the restrictions on which devices, if any, are to be notified when a restart occurs. The value of this property shall be a list of zero or more BACnetRecipients. If the list is of length zero, a device is prohibited from sending a device restart notification. The default value of the property shall be a single entry representing a broadcast on the local network. If the property is not writable, then it shall contain the default value. If the list is of length one or more, a device shall send a restart notification, but only to the devices or addresses listed.

~~**This property shall be present if and only if the device supports the BACnet restart procedure as described in Clause 19.3.**~~ ***See Clause 19.3 for a description of the restart procedure.***

135-2010*af*-13. Clarify when the Log_DeviceObjectProperty Property is Allowed to be Present.

Rationale

Clarify that Log_DeviceObjectProperty shall not be present if the Trend Log is monitoring a non-BACnet property.

[Change Table 12-29, p. 272]

Table 12-29. Properties of the Trend Log Object Type

Property Identifier	Property Datatype	Conformance Code
…		
Start_Time	BACnetDateTime	O[1,2]
Stop_Time	BACnetDateTime	O[1,2]
Log_DeviceObjectProperty	BACnetDeviceObjectPropertyReference	O[1x]
Log_Interval	Unsigned	O[1,3]
…		

[x] ***This property is required if, and shall be present only if, the monitored property is a BACnet property.***

135-2010*af*-14. Clarify when the Clock Aligning Properties are Allowed to be Present.

Rationale

Clarify that Align_Intervals and Interval_Offset shall not be present in log objects that do not support clock aligning.

[Change Footnote 5 to Table 12-29, Trend Log property table, p. 272]

[5] These properties are required ~~to be present~~ if*, and shall be present only if,* the object supports clock-aligned logging.

[Change Footnote 3 to Table 12-35, Trend Log Multiple property table, p. 305]

[3] These properties are required ~~to be present~~ if*, and shall be present only if,* the object supports clock-aligned logging.

[Change Clause12.25.28, p. 277]

12.25.28 Align_Intervals

This ~~optional~~ property, of type BOOLEAN, specifies whether (TRUE) or not (FALSE) clock-aligned periodic logging is enabled. If clock-aligned periodic logging is enabled and the value of Log_Interval is a factor of (that is, divides without remainder) a second, minute, hour or day, then the beginning of the period specified for logging shall be aligned to the second, minute, hour or day, respectively.

~~This property is required if the object supports clock-aligned logging.~~ This property has no effect on the behavior of the Trend Log object if the Logging_Type property has a value other than POLLED.

[Change Clause 12.25.29, p. 277]

12.25.29 Interval_Offset

This ~~optional~~ property, of type Unsigned, specifies the offset in hundredths of seconds from the beginning of the period specified for logging until the actual acquisition of a log record begins. The offset used shall be the value of Interval_Offset modulo the value of Log_Interval; i.e., if Interval_Offset has the value 31 and Log_Interval is 30, the offset used shall be 1. Interval_Offset shall have no effect if Align_Intervals is FALSE.

~~This property is required if the object supports clock-aligned logging.~~

[Change Clause 12.30.14, p. 308]

12.30.14 Align_Intervals

This ~~optional~~ property, of type BOOLEAN, specifies whether (TRUE) or not (FALSE) clock-aligned periodic logging is enabled. If clock-aligned periodic logging is enabled and the value of Log_Interval is a factor of (that is, divides without remainder) a second, minute, hour or day, then the beginning of the period specified for logging shall be aligned to the second, minute, hour or day, respectively.

~~This property is required if the object supports clock-aligned logging.~~ This property has no effect on the behavior of the Trend Log Multiple object if the Logging_Type property has a value other than POLLED.

[Change Clause 12.30.15, p. 308]

12.30.15 Interval_Offset

This ~~optional~~ property, of type Unsigned, specifies the offset in hundredths of seconds from the beginning of the period specified for logging until the actual acquisition of a log record begins. The

offset used shall be the value of Interval_Offset modulo the value of Log_Interval; i.e., if Interval_Offset has the value 31 and Log_Interval is 30, the offset used shall be 1. Interval_Offset shall have no effect if Align_Intervals = FALSE.

~~**This property is required if the object supports clock-aligned logging.**~~

135-2010*af*-15. Clarify when the Occupancy Counting Properties are Allowed to be Present.

Rationale

Clarify that Occupancy_Count, Occupancy_Count_Enable, and Adjust_Value shall not be present in Access Zone objects that do not support occupancy counting.

[Change Footnote 4 to Table 12-37, p.329]

[4] These properties are required if*, and shall be present only if,* the object supports occupancy counting.

[Change Clause 12.32.11, , p. 331]

12.32.11 Occupancy_Count

This ~~optional~~ property, of type Unsigned, is used to indicate the actual occupancy count of a zone. If the value of the Occupancy_Count_Enable property is FALSE, then this property shall have a value of zero. The value of the Occupancy_Count property may be adjusted by writing to the Adjust_Value property. The Occupancy_Count property shall be writable when Out_Of_Service is TRUE. When Out_Of_Service becomes FALSE, it is a local matter as to what value this property is set to.

~~If occupancy counting is supported by this object, then this property shall be present.~~

~~This property is required if intrinsic reporting is supported by this object.~~

[Change Clause 12.32.12, p. 332]

12.32.12 Occupancy_Count_Enable

This ~~optional~~ property, of type BOOLEAN, indicates whether occupancy counting is enabled (TRUE) or not (FALSE).

If this property has a value of FALSE, then the Occupancy_State property shall have a value of DISABLED.

When this property changes from FALSE to TRUE it is a local matter as to what value the Occupancy_Count property is set to.

~~If occupancy counting is supported by this object, then this property shall be present.~~

~~This property is required if intrinsic reporting is supported by this object.~~

[Change Clause 12.32.13, p. 332]

12.32.13 Adjust_Value

This ~~optional~~ property, of type INTEGER, shall adjust the Occupancy_Count property when written.

The following series of operations shall be performed atomically when this property is written and the value of the Occupancy_Count_Enable property is TRUE:

(1) The value written to Adjust_Value shall be stored in the Adjust_Value property.

(2) If the value written is non-zero, then this value shall be added to the value of the Occupancy_Count property. If the value written is negative and the resulting value of the Occupancy_Count property would be less than zero, then the Occupancy_Count property shall

be set to zero. If the value written is zero, then the value of the Occupancy_Count property shall be set to zero.

When this property is written and the value of the Occupancy_Count_Enable property is FALSE, then the Adjust_Value property shall be set to zero.

If Adjust_Value has never been written or the Occupancy_Count_Enable property is FALSE, then this property shall have a value of zero.

~~**If occupancy counting is supported by this object, then this property shall be present and writable.**~~

~~**This property is required if intrinsic reporting is supported by this object.**~~

135-2010*af*-16. Add the Ability to Configure Event Message Text.

Rationale

In order to allow the interoperable configuration of the message text that is sent with an event notification, the Event_Message_Texts_Config property is added to all object types that are capable of event reporting.

The changes are applied to all objects types defined in the standard and all addenda that support intrinsic or algorithmic event reporting.

[Add entry to every "Table 12-X. Properties of..." for object types that are allowed to support intrinsic reporting.]

Table 12-X. Properties of...

Property Identifier	Property Datatype	Conformance Code
…		
Event_Message_Texts_Config	*BACnetARRAY[3] of CharacterString*	*O*[y]
…		

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

[Add entry to Table 12-14, p. 205]

Table 12-14. Properties of the Event Enrollment Object Type

Property Identifier	Property Datatype	Conformance Code
…		
Event_Message_Texts_Config	*BACnetARRAY[3] of CharacterString*	*O*
…		

[Add new Clause 12.X.Y to all object types that are allowed to support intrinsic or algorithmic reporting.]

12.X.Y 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.

[Change Clause 21, p 615]

```
BACnetPropertyIdentifier ::= ENUMERATED {
   …
   event-message-texts              (351),
   event-message-texts-config       (352),
   …

   -- see event-message-texts       (351),
   -- see event-message-texts-config (352),
   ...
   }
```

135-2010*af*-17. Add an Event Detection Enable / Disable Property.

Rationale

In order to allow the intrinsic or algorithmic reporting in an object to be enabled / disabled and to allow the indication of whether or not an object is configured for intrinsic reporting, the property Event_Detection_Enable is added to all objects types that support event detection.

The changes are applied to all objects types defined in the standard and all addenda that support event detection.

[Add entry to every "Table 12-X. Properties of..." for object types that are allowed to support intrinsic reporting.]

[Note that each of the object types already has a footnote x with the text as shown, and each object type will have a footnote y added by Section 2 of this addendum]

Table 12-X. Properties of...

Property Identifier	Property Datatype	Conformance Code
…		
Event_Detection_Enable	*BOOLEAN*	*O*[x,y]
…		

[x] **These properties are required if the object supports intrinsic reporting.**
[y] ***These properties shall be present only if the object supports intrinsic reporting.*** **[Added by Section 2]**

[Add entry to Table 12-14, p. 205]

Table 12-14. Properties of the Event Enrollment Object Type

Property Identifier	Property Datatype	Conformance Code
…		
Event_Detection_Enable	*BOOLEAN*	*R*
…		

[Add new Clause 12.X.Y to all object types that are allowed to support intrinsic or algorithmic reporting.]

12.X.Y 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.

[Change Clause 13.10.2, the GetAlarmSummary service procedure, p. 446]

13.10.2 Service Procedure

After verifying the validity of the request, the responding BACnet-user shall search all event-initiating objects that have an Event_State property not equal to NORMAL and a Notify_Type property whose

value is ALARM. *Any object that has an Event_Detection_Enable property with a value of FALSE shall be ignored.* A positive response containing the alarm summaries for objects found in this search shall be constructed. If no objects are found that meet these criteria, then a list of length zero shall be returned.

[Change Clause 13.11.2, the GetEnrollmentSummary service procedure, p. 449]

13.11.2 Service Procedure

After verifying the validity of the request, the responding BACnet-user shall search all event-initiating objects that meet the search criteria specified in the request primitive. The search criteria are the logical conjunctions of all of the explicitly stated filters and the default values for any filters that were omitted in the request primitive. . *Any object that has an Event_Detection_Enable property with a value of FALSE shall be ignored.* A positive response containing the enrollment summaries for objects found in this search shall be constructed. If no objects are found that meet these criteria, then a list of length zero shall be returned.

[Change Clause 13.12.2, the GetEventInformation service procedure, p. 451]

13.12.2 Service Procedure

After verifying the validity of the request, the responding BACnet-user shall search for all event-initiating objects *that do not have an Event_Detection_Enable property with a value of FALSE and* that meet the following conditions, beginning with the object following (in whatever internal ordering of objects is used by the responding device) the object specified by the 'Last Received Object Identifier' parameter, if present:

(a) have an Event_State property whose value is not equal to NORMAL, or
(b) have an Acked_Transitions property that has at least one of the following bits (TO-OFFNORMAL, TO-FAULT, TO-NORMAL) set to FALSE.

A positive response containing the event summaries for objects found in this search shall be constructed. If no objects are found that meet these criteria, then a list of length zero shall be returned. As many of the included objects as can be returned within the APDU shall be returned. If more objects exist that meet the criteria but cannot be returned in the APDU, the 'More Events' parameter shall be set to TRUE, otherwise it shall be set to FALSE.

[Change Clause 21, p 615]

```
BACnetPropertyIdentifier ::= ENUMERATED {
   ...
   error-limit                      (34),
   event-detection-enable           (353),
   ...
   -- see event-message-texts       (351),
   -- see event-detection-enable    (353),
   ...
   }
```

135-2010*af*-18. Add the Ability to Dynamically Suppress Event Detection.

Rationale

To allow the detection of off-normal conditions to be dynamically suppressed, two properties are added to all event-initiating objects.

The changes are applied to all objects types defined in the standard and all addenda that support intrinsic or algorithmic event reporting.

[Add entry to every "Table 12-X. Properties of..." for object types that are allowed to support intrinsic reporting]
[Note that each object type will have a footnote y added by Section 2 of this addendum]

Table 12-X. Properties of...

Property Identifier	Property Datatype	Conformance Code
…		
Event_Algorithm_Inhibit_Ref	*BACnetObjectPropertyReference*	*O*[y]
Event_Algorithm_Inhibit	*BOOLEAN*	*O*[y,z]
…		

[y] *These properties shall be present only if the object supports intrinsic reporting.* **[Added by Section 2]**
[z] *Event_Algorithm_Inhibit shall be present if Event_Algorithm_Inhibit_Ref is present.*

[Add entry to Table 12-14, p. 205]

Table 12-14. Properties of the Event Enrollment Object Type

Property Identifier	Property Datatype	Conformance Code
…		
Event_Algorithm_Inhibit_Ref	*BACnetObjectPropertyReference*	*O*
Event_Algorithm_Inhibit	*BOOLEAN*	*O*[w]
…		

[w] *Event_Algorithm_Inhibit shall be present if Event_Algorithm_Inhibit_Ref is present.*

[Add new Clause 12.X.Y1 to all object types that are allowed to support intrinsic or algorithmic reporting.]

12.X.Y1 Event_Algorithm_Inhibit_Ref

This property, of type BACnetObjectPropertyReference, indicates the property which controls the value of property Event_Algorithm_Inhibit. When this property is present and initialized (contains an instance other than 4194303), the referenced property shall be of type BACnetBinaryPV or BOOLEAN.

[Add new Clause 12.X.Y2 to all object types that are allowed to support intrinsic or algorithmic reporting.]
[Note that the content of the referenced clause, 13.2.2.1, is found in Section 32 of this addendum.]

12.X.Y2 Event_Algorithm_Inhibit

This property, of type BOOLEAN, indicates whether (TRUE) or not (FALSE) the event algorithm has been disabled for the object (see Clause 13.2.2.1). This property is a runtime override that allows temporary disabling of the event algorithm.

If the Event_Algorithm_Inhibit_Ref property is present and initialized (contains an instance other than 4194303), then the Event_Algorithm_Inhibit property shall be read only and shall reflect the value of the property referenced by Event_Algorithm_Inhibit_Ref. A BACnetBinaryPV value of INACTIVE

shall map to a value of FALSE and a value of ACTIVE shall map to a value of TRUE. If the referenced property does not exist, it shall be assumed to have a value of FALSE.

If the Event_Algorithm_Inhibit_Ref property is absent or is uninitialized and Event_Detection_Enable is TRUE, then the Event_Algorithm_Inhibit property shall be writable.

[Change Clause 21, p 615]

```
BACnetPropertyIdentifier ::= ENUMERATED {
   ...
   error-limit                         (34),
   event-algorithm-inhibit             (354),
   event-algorithm-inhibit-ref         (355),
   ...
   -- see event-message-texts          (351),
   -- see event-algorithm-inhibit      (354),
   -- see event-algorithm-inhibit-ref  (355),
   ...
   }
```

135-2010*af*-19. Add the Ability to Specify a Different Time Delay for TO-NORMAL Transitions.

Rationale

In order to allow a different time delay to be used for TO-NORMAL event transitions, the Time_Delay_Normal property is added to all event-initiating objects.

The changes are applied to all objects types defined in the standard and all addenda that support intrinsic or algorithmic event reporting.

The Time_Delay_Normal property is added to the Event_Enrollment object as a property instead of as a field in the BACnetEventParameter production in order to keep backward compatibility.

[Add entry to every "Table 12-X. Properties of..." for object types that are allowed to support intrinsic reporting and have a Time_Delay property]
[Note that each of the object types already has a footnote y, added by Section 2 of this addendum]

Table 12-X. Properties of...

Property Identifier	Property Datatype	Conformance Code
…		
Time_Delay_Normal	*Unsigned*	*O*[y]
…		

[y] ***These properties shall be present only if the object supports intrinsic reporting.*** **[Added by Section 2]**

[Add entry to Table 12-14, p. 205]

Table 12-14. Properties of the Event Enrollment Object Type

Property Identifier	Property Datatype	Conformance Code
…		
Time_Delay_Normal	*Unsigned*	*O*
…		

[Add new Clause 12.X.Y1 to all object types that are allowed to support intrinsic or algorithmic reporting.]
[Note that the changed content of the referenced clause, 13.3, is found in Section 32 of this addendum.]

12.X.Y Time_Delay_Normal

This property, of type Unsigned, is the pTimeDelayNormal parameter for the object's event algorithm. See Clause 13.3 for event algorithm parameter descriptions.

[Change Clause 21, p 618]

```
BACnetPropertyIdentifier ::= ENUMERATED {
   ...
   time-delay                    (113),
   time-delay-normal             (356),
   ...
   -- see event-message-texts    (351),
   -- see time-delay-normal      (356),
   ...
   }
```

135-2010*af*-20. Add the Ability to Inhibit the Evaluation of Fault Conditions.

Rationale

In order to inhibit the detection and reporting of Fault conditions, the Reliability_Evaluation_Inhibit property is added to all objects types that contain a Reliability property.

The changes are applied to all objects types defined in the standard and all addenda that support the Reliability property.

[Add entry to every "Table 12-X. Properties of..." for object types for which the Reliability property is required.]

Table 12-X. Properties of...

Property Identifier	Property Datatype	Conformance Code
…		
Reliability_Evaluation_Inhibit	*BOOLEAN*	*O*
…		

[Add entry to every "Table 12-X. Properties of..." for object types for which the Reliability property is optional.]

Table 12-X. Properties of...

Property Identifier	Property Datatype	Conformance Code
…		
Reliability_Evaluation_Inhibit	*BOOLEAN*	*O*[x]
…		

[x] *If this property is present, then the Reliability property shall be present.*

[Add new Clause 12.X.Y to all object types that support a Reliability property.]
[Note that the term "reliability-evaluation" is defined in a different section of this addendum.]

12.X.Y 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.

[Change Clause 21, p 617]

```
BACnetPropertyIdentifier ::= ENUMERATED {
   ...
   reliability                         (103),
   reliability-evaluation-inhibit      (357),
   ...
   -- see event-message-texts          (351),
   -- see reliability-evaluation-inhibit (357),
   ...
   }
```

135-2010*af*-21. Separate the Detection of Fault Conditions from Intrinsic Reporting.

Rationale

In object types where some fault conditions are defined by properties in the object (i.e., Fault_Values), the properties associated with reliability-evaluation are separated from intrinsic reporting by this addendum. This will allow an object to be able to detect fault conditions using these properties when the object does not support intrinsic reporting.

The changes are applied to all objects types that support the Fault_Values property which are defined in the standard and all addenda.

[Change "Table 12-X. Properties of..." for object types which have a Fault_Values property and for which the Reliability property is required.]

Table 12-X. Properties of...

Property Identifier	Property Datatype	Conformance Code
…		
Fault_Values	List of ...	O[x]
…		

…

[x] ~~These properties are required if the object supports intrinsic alarming.~~

…

[Change "Table 12-X. Properties of..." for object types which have a Fault_Values property and for which the Reliability property is optional.]

Table 12-X. Properties of...

Property Identifier	Property Datatype	Conformance Code
...		
Reliability	BACnetReliability	O[2]
Fault_Values	List of ...	O[x,w]
...		

…

[2] ~~This property shall be required if Fault_Values is present.~~

…

[x] These properties are required if the object supports intrinsic reporting.

…

[w] *If this property is present, then the Reliability property shall be present.*

[Change Clause 12.X.X for object types which have a Fault_Values property and for which the Reliability property is optional.]

12.X.X Reliability

~~…~~

~~The Reliability property is required to be present if the Fault_Values property is present.~~

[Change Clause 12.X.Y for object types which have a Fault_Values property]

12.X.Y Fault_Values

This ~~optional~~ property, of type List of …

135-2010*af*-22. Specify the Content of the Message Text in Acknowledgement Notifications.

This section was removed after the first public review.

135-2010*af*-23. Ensure that Event Notifications are not Ignored due to Character Set Issues.

Rationale

This change ensures that event notifications and acknowledgement requests are processed even when the Message Text or Acknowledgement Source parameter is in an unknown character set.

This change also clarifies that acknowledgment notifications are sent to the currently active set of recipients, not the set of recipients active at the time of the acknowledged event transition.

[Change Clause 13.5.2, p. 436]

13.5.2 Service Procedure

After verifying the validity of the request, the responding BACnet-user shall attempt to locate the specified object. If the object exists and if the 'Time Stamp' parameter matches the most recent time for the event being acknowledged, then the bit in the Acked_Transitions property of the object that corresponds to the value of the 'Event State Acknowledged' parameter shall be set to 1, a 'Result(+)' primitive shall be issued, and an event notification with a 'Notify Type' parameter equal to ACK_NOTIFICATION shall be issued. Otherwise, a 'Result(-)' primitive shall be issued. An acknowledgment notification shall use the same type of service (confirmed or unconfirmed) directed to the same recipients to which ~~the original~~ *a* confirmed or unconfirmed event notification *for the same transition type would be* ~~was~~ sent. The Time Stamp conveyed in the acknowledgement notification shall not be derived from the Time Stamp of the original event notification, but rather the time at which the acknowledgement notification is generated.

A device shall not fail to process, or issue a Result(-), upon receiving an AcknowledgeAlarm service request containing an 'Acknowledgment Source' parameter in an unsupported character set. In this case, it is a local matter whether the 'Acknowledgment Source' parameter is used as provided or whether a character string, in a supported character set, of length 0 is used in its place.

[Change Clause 13.8.2, p. 442]

13.8.2 Service Procedure

After verifying the validity of the request, the responding BACnet-user shall take whatever local actions have been assigned to the indicated event occurrence and issue a 'Result(+)' service primitive. If the service request cannot be executed, a 'Result(-)' service primitive shall be issued indicating the encountered error. *A device shall not fail to process, or issue a Result(-), upon receiving a ConfirmedEventNotification service request containing a 'Message Text' parameter in an unsupported character set.*

[Change Clause 13.9.2, p. 444]

13.9.2 Service Procedure

Since this is an unconfirmed service, no response primitives are expected. Actions taken in response to this notification are a local matter. *However, a device shall not fail to process an UnconfirmedEventNotification service request containing a 'Message Text' parameter in an unsupported character set.*

135-2010*af*-24. Make the Event Reporting Property Descriptions Consistent.

Rationale

The unification of the property descriptions will help to ensure that event algorithms and event reporting properties are specified and applied consistently throughout the standard.

The property descriptions for event reporting properties that are the parameters to an object's event algorithm are changed to rely on the description of the event algorithm. Changes to the way in which event algorithms are described are made in another section of this addendum.

Where clauses are referenced in the next property descriptions, the referenced clauses can be found in another section of this addendum.

[Replace all Clauses 12.X.Y for Event_State properties in the standard and all published addenda to the standard]
[Note that there are currently multiple different descriptions for the Event_State property. All are to be changed to the following text.]

12.X.Y 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.

[For reviewing purposes only, the different versions are shown below with change marks in order to highlight changes in functional behavior.]

[Accumulator, Pulse Converter, Load Control:

The Event_State property, of type BACnetEventState, is included in order to provide a way to determine *whether*~~if~~ this object has an active event state associated with it *(see Clause 13.2.2.1).* If the object supports ~~intrinsic~~ *event* reporting, then the Event_State property shall indicate the event state of the object. If the object does not support ~~intrinsic~~ *event* reporting ~~and if the Reliability property is not present,~~ then the value of this property shall be NORMAL. ~~If the Reliability property is present and does not have a value of NO_FAULT_DETECTED, then the value of the Event_State property shall be FAULT. Changes in the Event_State property to the value FAULT are considered to be "fault" events.~~

]

[Analog Input, Analog Output, Analog Value, Binary Input, Binary Output, Binary Value, Global Group:

The Event_State property, of type BACnetEventState, is included in order to provide a way to determine *whether*~~if~~ this object has an active event state associated with it *(see Clause 13.2.2.1).* If the object supports ~~intrinsic~~ *event* reporting, then the Event_State property shall indicate the event state of the object. If the object does not support ~~intrinsic~~ *event* reporting, then the value of this property shall be NORMAL. ~~If the Reliability property is present and does not have a value of NO_FAULT_DETECTED, then the value of the Event_State property shall be FAULT. Changes in the Event_State property to the value FAULT are considered to be "fault" events.~~

]

[Event Enrollment:

This property, of type BACnetEventState~~, contains the current state of the event. The permitted values for Event_State are specific to the Event_Type. See Table 12-15. The value of the Event_State property is independent of the Event_Enable flags. See 12.12.10.~~ *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.*

]

[Life Safety Point, Access Door:

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

]

[Life Safety Zone:

The Event_State property, of type BACnetEventState, is included in order to provide a way to determine *whether~~if~~* this object has an active event state associated with it *(see Clause 13.2.2.1). ~~The~~ If the object supports event reporting, 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.*

]

[Loop:

The Event_State property, of type BACnetEventState, is included in order to provide a way to determine *whether~~if~~* this object has an active event state associated with it *(see Clause 13.2.2.1).* If the object supports ~~intrinsic~~ *event* reporting, then the Event_State property shall indicate the event state of the object. If the object does not support ~~intrinsic~~ *event* reporting, then the value of this property shall be NORMAL. ~~If the Reliability property is present and does not have a value of NO_FAULT_DETECTED, then the value of the Event_State property shall be FAULT. Changes in the Event_State property to the value FAULT are considered to be "fault" events.~~

]

[Multi-state Input, Multi-state Value:

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

~~(a) if the Reliability property is not present, then the value of Event_State shall be NORMAL, or~~
~~(b) if the Reliability property is present and Reliability is NO_FAULT_DETECTED then Event_State shall be NORMAL, or~~
~~(c) If the Reliability property is present and Reliability is not NO_FAULT_DETECTED then Event_State shall be FAULT.~~

]

[Multi-state Output:

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

]

[Trend Log:

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

]

[Event Log, Trend Log Multiple:

The Event_State property, of type BACnetEventState, is included in order to provide a way to determine *whether*~~if~~ this object has an active event state associated with it *(see Clause 13.2.2.1).* If the object supports ~~intrinsic~~ *event* reporting, then the Event_State property shall indicate the event state of the object. If the object does not support ~~intrinsic~~ *event* reporting, then the value of this property shall be NORMAL. ~~The Event_State property for this object may have either of the following values:~~

~~**{NORMAL, FAULT}**~~

]

[CharacterString Value, Large Analog Value, BitString Value, Integer Value, Positive Integer Value:

~~**This optional property,**~~***The Event_State property,*** **of type BACnetEventState, is included in order to provide a way to determine *whether*~~*if*~~ this object has an active event state associated with it. If the object supports ~~intrinsic~~ *event* reporting, then the Event_State property shall indicate the event state of the object. If the object does not support ~~intrinsic~~ *event* reporting, then the value of this property shall be NORMAL. ~~If the Reliability property is present and does not have a value of NO_FAULT_DETECTED, then the value of the Event_State property shall be FAULT. Changes in the Event_State property to the value FAULT are considered to be "fault" events. This property is required if intrinsic reporting is supported by this object.~~**

]

[DateTime Value, OctetString Value, Time Value, Date Value, DateTime Pattern Value, Time Pattern Value, Date Pattern Value:

~~**This optional property,**~~***The Event_State property,*** **of type BACnetEventState, is included in order to provide a way to determine *whether*~~*if*~~ this object has an active event state associated with it. ~~If the Reliability property is not present, or if the Reliability property is present and has a value of NO_FAULT_DETECTED,~~ *If the object supports event reporting, then the Event_State property shall indeicate the event state of the object. If the object does not support event reporting,* then the value of this property shall be NORMAL. ~~If the Reliability property is present and does not have a value of NO_FAULT_DETECTED, then the value of the Event_State property shall be FAULT.~~**

]

[Change all Clauses 12.X.Y for Status_Flags properties of object types that can support event reporting in the standard and all addenda, with the exception of the Event Enrollment object (Status_Flags for Event Enrollment is added in Section 27 of this addendum and differs from the below language in that the referenced object's Status_Flags are used as the algorithm parameter and not the Event Enrollment's Status_Flags)]

12.X.Y Status_Flags

[current text retained, the following paragraph appended]

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

[Replace all Clause 12.X.Y for High_Limit property descriptions, including any sub-clauses, in the standard and all addenda.]

12.X.Y High_Limit

This property is the pHighLimit parameter for the object's event algorithm. See Clause 13.3 for event algorithm parameter descriptions.

[Replace all Clause 12.X.Y for Low_Limit property descriptions, including any sub-clauses, in the standard and all addenda.]

12.X.Y Low_Limit

This property is the pLowLimit parameter for the object's event algorithm. See Clause 13.3 for event algorithm parameter descriptions.

[Replace all Clause 12.X.Y for Notification_Class property descriptions, including any sub-clauses, in the standard and all addenda.]

12.X.Y Notification_Class

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

[Replace all Clause 12.X.Y for Time_Delay property descriptions, including any sub-clauses, in the standard and all addenda.]

12.X.Y Time_Delay

This property, of type Unsigned, is the pTimeDelay parameter for the object's event algorithm. See Clause 13.3 for event algorithm parameter descriptions.

[Replace all Clause 12.X.Y for Limit_Enable property descriptions, including any sub-clauses, in the standard and all addenda.]

12.X.Y Limit_Enable

This property, of type BACnetLimitEnable, is the pLimitEnable parameter for the object's event algorithm. See 13.3 for event algorithm parameter descriptions.

[Replace all Clause 12.X.Y for Event_Enable property descriptions, including any sub-clauses, in the standard and all addenda.]

12.X.Y 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.

[Replace all Clause 12.X.Y for Acked_Transition property descriptions, including any sub-clauses, in the standard and all addenda.]

12.X.Y 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.

[Replace all Clause 12.X.Y for Notify_Type property descriptions, including any sub-clauses, in the standard and all addenda.]

12.X.Y 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.

[For reviewing purposes only, the different versions are shown with change marks in order to show that there are no changes to functional behavior.]

[All objects except Event Enrollment:

This ~~optional~~ 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. ~~**This property is required if intrinsic reporting is supported by this object.**~~

]

[Event Enrollment:

This property, of type BACnetNotifyType, shall convey whether the notifications generated by the ~~**monitoring algorithm specified by the Event_Type property**~~ ***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.***

]

[Change all Clause 12.X.Y for Event_Time_Stamps property descriptions in the standard and all addenda.]
[Note that the last sentence does not exist in the Event Enrollment object's version.]

12.X.Y Event_Time_Stamps

This ~~**optional**~~ ***read-only*** **property, of type BACnetARRAY[3] of BACnetTimeStamp, shall convey the times of the last** ~~**event notifications for**~~ **TO-OFFNORMAL, TO-FAULT, and TO-NORMAL events**~~**,**~~ ~~**respectively**~~ ***(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** ~~**notification**~~ **of that type has** ***ever occurred for the object.*** ~~**been generated since the object was created. This property is required if this object supports intrinsic reporting.**~~

[Change all Clause 12.X.Y for Event_Message_Texts property descriptions.]

12.X.Y Event_Message_Texts

This ~~**optional**~~ ***read-only*** **property, of type BACnetARRAY[3] of CharacterString, shall convey the** ~~**Message Text**~~ ***message text values*** **of the last** ~~**event notifications for**~~ **TO-OFFNORMAL, TO-FAULT, and TO-NORMAL events, respectively** ***(see Clause 13.2.2.1).*** ~~**This property shall be read-only. If no notification of a transition has occurred yet for a transition,**~~ ***If a particular type of event has yet to occur,*** **an empty string shall be stored in the respective array element.**

[Replace all Clause 12.X.Y for Deadband property descriptions, including any sub-clauses, in the standard and all addenda.]

12.X.Y Deadband

This property is the pDeadband parameter for the object's event algorithm. See Clause 13.3 for event algorithm parameter descriptions.

[Replace all Clause 12.X.Y for Alarm_Value property descriptions, including any sub-clauses, in the standard and all addenda.]

12.X.Y Alarm_Value

This property is the pAlarmValues parameter for the object's event algorithm. See Clause 13.3 for event algorithm parameter description.

[Replace all Clause 12.X.Y for Feedback_Value property descriptions, including any sub-clauses, in the standard and all addenda.]

12.X.Y Feedback_Value

This property is an indication of the actual value of the entity controlled by Present_Value. The manner by which the Feedback_Value is determined shall be a local matter.

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

[Replace all Clause 12.X.Y for Life_Safety_Alarm_Values property descriptions, including any sub-clauses, in the standard and all addenda.]

12.X.Y Life_Safety_Alarm_Values

This property, of type List of BACnetLifeSafetyState, is the pLifeSafetyAlarmValues parameter of the object's event algorithm. See Clause 13.3 for event algorithm parameter descriptions.

[Replace all Clause 12.X.Y for Alarm_Values property descriptions, including any sub-clauses, in the standard and all addenda.]

12.X.Y Alarm_Values

This property is the pAlarmValues parameter of the object's event algorithm. See Clause 13.3 for event algorithm parameter descriptions.

[Replace all Clause 12.X.Y for Fault_Values property descriptions, including any sub-clauses, in the standard and all addenda.]

12.X.Y Fault_Values

This property is the value of the pFaultValues parameter of the object's fault algorithm. See Clause 13.4 for fault algorithm parameter descriptions.

[Replace all Clause 12.X.Y for Notification_Threshold property descriptions, including any sub-clauses, in the standard and all addenda.]

12.X.Y Notification_Threshold

This property is the pThreshold parameter for the object's event algorithm. See 13.3 for event algorithm parameter descriptions.

[Change all Clause 12.X.Y for Records_Since_Notification property descriptions in the standard and all addenda.]

12.X.Y Records_Since_Notification

This property, of type Unsigned32, represents the number of records collected since the previous notification, or since the beginning of logging if no previous notification has occurred. ~~This property is required if intrinsic reporting is supported by this object.~~

[Replace all Clause 12.X.Y for Last_Notify_Record property descriptions, including any sub-clauses, in the standard and all addenda.]

12.X.Y Last_Notify_Record

This property is the pPreviousCount parameter of the object's event algorithm. See Clause 13.3 for event algorithm parameter descriptions.

[Change all Clause 12.X.Y for Total_Record_Count property descriptions, including any sub-clauses, in the standard and all addenda.]

12.X.Y Total_Record_Count

This property, of type Unsigned32, shall represent the total number of records collected by the object since creation. When the value of Total_Record_Count reaches its maximum possible value of $2^{32} - 1$, the next value it takes shall be one. Once this value has wrapped to one, its semantic value (the total number of records collected) has been lost but its use in generating notifications remains.

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 Clause 12.17.14, p. 233]

12.17.14 Controlled_Variable_Value

This property, of type REAL, is the value of the property of the object referenced by the Controlled_Variabel_Reference property. This control loop compares the Controlled_Variable_Value with the Setpoint to calculate the error.

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 Clause 12.17.17, p. 233]

12.17.17 Setpoint

This property, of type REAL, is the value of the loop setpoint or of the property of the object referenced by the Setpoint_Reference, expressed in units of the Controlled_Variable_Units property.

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

[Replace all Clause 12.X.Y for Error_Limit property descriptions, including any sub-clauses, in the standard and all addenda.]

12.X.Y Error_Limit

This property is both the pLowDifLimit and pHighDiffLimit parameter for the object's event algorithm. See 13.3 for event algorithm parameter descriptions.

[Change Clause 12.31.27,p. 320]

12.31.27 Access_Event

This property, of type BACnetAccessEvent, indicates the last access event which occurred at this Access Point. *This property is the pMonitoredValue parameter of the object's ACCESS_EVENT event algorithm for both Access Alarm Events and Access Transaction Events. See Clause 13.3 for event algorithm parameter description.*

BACnetAccessEvent is an enumeration of authentication and authorization decisions, subsequent actions, and results of these actions. An Access Point is not required to support all values of this enumeration.

…

[Change Clause 12.31.28, p. 324]

12.31.28 Access_Event_Tag

This property, of type Unsigned, is a numeric value which identifies the access transaction to which the current access event belongs. Multiple access events may be generated by a single access transaction.

The value of this property shall increase monotonically for each new access transaction. It may be implemented using modulo arithmetic. The initial value of this property before any access transaction has occurred shall be a local matter.

This property is the pAccessEventTag parameter of the object's ACCESS_EVENT event algorithm for both Access Alarm Events and Access Transaction Events. See Clause 13.3 for event algorithm parameter description.

[Change Clause 12.31.29, p. 324]

12.31.29 Access_Event_Time

This property, of type BACnetTimeStamp, indicates the most recent update time of the Access_Event property. This property shall update its value on each update of Access_Event. Update times of type Time or Date shall have X'FF' in each octet, and Sequence number update times shall have the value 0 if no update has yet occurred.

This property is the pAccessEventTime parameter of the object's ACCESS_EVENT event algorithm for both Access Alarm Events and Access Transaction Events. See Clause 13.3 for event algorithm parameter description.

[Change Clause 12.31.30, p. 324]

12.31.30 Access_Event_Credential

This property, of type BACnetDeviceObjectReference, shall specify the Access Credential object that corresponds to the access event specified in the Access_Event property, if applicable. This property shall contain 4194303 in the instance part of the object identifier and in the device instance part of the device identifier, if present, under the following conditions:

(a) there is no credential recognized up to now, or
(b) there is no credential that is associated to the current access event, or
(c) the credential of the authentication factor that is associated to the current event is unknown, or
(d) the device chooses not to expose the credential.

This property is the pAccessCredential parameter of the object's ACCESS_EVENT event algorithm for both Access Alarm Events and Access Transaction Events. See Clause 13.3 for event algorithm parameter description.

[Change Clause 12.31.31, p. 324]

12.31.31 Access_Event_Authentication_Factor

This ~~optional~~ property, of type BACnetAuthenticationFactor, shall specify the authentication factor that corresponds to the access event specified in the Access_Event property, if applicable. Otherwise it shall contain a value of format type UNDEFINED.

This property shall contain a value of format type UNDEFINED under the following conditions:

(a) there was no authentication factor read up to now, or
(b) there is no authentication factor that is associated to the current access event, or
(c) the device chooses not to expose the authentication factor.

This property is the pAccessFactor parameter of the object's ACCESS_EVENT event algorithm for both Access Alarm Events and Access Transaction Events. See Clause 13.3 for event algorithm parameter description.

[Change Clause 12.31.38, p. 325]

12.31.38 Transaction_Notification_Class

This ~~optional~~ property, of type Unsigned, shall specify the ~~notification class~~ *instance of the Notification Class object to use for event-notification-distribution* of Access Transaction Events. ~~The Transaction_Notification_Class property implicitly refers to a Notification Class object that has a Notification_Class property with the same value.~~ If this property is not present, then the Notification Class specified by the property Notification_Class shall be used for Access Transaction Events.

[Change Clause 12.31.39, p. 325]

12.31.39 Access_Alarm_Events

~~This optional property, of type List of BACnetAccessEvent, shall specify any access events which are~~

~~**reported as Access Alarm Events. This property is required if intrinsic reporting is supported by this object.**~~

The value of this property is used as the value of the pAccessEvents parameter of the object's ACCESS_EVENT event algorithm for Access Alarm Events.

[Change Clause 12.31.40, p. 325]

12.31.40 Access_Transaction_Events

~~**This optional property, of type List of BACnetAccessEvent, shall specify any access events that are reported as Access Transaction Events. This property is required if intrinsic reporting is supported by this object.**~~

The value of this property is used as the value of the pAccessEvents parameter of the object's ACCESS_EVENT event algorithm for Access Transaction Events.

[Change Clause 12.40.6 p. 372]

12.40.16 Bit_Mask

~~***This optional property, of type BIT STRING, shall specify a bitmask that is used to indicate which bits in the Present_Value are to be monitored by the intrinsic alarm algorithm. A value of one in a bit position indicates that the bit in this position in the Present_Value is to be monitored by the algorithm. A value of zero in a bit position indicates that the bit in this position in the Present_Value is not significant for the purpose of the algorithm.***~~

This property is the pBitMask parameter of the object's event algorithm. See Clause 13.3 for event algorithm parameter descriptions.

135-2010*af*-25. Identify the Property in each Object that is Monitored by Intrinsic Reporting.

Rationale

In extracting the algorithm descriptions from the object types, the identification of the value(s) that are monitored by the event algorithm is lost. This section adds the identification back in.

Where clauses are referenced in the property descriptions, the referenced clauses can be found in another section of this addendum.

[Change Clause 12.1.20,in the Accumulator Object Type, p. 153]

12.1.20 Pulse_Rate

This property, of type Unsigned, shall indicate the number of input pulses received during the most recent period specified by Limit_Monitoring_Interval. The mechanism that associates the input signal with the value indicated by this property is a local matter.

This property shall be writable when Out_Of_Service is TRUE.

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 Clause 12.2.4, in the Analog Input Object Type, p. 156]

12.2.4 Present_Value

This property, of type REAL, indicates the current value, in engineering units, of the input being measured. The Present_Value property shall be writable when Out_Of_Service is TRUE.

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 Clause 12.3.4, in the Analog Output Object Type, p. 161]

12.3.4 Present_Value (Commandable)

This property, of type REAL, indicates the current value, in engineering units, of the output.

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 Clause 12.4.4, in the Analog Value Object Type, p. 167]

12.4.4 Present_Value

This property, of type REAL, indicates the current value, in engineering units, of the analog value. Present_Value shall be optionally commandable. If Present_Value is commandable for a given object instance, then the Priority_Array and Relinquish_Default properties shall also be present for that instance. The Present_Value property shall be writable when Out_Of_Service is TRUE.

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 Clause 12.6.4, in the Binary Input Object Type, p. 175]

12.6.4 Present_Value

This property, of type BACnetBinaryPV, reflects the logical state of the Binary Input. The logical state of the Input shall be either INACTIVE or ACTIVE. The relationship between the Present_Value and the physical state of the Input is determined by the Polarity property. The possible states are summarized in Table 12-7.

Table 12-7. BACnet Polarity Relationships

Present_Value	Polarity	Physical State of Input	Physical State of Device
INACTIVE	NORMAL	OFF or INACTIVE	not running
ACTIVE	NORMAL	ON or ACTIVE	Running
INACTIVE	REVERSE	ON or ACTIVE	not running
ACTIVE	REVERSE	OFF or INACTIVE	running

The Present_Value property shall be writable when Out_Of_Service is TRUE.

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 Clause 12.7.4, in the Binary Output Object Type, p. 180]

12.7.4 Present_Value (Commandable)

This property, of type BACnetBinaryPV, reflects the logical state of the Binary Output. The logical state of the output shall be either INACTIVE or ACTIVE. The relationship between the Present_Value and the physical state of the output is determined by the Polarity property. The possible states are summarized in Table 12-9.

Table 12-9. BACnet Polarity Relationships

Present_Value	Polarity	Physical State of Output	Physical State of Device
INACTIVE	NORMAL	OFF or INACTIVE	not running
ACTIVE	NORMAL	ON or ACTIVE	running
INACTIVE	REVERSE	ON or ACTIVE	not running
ACTIVE	REVERSE	OFF or INACTIVE	running

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 Clause 12.8.4, in the Binary Value Object Type, p. 186]

12.8.4 Present_Value

This property, of type BACnetBinaryPV, reflects the logical state of the Binary Value. The logical state shall be either INACTIVE or ACTIVE. Present_Value shall be optionally commandable. If Present_Value is commandable for a given instance, then the Priority_Array and Relinquish_Default properties shall also be present for that instance. The Present_Value property shall be writable when Out_Of_Service is TRUE.

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 Clause 12.12.8, in the Event Enrollment Object Type, p. 209]

12.12.8 Object_Property_Reference

This property, of type BACnetDeviceObjectPropertyReference, designates the particular object and property referenced by this Event Enrollment object. The *event* algorithm specified by the Event_Type property is applied to the referenced property in order to determine the Event_State of the event. *The value of the referenced property is used as the value of the pMonitoredValue parameter of the event algorithm. The value of the referenced property is also used as the value of the pMonitoredValue parameter of the fault algorithm, if a fault algorithm is applied.*

If this property is writable, it may be restricted to only support references to objects inside of the device containing the Event Enrollment object. If the property is restricted to referencing objects within the containing device, an attempt to write a reference to an object outside the containing device into this property shall cause a Result(-) to be returned with an error class of PROPERTY and an error code of OPTIONAL_FUNCTIONALITY_NOT_SUPPORTED.

If this property is set to reference an object outside the device containing the Event Enrollment object, the method used for acquisition of the referenced property value for the purpose of monitoring is a local matter. The only restriction on the method of data acquisition is that the monitoring device be capable of using ReadProperty for this purpose so as to be interoperable with all BACnet devices.

Depending on the event algorithm, the values of additional properties of the monitored object are used for particular event algorithm parameters, as specified by Table 12-X1.

Table 12-X1. Additional Monitored Object Properties by Event Algorithm

Event Algorithm	*Additional Monitored Object Properties*	*Event Algorithm Parameters*
NONE	*none*	*none*
ACCESS_EVENT	*Status_Flags* *Access_Event_Tag* *Access_Event_Credential* *Access_Event_Authentication_Factor*	*pStatusFlags* *pAccessEventTag* *pAccessCredential* *pAccessFactor*
BUFFER_READY	*Log_Buffer*	*pLogBuffer references the Log_Buffer property*
CHANGE_OF_BITSTRING	*Status_Flags*	*pStatusFlags*
CHANGE_OF_CHARACTERSTRING	*Status_Flags*	*pStatusFlags*
CHANGE_OF_LIFE_SAFETY	*Status_Flags* *Operation_Expected*	*pStatusFlags* *pOperationExpected*
CHANGE_OF_STATE	*Status_Flags*	*pStatusFlags*
CHANGE_OF_STATUS_FLAGS	*Present_Value*	*pPresentValue*
CHANGE_OF_VALUE	*Status_Flags*	*pStatusFlags*
COMMAND_FAILURE	*Status_Flags*	*pStatusFlags*
DOUBLE_OUT_OF_RANGE	*Status_Flags*	*pStatusFlags*
EXTENDED	*Defined by vendor*	*Defined by vendor*
FLOATING_LIMIT	*Status_Flags*	*pStatusFlags*
OUT_OF_RANGE	*Status_Flags*	*pStatusFlags*
SIGNED_OUT_OF_RANGE	*Status_Flags*	*pStatusFlags*
UNSIGNED_OUT_OF_RANGE	*Status_Flags*	*pStatusFlags*
UNSIGNED_RANGE	*Status_Flags*	*pStatusFlags*

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-X2.

Table 12-X2. 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*

[Change Clause 12.15.4, in the Life Safety Point Object Type, p. 218]

12.15.4 Present_Value

This property, of type BACnetLifeSafetyState, reflects the state of the Life Safety Point object. The means of deriving the Present_Value shall be a local matter. Present_Value may latch non-NORMAL state values until reset.

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.

If a fault algorithm is applied, then this property shall be the pMonitoredValue fault algorithm parameter. See Clause 13.4 for fault algorithm parameter descriptions.

[Change Clause 12.15.12, in the Life Safety Point Object Type, p. 219]

12.15.12 Mode

This writable property, of type BACnetLifeSafetyMode, shall convey the desired operating mode for the ~~Life Safety Point~~ object. ~~The Life Safety Point object shall generate CHANGE_OF_LIFE_SAFETY event notifications for any mode transition if the respective flags TO-OFFNORMAL, TO-FAULT or TO-NORMAL are set in the Event_Enable property (see 12.15.16.1, 12.15.16.2, 12.15.17.1, 12.15.17.2, 12.15.18.1, and 12.15.18.2).~~

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

[Change Clause 12.15.24, in the Life Safety Point Object Type, p. 221]

12.15.24 Operation_Expected

The Operation_Expected property, of type BACnetLifeSafetyOperation, shall specify the next operation expected by this object to handle a specific life safety situation.

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

[Change Clause 12.16.4, in the Life Safety Zone Object Type, p. 224]

12.16.4 Present_Value

This property, of type BACnetLifeSafetyState, reflects the state of the Life Safety Zone object. The means of deriving the Present_Value shall be a local matter. Present_Value may latch non-NORMAL state values until reset.

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.

If a fault algorithm is applied, then this property shall be the pMonitoredValue fault algorithm parameter. See Clause 13.4 for fault algorithm parameter descriptions.

[Change Clause 12.16.12, in the Life Safety Zone Object Type, p. 226]

12.16.4 Mode

This writable property, of type BACnetLifeSafetyMode, shall convey the desired operating mode for the ~~Life Safety Point~~ object. ~~The Life Safety Point object shall generate CHANGE_OF_LIFE_SAFETY event notifications for any mode transition if the respective flags TO-OFFNORMAL, TO-FAULT or TO-NORMAL are set in the Event_Enable property (see 12.16.16.1, 12.16.16.2, 12.16.17.1, 12.16.17.2, 12.16.18.1, and 12.16.18.2).~~

If the object supports event reporting, then this property shall be the pMode parameters of the object's event algorithm. See Clause 13.3 for event algorithm parameter description.

[Change Clause 12.16.24, in the Life Safety Zone Object Type, p. 228]

12.15.24 Operation_Expected

The Operation_Expected property, of type BACnetLifeSafetyOperation, shall specify the next operation expected by this object to handle a specific life safety situation.

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

[Change Clause 12.17.4, in the Loop Object Type, p. 231]

12.17.4 Present_Value

This property indicates the current output value of the loop algorithm in units of the Output_Units property.

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 Clause 12.18.4, in the Multi-state Input Object Type, p. 238]

12.18.4 Present_Value

This property, of type Unsigned, reflects the logical state of the input. The logical state of the input shall be one of 'n' states, where 'n' is the number of states defined in the Number_Of_States property. The means used to determine the current state is a local matter. The Present_Value property shall always have a value greater than zero. The Present_Value property shall be writable when Out_Of_Service is TRUE.

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.

If a fault algorithm is applied, this property shall be the pMonitoredValue fault algorithm parameter. See Clause 13.4 for fault algorithm parameter descriptions.

[Change Clause 12.19.4, in the Multi-state Output Object Type, p. 242]

12.19.4 Present_Value (Commandable)

This property, of type Unsigned, reflects the logical state of an output. The logical state of the output shall be one of 'n' states, where 'n' is the number of states defined in the Number_Of_States property. How the Present_Value is used is a local matter. The Present_Value property shall always have a value greater than zero.

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 Clause 12.20.4, in the Multi-state Value Object Type, p. 247]

12.20.4 Present_Value

This property, of type Unsigned, reflects the logical state of the multi-state value. The logical state of the multi-state value shall be one of 'n' states, where 'n' is the number of states defined in the Number_Of_States property. How the Present_Value is used is a local matter. The Present_Value property shall always have a value greater than zero. Present_Value shall be optionally commandable. If Present_Value is commandable for a given object instance, then the Priority_Array and Relinquish_Default properties shall also be present for that instance. The Present_Value property shall be writable when Out_Of_Service is TRUE.

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.

If a fault algorithm is applied, then this property shall be the pMonitoredValue fault algorithm parameter. See Clause 13.4 for fault algorithm parameter descriptions.

[Change Clause 12.23.5, in the Pulse Converter Object Type, p. 261]

12.23.5 Present_Value

This property, of type REAL, indicates the accumulated value of the input being measured. It is computed by multiplying the current value of the Count property by the value of the Scale_Factor property. The value of the Present_Value property may be adjusted by writing to the Adjust_Value property. The Present_Value property shall be writable when Out_Of_Service is TRUE.

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 Clause 12.25.14, in the Trend Log Object Type, p. 275]

12.25.14 Log_Buffer

…

The buffer is not network accessible except through the use of the ReadRange service, in order to avoid problems with record sequencing when segmentation is required. Attempts to read this property with the ReadProperty-Request or ReadPropertyMultiple-Request shall cause a Result(-) response to be issued, specifying an 'Error Class' of PROPERTY and an 'Error Code' of READ_ACCESS_DENIED.

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

[Change Clause 12.26.20, in the Access Door Object Type, p. 283]

12.26.20 Door_Alarm_State

This ~~optional~~ property, of type BACnetDoorAlarmState, is the alarm state for the physical door *and is restricted to the values NORMAL and those contained in Alarm_Values and Fault_Values.* ~~The CHANGE_OF_STATE event algorithm will monitor this property.~~ When no alarm or fault condition exists for this object, this property shall take on the value NORMAL. It is considered a local matter as to when this property is set to a non-normal value. It is up to the internal control logic to take Lock_Status, Door_Status, Present_Value and information from other objects into account when calculating the proper alarm state. However, this property cannot take on any value which is also in the Masked_Alarm_Values list. If the property is currently set to a specific state and that state is written to the Masked_Alarm_Values list, then the Door_Alarm_State will immediately return to the NORMAL state.

This property ~~is required if intrinsic reporting is supported by this object~~, ~~and~~ if present, *is* required to be writable when Out_Of_Service is TRUE.

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.

If a fault algorithm is applied, then this property shall be the pMonitoredValue fault algorithm parameter. See Clause 13.4 for fault algorithm parameter descriptions.

[Change Clause 12.27.13, in the Event Log Object Type, p. 290]

12.27.13 Log_Buffer

…

The buffer is not network accessible except through the use of the ReadRange service in order to avoid problems with record sequencing when segmentation is required. Attempts to read this property with the ReadProperty-Request or ReadPropertyMultiple-Request shall result in an error specifying an error class of PROPERTY and an error code of READ_ACCESS_DENIED.

If the object supports event reporting, then a reference to this property shall be the pLogBuffer parameter for the object's event algotihm. See Clause 13.3 for event algorithm parameter descriptions.

[Change Clause 12.28.5, in the Load Control Object Type, p. 297]

12.28.5 Present_Value

This property, of type BACnetShedState, indicates the current load shedding state of the object. See Figure 12-5 for a diagram of the state machine governing the value of Present_Value.

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

[Change Clause 12.30.19, in the Trend Log Multiple Object Type, p. 310]

12.30.19 Log_Buffer

…

The Log_Buffer is not network accessible except through the use of the ReadRange service, in order to avoid problems with record sequencing when segmentation is required. Attempts to read this property with the ReadProperty-Request or ReadPropertyMultiple-Request shall cause a Result(-) response to be issued, specifying an 'Error Class' of PROPERTY and an 'Error Code' of READ_ACCESS_DENIED.

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

[Change Clause 12.32.6, p. 330]

12.32.6 Occupancy_State

This property, of type BACnetAccessZoneOccupancyState, reflects the occupancy state of the zone.

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.

BACnetAccessZoneOccupancyState is an enumeration of possible occupancy states.

…

[Change Clause 12.37.5 p. 356]

12.37.5 Present_Value

This property, of type CharacterString, indicates the current value of the object. The Present_Value property shall be writable when Out_Of_Service is TRUE (see Clause 12.37.9).

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.

If a fault algorithm is applied, then this property shall be the pMonitoredValue fault algorithm parameter. See Clause 13.4 for fault algorithm parameter descriptions.

[Change Clause 12.39.5 p. 366]

12.39.5 Present_Value

This property, of type Double, indicates the current value of the object. The Present_Value property shall be writable when Out_Of_Service is TRUE (see Clause 12.39.9).

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

[Change Clause 12.40.5 p. 370]

12.40.5 Present_Value

This property, of type BIT STRING, indicates the current value of the object. The Present_Value property shall be writable when Out_Of_Service is TRUE (see Clause 12.40.10).

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 Clause 12.43.5 p. 381]

12.43.5 Present_Value
This property, of type INTEGER, indicates the current value of the object. The Present_Value property shall be writable when Out_Of_Service is TRUE (see Clause 12.43.9).

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 Clause 12.44.5 p. 386]

12.44.5 Present_Value

This property, of type Unsigned, indicates the current value of the object. The Present_Value property shall be writable when Out_Of_Service is TRUE (see Clause 12.44.9).

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 Clause 12.50.10, and delete sub-clauses, p. 407]

12.50.10 Member_Status_Flags

The Member_Status_Flags property is a logical combination of all the Status_Flags properties contained in the Present_Value. The logical combination means that each of the flags in this property (IN_ALARM, FAULT, OVERRIDDEN, OUT_OF_SERVICE) is TRUE if and only if the corresponding flag is set in any of the Status_Flags property values in the Present_Value property. This property shall be updated whenever new Status_Flags property values are updated in the Present_Value.

If the object supports event reporting, then this property shall be the pMonitoredValue parameter for the object's event algorithm and the pSelectedFlags parameter shall have the IN_ALARM and FAULT bits set and the others cleared. See Clause 13.3 for event algorithm parameter descriptions.

This property is the pMonitoredValue fault algorithm parameter. See Clause 13.4 for fault algorithm parameter descriptions.

~~**12.50.10.1 Conditions for Generating a TO-OFFNORMAL Event**~~

~~...~~

~~**12.50.10.2 Conditions for Generating a TO-NORMAL Event**~~

~~...~~

~~**12.50.10.3 Conditions for Generating a TO-FAULT Event**~~

~~...~~

135-2010*af*-26. Change the Description of the Reliability Property.

Rationale

Allow objects to use any of the Reliability values that are appropriate as defined by the central table of Reliability values in the Clause 12 overview.

For those object types that use are allowed to implement a fault algorithm (as defined in another section of this addendum), the Reliability property is identified as the pCurrentReliability parameter for the algorithm.

[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" *that indicates the existance of fault conditions for the object. Reliability-evaluation is the process of determing the value for this property. The first stage of reliability-evaluation is internal to the object and is completely defined by the device's vendor. The second stage, which is only found in certain object types, is the application of a fault algorithm. See Clause 13.4 for fault algorithm definitions and see the object type definitions to determine the algorithm supported by any particular object type. The different values that the Reliability property can take on are described below. Note that not all values are applicable to all object types.* ~~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.~~

…

[Change all Clause 12.X.Y for Reliability property descriptions in the standard and all addenda by removing the indicated sentence where it occurs, and remove the associated list of values.]
[Note that the sentence, where it occurs, is the same in all occurrences (although the rest of the Reliability clause may differ), while the list of values may differ.]

12.X.Y Reliability

The Reliability property, of type BACnetReliability, provides an indication of whether the Present_Value or the operation of the physical inputs in question are "reliable" as far as the BACnet Device or operator can determine and, if not, why. ~~The Reliability property is required to be present if the Fault_Values property is present. The Reliability property for this object type may have any of the following values:~~

~~{NO_FAULT_DETECTED, ..., UNRELIABLE_OTHER}~~

[The following changes modify the Reliability property descriptions for object types that have a Fault_Values property by deleting the sub-clauses that describe TO_FAULT conditions and by indicating that the Reliability property is the pCurrentReliability parameter.]

[Change Clause 12.15.10, the Life Safety Point object, p. 218]

12.15.10 Reliability

The Reliability property, of type BACnetReliability, provides an indication of whether the Present_Value or the operation of the physical input(s) in question are "reliable" as far as the BACnet Device or operator can determine and, if not, why. ~~The Reliability property for this object type may have any of the following values:~~

~~{NO_FAULT_DETECTED, OPEN_LOOP, SHORTED_LOOP, MULTI_STATE_FAULT, COMMUNICATION_FAILURE, UNRELIABLE_OTHER}~~

If a fault algorithm is applied, then this property shall be the pCurrentReliability parameter for the object's fault algorithm. See Clause 13.4 for fault algorithm parameter descriptions.

~~12.15.10.1 Conditions for Generating a TO-FAULT Event~~

~~A TO-FAULT event is generated under these conditions:~~

~~(a) the Reliability property becomes not equal to NO_FAULT_DETECTED, and~~
~~(b) the TO-FAULT flag is enabled in the Event_Enable property.~~

[Change Clause 12.16.10, the Life Safety Zone object, p. 225]

12.16.10 Reliability

The Reliability property, of type BACnetReliability, provides an indication of whether the Present_Value or the operation of the physical input(s) in question are "reliable" as far as the BACnet Device or operator can determine and, if not, why. ~~The Reliability property for this object type may have any of the following values:~~

~~{NO_FAULT_DETECTED, OPEN_LOOP, SHORTED_LOOP, MULTI_STATE_FAULT, COMMUNICATION_FAILURE, UNRELIABLE_OTHER}~~

If a fault algorithm is applied, then this property shall be the pCurrentReliability parameter for the object's fault algorithm. See Clause 13.4 for fault algorithm parameter descriptions.

~~12.16.10.1 Conditions for Generating a TO-FAULT Event~~

~~A TO-FAULT event is generated under these conditions:~~

~~(a) the Reliability property becomes not equal to NO_FAULT_DETECTED, and~~
~~(b) the TO-FAULT flag is enabled in the Event_Enable property.~~

[Change Clause 12.18.9, the Multi-state Input object, p. 238]

12.18.9 Reliability

The Reliability property, of type BACnetReliability, provides an indication of whether the Present_Value or the operation of the physical inputs in question are "reliable" as far as the BACnet Device or operator can determine and, if not, why. The Reliability property is required to be present if the Fault_Values property is present. ~~The Reliability property for this object type may have any of the following values:~~

~~{NO_FAULT_DETECTED, NO_SENSOR, OVER_RANGE, UNDER_RANGE, OPEN_LOOP, SHORTED_LOOP, MULTI_STATE_FAULT, , COMMUNICATION_FAILURE, UNRELIABLE_OTHER}~~

If a fault algorithm is applied, then this property shall be the pCurrentReliability parameter for the object's fault algorithm. See Clause 13.4 for fault algorithm parameter descriptions.

~~12.18.9.1 Conditions for Generating a TO-FAULT Event~~

~~A TO-FAULT event is generated under these conditions:~~

~~(a) the Reliability property becomes not equal to NO_FAULT_DETECTED, and~~
~~(b) the TO-FAULT flag must be enabled in the Event_Enable property.~~

[Change Clause 12.20.8, the Multi-state Value object, p. 248]

12.20.8 Reliability

The Reliability property, of type BACnetReliability, provides an indication of whether the Present_Value is "reliable" as far as the BACnet Device or operator can determine and, if not, why. The Reliability property is required to be present if the Fault_Values property is present. ~~The Reliability property for this object type may have any of the following values:~~

~~{NO_FAULT_DETECTED, MULTI_STATE_FAULT, COMMUNICATION_FAILURE, UNRELIABLE_OTHER}~~

If a fault algorithm is applied, then this property shall be the pCurrentReliability parameter for the object's fault algorithm. SeeClause 13.4 for fault algorithm parameter descriptions.

~~12.20.8.1 Conditions for Generating a TO-FAULT Event~~

~~A TO-FAULT event is generated under these conditions:~~

~~(a) the Reliability property becomes not equal to NO_FAULT_DETECTED, and~~
~~(b) the TO-FAULT flag must be enabled in the Event_Enable property.~~

[Change Clause 12.26.8, the Access Door object, p. 281]

12.26.8 Reliability

The Reliability property, of type BACnetReliability, provides an indication of whether the Present_Value or the operation of the physical inputs or outputs which comprise this door are "reliable" as far as the BACnet Device or operator can determine and, if not, why. ~~The Reliability property for this object may have any of the following values:~~

~~{NO_FAULT_DETECTED, MULTISTATE_FAULT, CONFIGURATION_ERROR, COMMUNICATION_FAILURE, UNRELIABLE_OTHER}~~

If a fault algorithm is applied, then this property shall be the pCurrentReliability parameter for the object's fault algorithm. See Clause 13.4 for fault algorithm parameter descriptions.

~~12.26.8.1 Conditions for Generating a TO-FAULT Event~~

~~A TO-FAULT event is generated under these conditions:~~

~~(a) the Reliability property becomes not equal to NO_FAULT_DETECTED, and~~
~~(b) the TO-FAULT flag must be enabled in the Event_Enable property.~~

[Change Clause 12.37.8, the CharacterString Value object, p. 358]

12.37.8 Reliability

This ~~optional~~ property, of type BACnetReliability, provides an indication of whether the CharacterString Value object is reliably reporting its value. ~~The Reliability property for this object type may have any of the following values:~~

~~{NO_FAULT_DETECTED, COMMUNICATION_FAILURE, MULTI_STATE_FAULT, UNRELIABLE_OTHER}~~

If a fault algorithm is applied, then this property shall be the pCurrentReliability parameter for the object's fault algorithm. See Clause 13.4 for fault algorithm parameter descriptions.

~~12.37.8.1 Conditions for Generating a TO-FAULT Event~~

~~**A TO-FAULT event is generated under these conditions:**~~

~~**(a) the Reliability property becomes not equal to NO_FAULT_DETECTED, and**~~
~~**(b) the TO-FAULT flag must be enabled in the Event_Enable property.**~~

[Change Clause 12.50.11, the Global Group object, p. 408]

12.50.11 Reliability

This ~~optional~~ property, of type BACnetReliability, provides an indication of whether the Present_Value is "reliable" as far as the BACnet Device or operator can determine. If the FAULT flag of the Member_Status_Flags has a value of TRUE, then the value of this property shall be MEMBER_FAULT. If one or more group member values cannot be updated because of a communication failure, the value of this property shall be COMMUNICATION_FAILURE. If the conditions for a MEMBER_FAULT and a COMMUNICATION_FAILURE are both present, the selection of which value to use is a local matter. ~~The Reliability property for this object type may have any of the following values:~~

~~**{NO_FAULT_DETECTED, MEMBER_FAULT, COMMUNICATION_FAILURE, UNRELIABLE_OTHER}.**~~

If a fault algorithm is applied, then this property shall be the pCurrentReliability parameter for the object's fault algorithm. See Clause 13.4 for fault algorithm parameter descriptions.

135-2010*af*-27. Improve Fault Detection in Event Enrollment Objects.

Rationale

This change improves the fault detection in the Event Enrollment objects so that faults in the monitored object and faults in the Event Enrollment object can be reported and distinguished.

[Note that with this change, the Event Enrollment object will contain a Reliability property and thus the section of this addendum that adds in Reliability_Evaluation_Inhibit will apply to the Event Enrollment object as well.]

[Change Table 12-14, p. 205]

Table 12-14. Properties of the Event Enrollment Object Type

Property Identifier	Property Datatype	Conformance Code
…		
Status_Flags	*BACnetStatusFlags*	*R*
Reliability	*BACnetReliability*	*R*
Fault_Type	*BACnetFaultType*	*O*[x]
Fault_Parameters	*BACnetFaultParameter*	*O*[x]
Profile_Name	CharacterString	O

[x] ***These properties are required if, and shall be present only if, the object supports fault algorithms other than NONE.***

[Add new Clauses 12.12.X..., p. 210]

12.12.X1 Status_Flags

This property, of type BACnetStatusFlags, represents four Boolean flags that indicate the general "health" of an object. Three of the flags are associated with the values of other properties of the object. A more detailed status could be determined by reading the properties that are linked to these flags. The relationship between individual flags is not defined by the protocol. The four flags are

{IN_ALARM, FAULT, OVERRIDDEN, OUT_OF_SERVICE}

where:

IN_ALARM	**Logical FALSE (0) if the Event_State property has a value of NORMAL, otherwise logical TRUE (1).**
FAULT	**Logical TRUE (1) if the Reliability property is present and does not have a value of NO_FAULT_DETECTED, otherwise logical FALSE (0).**
OVERRIDDEN	**Logical FALSE (0).**
OUT_OF_SERVICE	**Logical FALSE (0).**

This property is not the pStatusFlags parameter for the object's event algorithm. The Status_Flags property of the object referenced by the Object_Property_Reference property shall be the pStatusFlags parameter for the event algorithm. See Clause 13.3 for event algorithm parameter descriptions.

12.12.X2 Reliability

This property, of type BACnetReliability, provides an indication of the reliability of the Event Enrollment object to perform its monitoring function in addition to the reliability of the monitored object.

When determining the reliability of an Event Enrollment object, first the reliability of the Event Enrollment object itself shall be checked, followed by the reliability of the monitored object, and finally the reliability conditions are checked by the fault algorithm, if one is in use. If one of these evaluations encounters a value other than NO_FAULT_DETECTED, the sequence of evaluations shall be stopped and that reliability value shall be stored in the Reliability property.

12.12.X3 Fault_Type

This read-only property, of type BACnetFaultType, indicates the type of fault algorithm that is applied by the object.

There is a specific relationship between each fault algorithm, the fault algorithm parameters, and the fault type. The Fault_Type is determined by the chosen fault parameters in the Fault_Parameters property and reflects the fault algorithm that is used to monitor the referenced object for fault conditions. The fault algorithm for each Fault_Type is specified in Clause 13.4. The Fault_Parameters property provides the parameters needed by the algorithm.

12.12.X4 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-X.

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-X. 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

[Change Clause 21, p 615]

```
BACnetPropertyIdentifier ::= ENUMERATED {
   ...
   failed-attempts-time          (274),
   fault-parameters              (358),
   fault-type                    (359),
   ...
   -- see event-message-texts    (351),
   -- see fault-parameters       (358),
   -- see fault-type             (359),
   ...
```

}

[Change Clause 21, add new productions, p 603]

```
BACnetFaultParameter ::= CHOICE {
    none                [0] NULL,
    fault-characterstring   [1] SEQUENCE {
                            list-of-fault-values [0] SEQUENCE OF CharacterString
                            },
    fault-extended      [2] SEQUENCE {
                            vendor-id       [0] Unsigned16,
                            extended-fault-type   [1] Unsigned,
                            parameters      [2] SEQUENCE OF CHOICE {
                                null            NULL,
                                real            REAL,
                                unsigned        Unsigned,
                                boolean         BOOLEAN,
                                integer         INTEGER,
                                double              Double,
                                octet           OCTET STRING,
                                characterString CharacterString,
                                bitstring       BIT STRING,
                                enum            ENUMERATED,
                                date            Date,
                                time            Time,
                                objectIdentifier BACnetObjectIdentifier,
                                reference [0] BACnetDeviceObjectPropertyReference
                                }
                            },
    fault-life-safety   [3] SEQUENCE {
                            list-of-fault-values [0] SEQUENCE OF BACnetLifeSafetyState,
                            mode-property-reference [1] BACnetDeviceObjectPropertyReference
                            },
    fault-state         [4] SEQUENCE {
                            list-of-fault-values [0] SEQUENCE OF BACnetPropertyStates
                            },
    fault-status-flags  [5] SEQUENCE {
                            status-flags-reference [0]  BACnetDeviceObjectPropertyReference
                            }
    }

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

135-2010*af*-28. Add Ability for some Objects Types to Send Only Fault Notifications.

Rationale

The Schedule, Program, and Credential Data Input object types are able to detect faults but are not currently able to send notifications indicating the faults. This change adds the ability for those objects to do so.

Note that there are other access control objects that are able to detect fault condition but are unable to send notifications for them. In these access control objects, the Reliability property is only used to warn when a configuration error has occurred. This will generally occur due to an operator error and so methods other than sending out a fault notification are more appropriate to signal a configuration error. In addition, the extra properties required to support fault notification (i.e., Event_Detection_Enable, Notification_Class, Acked_Transitions, etc.) are too heavyweight for the Access Rights and Access Credential Input object types because there may be thousands of these objects on a single device.

[Change Clause 12.22, p. 254]

12.22 Program Object Type

…

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

The Program object type and its standardized properties are summarized in Table 12-26 and described in detail in this subclause.

[Add entries to Table 12-26, p. 254]

Table 12-26. Properties of the Program Object Type

. . .	. . .	. . .
Out_Of_Service	BOOLEAN	R
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 12.22.X..., p. 258]
[Note that the text for the body of the added clauses is specified by other sections of this addendum]

12.22.X1 Event_Detection_Enable

12.22.X2 Notification_Class

12.22.X3 Event_Enable

12.22.X4 Acked_Transitions

12.22.X5 Notify_Type

12.22.X6 Event_Time_Stamps

12.22.X7 Event_Message_Texts

12.22.X8 Event_Message_Texts_Config

[Change Clause 12.24, p. 266]

12.24 Schedule Object Type

…

Versions of the Schedule object prior to Protocol_Revision 4 only support schedules that define an entire day, from midnight to midnight. For compatibility with these versions, this whole day behavior can be achieved by using a specific schedule format. Weekly_Schedule and Exception_Schedule values that begin at 00:00, and do not use any NULL values, will define schedules for the entire day. Property values in this format will produce the same results in all versions of the Schedule object.

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

[Add entries to Table 12-28, p. 266]

Table 12-28. Properties of the Schedule Object Type

. . .	. . .	. . .
Out_Of_Service	**BOOLEAN**	**R**
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 12.24.X..., p. 270]
[Note that the text for the body of the added clauses is specified by other sections of this addendum]

12.24.X1 Event_Detection_Enable

12.24.X2 Notification_Class

12.24.X3 Event_Enable

12.24.X4 Acked_Transitions

12.24.X5 Notify_Type

12.24.X6 Event_Time_Stamps

12.24.X7 Event_Message_Texts

12.24.X8 Event_Message_Texts_Config

[Change Clause 12.36 p. 353]

12.36 Credential Data Input Object Type

…

Credential Data Input objects may optionally support intrinsic reporting to facilitate the reporting of fault conditions. Credential Data Input objects that support intrinsic reporting shall apply the NONE event algorithm.

The Credential Data Input object type and its properties are summarized in Table 12-43 and described in detail in this subclause.

[Add entries to Table 12-43, p. 353]

Table 12-43. Properties of the Credential Data Input Object Type

. . .	. . .	. . .
Update_Time	**BACnetTimeStamp**	**R**
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 12.36.X..., , p. 356]
[Note that the text for the body of the added clauses is specified by other sections of this addendum]

12.36.X1 Event_Detection_Enable

12.36.X2 Notification_Class

12.36.X3 Event_Enable

12.36.X4 Acked_Transitions

12.36.X5 Notify_Type

12.36.X6 Event_Time_Stamps

12.36.X7 Event_Message_Texts

12.36.X8 Event_Message_Texts_Config

135-2010*af*-29. Add a Notification Forwarder Object Type.

Rationale

The Notification Forwarder object type is meant to fulfill a number of goals:
a) allow many notification-clients to subscribe for notifications while allowing simple devices to limit their number of subscribers.
b) reduce the number of objects that notification-clients need to subscribe to in order to subscribe to all alarms in the system.
c) allow for dynamic subscriptions for event notifications (in contrast to hard-configured recipients).

These changes are based on the assumption that in most BACnet deployments, notification-clients are interested in all alarms at all times, and that in some markets the connection between notification-servers and notification-clients is dynamic whereas in other markets it is not.

[Add new Clause 12.X, p. 410]

12.X Notification Forwarder Object Type

The Notification Forwarder object type defines a standardized object whose properties represent the externally visible characteristics required for the re-distribution of event notifications to zero or more destinations. It differs from a Notification Class in that the Notification Forwarder object is not used for originating event notifications, but rather is used to forward event notifications to a different and potentially larger number of recipients.

The Notification Forwarder allows devices that can distribute notifications to a small number of destinations to have their notifications received by many destinations. It also allows for a reduction in the number of objects that need to be modified in order to change the set of event destinations for a large number of devices. Notification Forwarder objects can also be restricted to forward only locally generated notifications as indicated by the Local_Forwarding_Only property. In doing so, the Notification Forwarder object allows for the use of recipient subscriptions and the centralizion of recipients for multiple Notification Classes.

A Notification Forwarder object's Process_Identifier_Filter value is used in the selection of event notification service requests that are to be forwarded by the object. If multiple Notification Forwarder objects exist within the device, any received event notification shall be passed to each Notification Forwarder object internally by the device if the Process_Identifier_Filter and other restrictions allow acceptance by the object. Event notifications generated by local event-initiating objects are only passed to local Notification Forwarder objects if the Notification Class object has the local device as a recipient.

The following restrictions are intended to reduce the likelihood of an accidental endless cycle of event forwarding for the same notification or of accidental duplicated notifications to the same device.

a) Any event notification received on a particular port of the device containing Notification Forwarder objects shall not be forwarded as a local broadcast to the BACnet network directly attached to that port.

b) Any event notification received as a global broadcast shall be ignored by Notification Forwarder objects in receiving devices.

c) The Notification Forwarder object shall not send any forwarded notification using a global broadcast.

d) Any event notification received as a broadcast to a particular BACnet network shall not be forwarded to any device resident on that same network.

e) Any event notification received on a particular port of the device containing Notification Forwarder objects shall be ignored by any Notification Forwarder object within the device that does not have the receiving port enabled within its Port_Filter property. In order to stop multiple notifications from being forwarded to the notification-clients, there should be at most one Notification Forwarder object on a network that will forward broadcast notifications. The Port_Filter allows a site to be configured this way when the Notification Forwarder objects are located in BACnet routers.

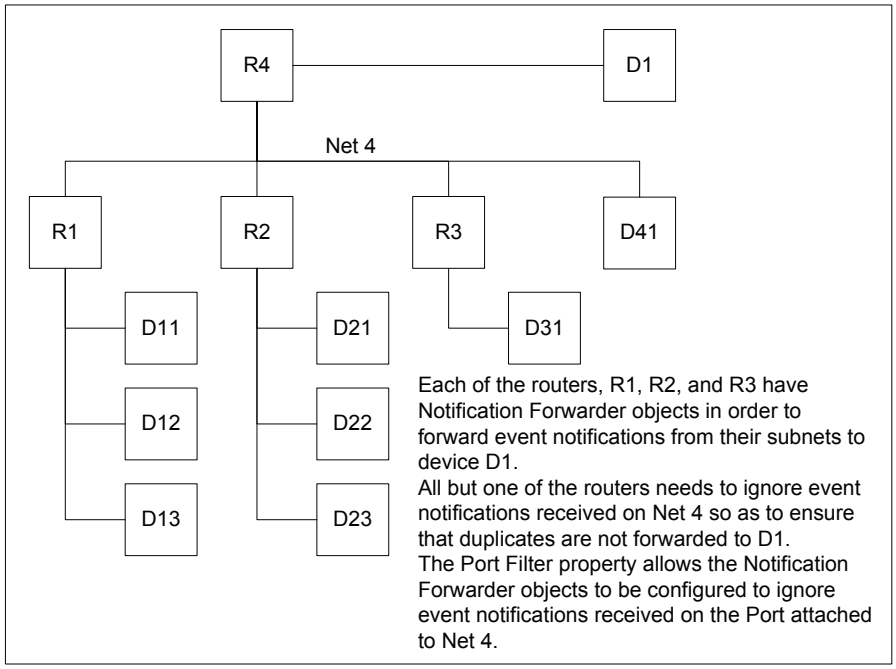

Figure 12-X1. General Use Case for the Port_Filter property.

Notification Forwarder objects can be logically chained together one after another in the same or different devices.

An event notification is sent through a Notification Forwarder object in the same device by specifying the local Device object in the Notification Class object's Recipient_List along with the Process_Identifier_Filter matching the Notification Forwarding object.

Like Notification Class objects, the Notification Forwarder object allows for date, time and transition filters for destinations in the Recipient_List property. The filtering works in the same manner as the Notification Class object. In order to allow central configuration of the date, time and transition filters, Recipient_List entries that direct event notifications to Notification Forwarder objects are expected to have the filtering parameters set to send all transitions, on all days, at all times so that filtering is performed only by the Notification Forwarder object.

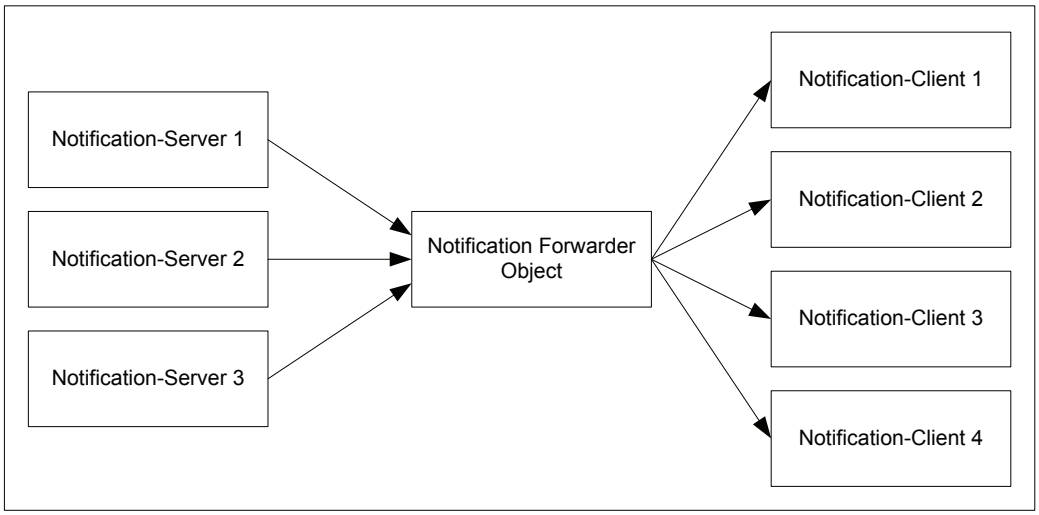

Figure 12-X2. Example of Multiple Notification-Servers Routing to Multiple Notification-Clients.

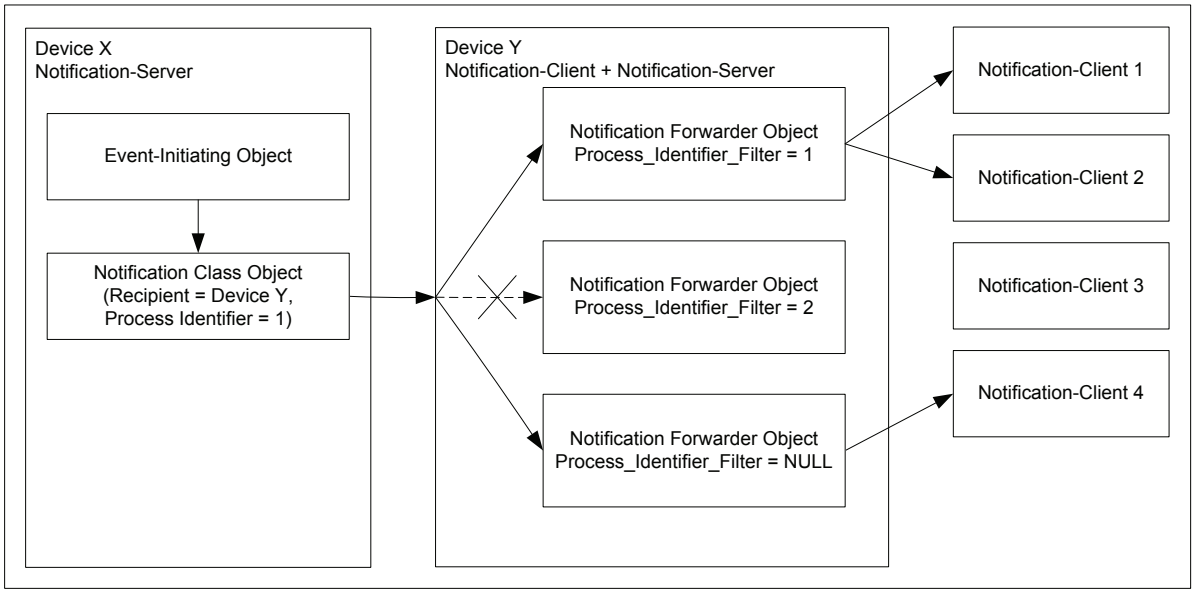

Figure 12-X3. Example of Forwarding a Single-Event Notification.

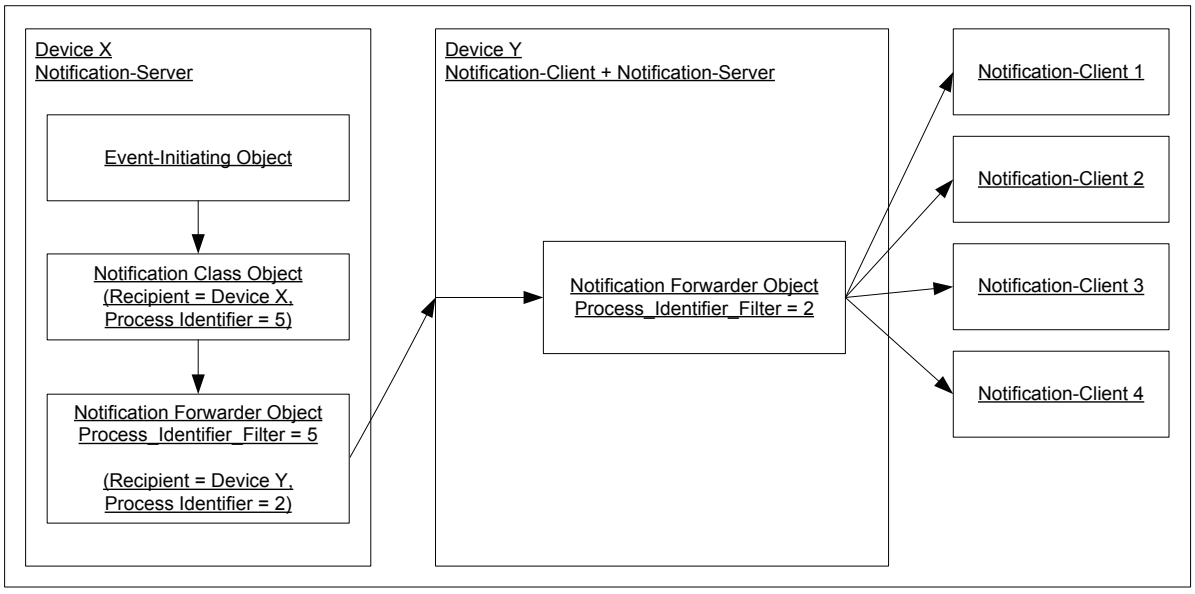

Figure 12-X4. Example of Local Forwarding and Multiple-Step Forwarding.

Notification Forwarders that are forwarding for other devices shall be capable of forwarding both ConfirmedEventNotification and UnconfirmedEventNotification services, and shall be capable of forwarding them using either service regardless of which is received.

Notification Forwarder objects shall forward event notifications regardless of the character set of any text in the event notification.

To forward an event notification to a BACnetDestination, the source notification shall have the Process Identifier parameter changed to match that of the BACnetDestination, and the notification shall be sent confirmed or unconfirmed as specified by the BACnetDestination. The notification shall then be sent to the recipient indicated by the BACnetDestination.

When acknowledging an event notification that has been forwarded by a Notification Forwarder, the acknowledging device shall send the AcknowledgeAlarm service request directly to the device indicated by the Initiating Device Identifier parameter of the event notification.

Table 12-X1. Properties of the Notification Forwarder Object Type

Property Identifier	Property Datatype	Conformance Code
Object_Identifier	BACnetObjectIdentifier	R
Object_Name	CharacterString	R
Object_Type	BACnetObjectType	R
Description	CharacterString	O
Status_Flags	BACnetStatusFlags	R
Reliability	BACnetReliability	R
Out_Of_Service	BOOLEAN	R
Recipient_List	List of Destination	R
Subscribed_Recipients	List of BACnetEventNotificationSubscription	W
Process_Identifier_Filter	BACnetProcessIdSelection	R
Port_Filter	BACnetARRAY[N] of BACnetPortPermission	O[1]
Local_Forwarding_Only	BOOLEAN	R
Profile_Name	CharacterString	O

[1] This property is required if the device includes BACnet router functionality.

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 NOTIFICATION_FORWARDER.

12.X.4 Description

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

12.X.5 Status_Flags

This property, of type BACnetStatusFlags, represents four Boolean flags that indicate the general "health" of a Notification Forwarder object. The OUT_OF_SERVICE flag is associated with the value of another property of this object. The relationship between individual flags is not defined by the protocol. The four flags are:

{IN_ALARM, FAULT, OVERRIDDEN, OUT_OF_SERVICE}

where:

IN_ALARM	**The value of this flag shall be Logical FALSE (0).**
FAULT	**Logical TRUE (1) if the Reliability property ~~is present and~~ does not have a value of NO_FAULT_DETECTED, otherwise logical FALSE (0).**
OVERRIDDEN	**The value of this flag shall be Logical FALSE (0).**
OUT_OF_SERVICE	**Logical TRUE (1) if the Out_Of_Service property has a value of TRUE, otherwise logical FALSE (0).**

12.X.6 Reliability

The Reliability property, of type BACnetReliability, provides an indication of whether the object is configured and operating. The value of Reliability shall not indicate that a subset of the possible recipients (Recipient_List, and Subscribed_Recipients) are not reachable, but may indicate that all possible recipients are unreachable or are undefined.

12.X.7 Out_Of_Service

The Out_Of_Service property, of type BOOLEAN is an indication whether (TRUE) or not (FALSE) the object has been prevented from forwarding event notifications. This property can be used to disable the Notification Forwarder object.

12.X.8 Recipient_List

This property, of type List of BACnetDestination, shall convey a list of 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.

12.X.9 Subscribed_Recipients

This property, of type List of BACnetEventNotificationSubscription, conveys a list of recipient destinations to which event notifications are sent when events are forwarded by the Notification Forwarder object. These recipient destinations are intended to be temporary, and will expire if not renewed.

To add, remove, renew or modify a subscription, the AddListElement or RemoveListElement services are used. When comparing entries to those provided by a list service, an entry in this property shall be considered a match when the Recipient fields are equal and Process Identifier fields are equal. Each entry in the list is a structure of parameters that is described in Table 12-X2.

Table 12-X2. Components of a BACnetEventNotificationSubscription

Parameter	Type	Description
Recipient	BACnetRecipient	The destination device(s) to receive notifications.
Process Identifier	Unsigned32	The handle of a process within the recipient device that is to receive the event notification.
Issue Confirmed Notifications	Boolean	(TRUE) if confirmed notifications are to be sent and (FALSE) if unconfirmed notifications are to be sent.
Time Remaining	Unsigned	Actual time the entry will remain in the Subscribed_Recipients in seconds.

The Time Remaining field of a BACnetEventNotificationSubscription entry, of type Unsigned, indicates the remaining lifetime of the subscription in minutes. An entry shall be removed from the list when the lifetime reaches zero, and therefore no entries in the property shall have a Time Remaining value of zero. Notification Forwarder objects shall accept subscriptions with Time Remaining values in the range of 1 through 1440 (24 hours). It is a local matter whether or not a Notification Forwarder accepts larger Time Remaining values.

When renewing an existing subscription, the values of all fields provided in the AddListElement service shall replace the values of all fields of the existing subscription.

The Subscribed_Recipients shall be maintained through a power failure or device "restart." After the restart, the Time Remaining may be any value between the value before the restart and the value provided in the entry's last subscription operation.

12.X.10 Process_Identifier_Filter

This property, of type BACnetProcessIdSelection, is used in the selection of event notification service requests that are to be forwarded by the object. When the Process Identifier parameter of a received event notification is the same as the value of the Process_Identifier_Filter property, or if the Process_Identifier_Filter property contains a NULL, then the notification will be accepted for forwarding by the Notification Forwarder object subject to the port, network and broadcast restrictions.

12.X.11 Port_Filter

This property, of type BACnetArray of BACnetPortPermission, enables or disables the forwarding of event notifications received on a particular network port. When an event notification is received on a port that is marked as disabled by this property, the Notification Forwarder object shall ignore that event notification.

If present, this property shall be writable. If not present, then the device is not a router and its only configured port shall be enabled for event notification forwarding.

Neither the size of the array nor the Port_ID portion of the BACnetPortPermission entries shall be modifiable via writes to this property.

The number of entries in the array shall match the number of BACnet ports currently defined in the device.

The BACnetPortPermission entries themselves define a structure of parameters that is summarized in Table 12-X3.

Table 12-X3. Components of a BACnetPortPermission

PARAMETER	TYPE	DESCRIPTION
Port_ID	Unsigned8	The Port_ID parameter shall correspond to the Port ID of the associated network as described in Clause 6. For non-routing nodes, this value shall be 0.
Enabled	Boolean	Indicates whether forwarding is enabled (TRUE) or not (FALSE) for event notifications received through the corresponding port.

12.X.12 Local_Forwarding_Only

This property, of type BOOLEAN is an indication whether (TRUE) or not (FALSE) the object is limited to forwarding notifications initiated from within the same device. If Local_Forwarding_Only has a value of FALSE, then the Notification Forwarder is capable of forwarding notifications for other devices.

12.X.13 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 is not required to 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 Clause 15.1.2, the AddListElement Service Procedure, p 468]

15.1.2 Service Procedure

After verifying the validity of the request, the responding BACnet-user shall attempt to modify the object identified in the 'Object Identifier' parameter. If the identified object exists and has the property specified in the 'Property Identifier' parameter, an attempt shall be made to add all of the elements specified in the 'List of Elements' parameter to the specified property. If this attempt is successful, a 'Result(+)' primitive shall be issued.

When comparing elements in the List of Elements with elements in the specified property, the complete element shall be compared unless the property description specifies otherwise. **If one or more of the elements is already present in the list, it shall be *updated with the provided element, that is, the existing element is over-written with the provided element. Optionally, if the provided element is exactly the same as the existing element in every way, it may be* ignored, that is, ~~shall~~ not ~~be~~ added to the list. Ignoring an element that already exists shall not cause the service to fail.**

If the specified object does not exist, the specified property does not exist, or the specified property is not a list, then the service shall fail and a 'Result(-)' response primitive shall be issued. If one or more elements cannot be added to*, or updated in,* the list~~ and they are not already members~~, a 'Result(-)' response primitive shall be issued and no elements shall be added to*, or updated in,* the list.

The effect of this service shall be to add to*, or update in,* the list all of the specified elements ~~that are not already present~~ or to *neither* add ~~no~~ *nor update any* elements ~~to the list~~ at all.

[Change Clause 15.2.2, the RemoveListElement Service Procedure, p 470]

15.2.2 Service Procedure

After verifying the validity of the request, the responding BACnet-user shall attempt to modify the object identified in the 'Object Identifier' parameter. If the identified object exists and it has the property specified in the 'Property Identifier' parameter, an attempt shall be made to remove the elements in the 'List of Elements' from the property of the object.

When comparing elements of the service with entries in the affected list, the complete element shall be compared unless the property description specifies otherwise. **If one or more of the elements does not exist or cannot be removed because of insufficient authority, none of the elements shall be removed and a 'Result(-)' response primitive shall be issued.**

[Change Clause 21, add new productions, p. 600]

```
BACnetEventNotificationSubscription ::= SEQUENCE {
   recipient                    [0] BACnetRecipient,
   processIdentifier                [1] Unsigned32,
   issueConfirmedNotifications      [2] BOOLEAN,
   timeRemaining                [3] Unsigned
}

BACnetPortPermission ::= SEQUENCE {
   port-id                          [0] Unsigned8,
   enabled                      [1] BOOLEAN
}

BACnetProcessIdSelection ::= CHOICE {
   processIdentifier                Unsigned32,
   nullValue                    NULL
}
```

[Change Clause 21, BACnetObjectType production, p 610]

```
BACnetObjectType ::= ENUMERATED {
…
      notification-class            (15),
      notification-forwarder            (51),

…
   -- see time-value               (50),
   -- see notification-forwarder        (51),
...
}
```

[Change Clause 21, BACnetObjectTypesSupported production, p 464]

```
BACnetObjectTypesSupported ::= BIT STRING {
…
   time-value               (50),
   notification-forwarder   (51),
   }
```

[Change Clause 21, BACnetPropertyIdentifier production, p 617]

```
BACnetPropertyIdentifier ::= ENUMERATED {
…
   local-date                   (56),
   local-forwarding-only        (360),
```

…
polarity (84),
port-filter (363),
...
process-identifier (89),
process-identifier-filter (361),
…
subordinate-list (211),
subscribed-recipients (362),
…
-- see event-message-texts (351),
-- local-forwarding-only (360),
-- process-identifier-filter (361),
-- subscribed-recipients (362),
...
}

[Add new Clauses K.2.X1 and K.2.X2, p 863]

K.2.X1 BIBB - Alarm and Event-Notification Forwarder-B (AE-NF-B)

Device B forwards alarm and event notifications for other devices.

BACnet Service	Initiate	Execute
ConfirmedEventNotification	x	x
UnconfirmedEventNotification	x	x
AddListElement		x
RemoveListElement		x

Devices claiming conformance to AE-NF-B shall support the Notification Forwarder object type with configurable Recipient_List properties that can contain at least 8 entries, and with a Subscribed_Recipients property with at least 8 entries.

The Notification Forwarder objects shall be capable of forwarding all event notifications received by the device. If the Process_Identifier_Filter properties of the Notification Forwarder objects are not configurable, then at least one Notification Forwarder object shall have a Process_Identifier_Filter property with a value of NULL or 0 in order to forward notifications from devices that only send event notifications using local broadcasts.

K.2.X2 BIBB - Alarm and Event-Notification Forwarder-Internal-B (AE-NF-I-B)

Device B forwards alarm and event notifications for the local device.

BACnet Service	Initiate	Execute
ConfirmedEventNotification	x	
UnconfirmedEventNotification	x	
AddListElement		x
RemoveListElement		x

Devices claiming conformance to AE-NF-I-B shall support the Notification Forwarder object type with configurable Recipient_List properties that can contain at least 8 entries, and with a Subscribed_Recipients property with at least 8 entries.

135-2010*af*-30. Reduce the Requirements on Notification-Servers.

Rationale

In order to reduce the level of effort required to implement basic event reporting, the notification distribution requirements are reduced. This change works in concert with the addition of the Notification Forwarder object that is added in a different section of this addendum.

This change also allows a large number of Notification_Class objects to share a small number of sets of recipients by allowing a Notification_Class to defer the recipient information to local Notification Forwarder objects.

[Change Clause 12.21.8, p. 252]

12.21.8 Recipient_List

This property, of type List of BACnetDestination, shall convey a list of one or more recipient destinations to which notifications shall be sent when event-initiating objects using this class detect the occurrence of an event. *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.

Table 12-25. Components of a BACnetDestination

Parameter	Type	Description
...	...	...

When combined with local Notification Forwarder objects, a device is able to contain a large number of Notification Class objects while centralizing the Recipient_List information in a small number of local Notification Forwarder objects. If local Notification Forwarder objects are being used to specify all of the recipients for the Notification Class, the Recipient_List property is allowed to be read-only. In this case, the read-only Recipient_List shall contain at least one entry, and all entries in the Recipient_List shall refer to the local device.

If the Recipient_List is not writable and the device is not using local Notification Forwarder objects, then the Recipient_List shall have a length of 1 with the Recipient set to a local broadcast, Valid Days set to all days, From Time set to 00:00:00.0, To_Time set to 23:59:59.99, Process Identifier set to 0, Issue Confirmed Notifications set to FALSE, and all bits in Transitions set to TRUE. When deploying devices configured in this manner, it is expected that a Notification Forwarder will be installed in a different device on the same BACnet network to forward the notifications to recipients that are not on the local network. Note that this implementation choice should not be chosen for devices on datalinks that are severly impacted by broadcasts.

For writable Recipient_List properties, devices are allowed to restrict the values for the Valid Days, From Time, To_Time, and the Transitions field such that they only accept the configuration that results in all transitions being sent without regard to the current time or date. In such cases, the Valid Days shall be all days, From Time shall be 0:00:00.0, To Time shall be 23:59:59.99, and Transitions shall be (TRUE, TRUE, TRUE). A device shall not otherwise restrict the value of Recipient_List entries.

[Change Clause K.2.2, p 857]

K.2.2 BIBB - Alarm and Event-Notification Internal-B (AE-N-I-B)

Device B generates notifications about alarms and other events.

BACnet Service	Initiate	Execute
ConfirmedEventNotification	x[1]	
UnconfirmedEventNotification	x	

[1]Devices that contain only read-only Recipient_List properties and no Notification Forwarder objects are not required to have the capability to initiate ConfirmedEventNotification requests.

Devices claiming conformance to AE-N-I-B shall also support either Intrinsic or Algorithmic reporting. Any device that supports the generation of event notifications that require operator acknowledgment shall support AE-ACK-B and AE-INFO-B. Any device that supports the generation of TO-FAULT or TO-OFFNORMAL event notifications shall support AE-INFO-B.

Devices that only support generation of CHANGE_OF_LIFE_SAFETY and/or BUFFER_READY notifications shall not claim support for this BIBB.

[Change Clause K.2.3, p 857]

K.2.3 BIBB - Alarm and Event-Notification External-B (AE-N-E-B)

Device B contains an Event Enrollment object that monitors values in another device. Device B is capable of generating event notifications for alarm conditions based on value(s) in another device. Devices conforming to this BIBB shall conform to DS-RP-A, AE-N-I-B, and shall support at least 1 Event Enrollment object with an Object_Property_Reference property that supports references to properties in objects contained in other devices. Any device that supports the generation of event notifications that require operator acknowledgment shall support AE-ACK-B and AE-INFO-B. Any device that supports the generation of TO-FAULT or TO-OFFNORMAL event notifications shall support AE-INFO-B.

Devices claiming support for this BIBB shall either support writable Recipient_List properties or support Notification_Forwarder objects.

Devices that only support Event Enrollment objects that only support generation of CHANGE_OF_LIFE_SAFETY and/or BUFFER_READY notifications shall not claim support for this BIBB.

135-2010*af*-31. Add an Alert Enrollment Object Type.

Rationale

This change allows objects to indicate the occurrence of interesting events that are unrelated to the object's event algorithm.

[Add Clause 12.X, p. 410]

12.X Alert Enrollment Object Type

The Alert Enrollment object type defines a standardized object that represents and contains the information required for managing information alerts from a BACnet device. "Information alerts" are interesting notifications that are not related to algorithmic or intrinsic reporting of an object. The Alert Enrollment object allows these alerts to be generated without impacting the Event_State of the object to which the alerts are related.

Alerts are always distributed using ConfirmedEventNotification or UnconfirmedEventNotification services with 'To State' and 'From State' set to NORMAL and an 'Event Type' of EXTENDED. The values used in the 'Vendor Id' and 'Extended Event Type' parameters allow for classification of alerts. The choice of values used in the 'Vendor Id' and 'Extended Event Type' parameters is a local matter. The definition of the parameters used with any pariticular pair of 'Vendor Id' and 'Extended Event Type' is controlled by the registered owner of the 'Vendor Id' value. The extended notification parameters allow for a vendor-specified set of values to be provided with the notification. For the notification of alerts, the first extended notification parameter shall be a BACnetObjectIdentifier that identifies the source of the alert (not the Alert Enrollment object, but the object that provided the alert to the Alert Enrollment object). If an alert is not logically associated with a specific object, the local Device object shall be referenced as the source of the alert.

When there are multiple alert enrollment objects in a device, the method used to associate an Alert Enrollment object to any particular alert generated by an object is a local matter.

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

Table 12-X. Properties of the Alert Enrollment Object Type

Property Identifier	Property Datatype	Conformance Code
Object_Identifier	BACnetObjectIdentifier	R
Object_Name	CharacterString	R
Object_Type	BACnetObjectType	R
Description	CharacterString	O
Present_Value	BACnetObjectIdentifier	R
Event_State	BACnetEventState	R
Event_Detection_Enable	BOOLEAN	R
Notification_Class	Unsigned	R
Event_Enable	BACnetEventTransitionBits	R
Acked_Transitions	BACnetEventTransitionBits	R
Notify_Type	BACnetNotifyType	R
Event_Time_Stamps	BACnetARRAY[3] of BACnetTimeStamp	R
Event_Message_Texts	BACnetARRAY[3] of CharacterString	O
Event_Message_Texts_Config	BACnetARRAY[3] of CharacterString	O
Event_Algorithm_Inhibit_Ref	BACnetObjectPropertyReference	O
Event_Algorithm_Inhibit	BOOLEAN	O

Profile_Name	**CharacterString**	**O**

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 ALERT_ENROLLMENT.

12.X.4 Description

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

12.X.5 Present_Value

This read-only property, of type BACnetObjectIdentifier, indicates the object that last provided an alert to this object for notification.

[Note that the text for the body of Clauses 12.X.6...12.X.15 is specified by other sections of this addendum.]

12.X.6 Event_State

12.X.7 Event_Detection_Enable

12.X.8 Notification_Class

12.X.9 Event_Enable

12.X.10 Acked_Transitions

12.X.11 Notify_Type

12.X.12 Event_Time_Stamps

12.X.13 Event_Message_Texts

12.X.14 Event_Message_Texts_Config

12.X.15 Event_Algorithm_Inhibit_Ref

12.X.16 Event_Algorithm_Inhibit

12.X.17 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 Clause 21, BACnetEventParameter production, p. 601]

BACnetEventParameter ::= CHOICE {
...
extended [9] SEQUENCE {
vendor-id [0] Unsigned16,
extended-event-type [1] Unsigned,
parameters [2] SEQUENCE OF CHOICE {
null NULL,
real REAL,
~~integer~~*unsigned* Unsigned,
boolean BOOLEAN,
***integer* *INTEGER*,**
double Double,
octet OCTET STRING,
***characterString* *CharacterString*,**
bitstring BIT STRING,
enum ENUMERATED,
***date* *Date*,**
***time* *Time*,**
***objectIdentifier* *BACnetObjectIdentifier*,**
reference [0] BACnetDeviceObjectPropertyReference
}
},
...
}

[Change Clause 21, BACnetNotificationParameters productions, p 607]

BACnetNotificationParameters ::= CHOICE {
...

extended [9] SEQUENCE {
vendor-id [0] Unsigned16,
extended-event-type [1] Unsigned,
parameters [2] SEQUENCE OF CHOICE {
null NULL,
real REAL,
~~integer~~*unsigned* Unsigned,
boolean BOOLEAN,
***integer* *INTEGER*,**
double Double,
octet OCTET STRING,
***characterString* *CharacterString*,**
bitstring BIT STRING,
enum ENUMERATED,
***date* *Date*,**
***time* *Time*,**
***objectIdentifier* *BACnetObjectIdentifier*,**
propertyValue [0] BACnetDeviceObjectPropertyValue
}
},

[Change Clause 21, BACnetObjectType production, p 609]

BACnetObjectType ::= ENUMERATED {
***alert-enrollment* *(52)*,**
access-credential (32),
...
-- see time-value (50),

-- see alert-enrollment (52),
...
}

[Change Clause 21, BACnetObjectTypesSupported production, p 611]

BACnetObjectTypesSupported ::= BIT STRING {
time-vlaue (50),
alert-enrollment (52)
}

135-2010*af*-32. Improve the Specification of Event Reporting.

Rationale

The purpose of this change is to ensure that event reporting is specified consistently throughout the standard and to make intrinsic reporting and algorithmic reporting behave consistently. A number of changes are made both to the manner in which event reporting is specified and to the way in which event reporting works. It also:

- **Adds in an overview of the general alarming model using a conceptual state machine. This is intended to describe the general model separate from the individual event algorithms.**
- **Improves the event algorithm descriptions and use the same descriptions for both intrinsic reporting and algorithmic reporting.**
- **Changes the method by which transitions to and from the fault state are reported so as to provide parameters that are appropriate for fault transitions. This is achieved through the addition of the CHANGE_OF_RELIABILITY event type. Also, objects are now allowed to report fault to fault transitions when a fault condition changes.**
- **Clearly separates the detection of the Reliability of an object and the reporting of changes in the Reliability of an object.**
- **Changes the OUT_OF_RANGE (all forms) and FLOATING_LIMIT event algorithms to allow implementations to optionally transition from HIGH_LIMIT to LOW_LIMIT and vice versa.**
- **Changes CHANGE_OF_STATE algorithm to optionally allow OFFNORMAL to OFFNORMAL transitions.**
- **Changes the CHANGE_OF_LIFE_SAFETY algorithms to optionally allow OFFNORMAL to OFFNORMAL and LIFE_SAFETY_ALARM to LIFE_SAFETY_ALARM transitions.**
- **Clarifies that Limit_Enable enables / disables the checking of the HIGH_LIMIT and/or LOW_LIMIT conditions and does not just enable / disable the reporting of HIGH_LIMIT and/or LOW_LIMIT events.**

[Add new entries to Clauses 3.2, alphabetically, p. 2]

alarm-acknowledgement: the process of indicating that a human operator has seen and responded to an event notification.

event algorithm: the rules that determine when an event-initiating object changes between normal and offnormal states. The event algorithm has no impact on an event-initiating object's transition to or from fault.

event-initiating object: an object that is configured to monitor its event state and can report changes in its event state.

event-notification-distribution: the process that a notification-server performs in the determination of notification-clients and in the sending of notifications to notification-clients when an event-initiating object changes the event or acknowledgement state.

event notification message: a ConfirmedEventNotification or UnconfirmedEventNotification service request used to indicate a change in the event or acknowledgement state of an event-initiating object.

event-state-detection: the process of executing an event-initiating object's event algorithm and monitoring the object's Reliability property to detect changes in the object's event state.

event-summarization: the querying of event-intiating objects in a device through one of the event summarization services to determine those that meet specific event state or reporting conditions.

notification-client: a BACnet device that receives and processes event notification messages.

notification-server: a BACnet device that contains event-initiating objects and performs event notifcation distribution.

reliability-evaluation: the process by which an object determines its reliability and thus the value to set into its Reliability property.

[Change Clause 12, p 146]

12 MODELING CONTROL DEVICES AS A COLLECTION OF OBJECTS

...

NO_FAULT_DETECTED	**The present value is reliable; that is, no other fault (enumerated below) has been detected.**
NO_SENSOR	**No sensor is connected to the Input object.**
OVER_RANGE	**The sensor connected to the Input is reading a value higher than the normal operating range. If the object is a Binary Input, this is possible when the Binary state is derived from an analog sensor or a binary input equipped with electrical loop supervision circuits.**
UNDER_RANGE	**The sensor connected to the Input is reading a value lower than the normal operating range. If the object is a Binary Input, this is possible when the Binary Input is actually a binary state calculated from an analog sensor.**
OPEN_LOOP	**The connection between the defined object and the physical device is providing a value indicating an open circuit condition.**
SHORTED_LOOP	**The connection between the defined object and the physical device is providing a value indicating a short circuit condition.**
NO_OUTPUT	**No physical device is connected to the Output object.**
PROCESS_ERROR	**A processing error was encountered.**
MULTI_STATE_FAULT	~~**For intrinsic reporting, the Present_Value of the Multi-state object is equal to one of the states in the Fault_Values property and no other fault has been detected.**~~ ***The FAULT_STATE, FAULT_LIFE_SAFETY or FAULT_CHARACTERSTRING fault algorithm has evaluated a fault condition. For details of this evaluation see the respective fault algorithms in Clause 13.4.***

CONFIGURATION_ERROR	**The object's properties are not in a consistent state.**
COMMUNICATION_FAILURE	**Proper operation of the object is dependent on communication with a remote sensor or device and communication with the remote sensor or device has been lost.**
MONITORED_OBJECT_FAULT	***Indicates that the monitored object is in fault.***
UNRELIABLE_OTHER	**The controller has detected that the present value is unreliable, but none of the other conditions describe the nature of the problem. A generic fault other than those listed above has been detected, e.g., a Binary Input is not cycling as expected.**
MEMBER_FAULT	**Indicates that the ~~group~~ *set of referenced member objects* includes one or more Status_Flags properties whose FAULT flag value is equal to TRUE.**

[Change Clause 12.1, p. 149]

12.1 Accumulator Object Type

…

Accumulator objects that support intrinsic reporting shall apply the UNSIGNED_RANGE event algorithm.

The object and its properties are summarized in Table 12-1 and described in detail in this subclause.

[Change Clause 12.2, p. 156]

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.* The object and its properties are summarized in Table 12-2 and described in detail in this subclause.

[Change Clause 12.3, p. 161]

12.3 Analog Output Object Type

The Analog Output object type defines a standardized object whose properties represent the externally visible characteristics of an analog output. *Analog Output objects that support intrinsic reporting shall apply the OUT_OF_RANGE event algorithm.* The object and its properties are summarized in Table 12-3 and described in detail in this subclause.

[Change Clause 12.4, p. 166]

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.*** **The object and its properties are summarized in Table 12-4 and described in detail in this subclause.**

[Change Clause 12.6, p. 174]

12.6 Binary Input Object Type

…

Binary Input objects that support intrinsic reporting shall apply the CHANGE_OF_STATE event algorithm.

The object and its properties are summarized in Table 12-6 and described in detail in this subclause.

[Change Clause 12.7, p. 179]

12.7 Binary Output Object Type

…

Binary Output objects that support intrinsic reporting shall apply the COMMAND_FAILURE event algorithm.

The object and its properties are summarized in Table 12-8 and described in detail in this subclause.

[Change Clause 12.8, p. 185]

12.8 Binary Value Object Type

…

Binary Value objects that support intrinsic reporting shall apply the CHANGE_OF_STATE event algorithm.

The Binary Value object and its properties are summarized in Table 12-10 and described in detail in this subclause.

[Change Clause 12.12, p. 205]

12.12 Event Enrollment Object Type

~~**The Event Enrollment object type defines a standardized object that represents and contains the information required for managing events within BACnet systems. "Events" are changes of value of any property of any object that meet certain predetermined criteria. The primary purpose for Event Enrollment objects is to define an event and to provide a connection between the occurrence of an event and the transmission of a notification message to one or more recipients.**~~

~~**The Event Enrollment object contains the event-type description, the parameters needed to determine if the event has occurred, and a device to be notified. Alternatively, a Notification Class object may serve to identify the recipients of event notifications. A device is considered to be "enrolled for event notification" if it is the recipient to be notified or one of the recipients in a Notification Class object referenced by the Event Enrollment object.**~~

The Event Enrollment object type defines a standardized object that represents and contains the information required for algorithmic reporting of events. For the general event concepts and algorithmic event reporting, see Clause 13.2.

For the Event Enrollment object, detecting events is accomplished by performing particular event and fault algorithms on monitored values of a referenced object. The parameters for the algorithms are provided by the Event Enrollment object. The standard event algorithms are defined in Clause 13.3. The standard fault algorithms are defined in Clause 13.4. Event Enrollment objects do not modify or otherwise influence the state or operation of the referenced object.

For the reliability indication by the Reliability property of the Event Enrollment object, internal unreliable operation such as configuration error or communication failure takes precedence over reliability indication for the monitored object (i.e., MONITORED_OBJECT_FAULT). Fault indications determined by the fault algorithm, if any, have least precedence.

Clause ~~13~~ *13.2* describes the interaction between Event Enrollment objects*, the Notification Class object,* and the Alarm and Event application services. The Event Enrollment object and its properties are summarized in Table 12-14 and described in detail in this subclause.

[Change Clause 12.12.5, p. 188]

12.12.5 Event_Type

This read-only property, of type BACnetEventType, indicates the type of event algorithm that is to be used to detect the occurrence of events *and the event value notification parameters conveyed in event notifications.*~~ and report to enrolled devices. This parameter~~ *The value of this property* is an enumerated type that may have any ~~of the following values:~~ *standard BACnetEventType value except CHANGE_OF_RELIABILITY.*

~~{CHANGE_OF_BITSTRING, CHANGE_OF_STATE, CHANGE_OF_VALUE, COMMAND_FAILURE, FLOATING_LIMIT, OUT_OF_RANGE, BUFFER_READY, CHANGE_OF_LIFE_SAFETY, EXTENDED, UNSIGNED_RANGE}~~

There is a specific relationship between each event algorithm, ~~the parameter list,~~ *the event algorithm parameters,* and the event *type.* ~~types that are valid for the event.~~ The Event_Type *is determined by the chosen event parameters in the Event_Parameters property and* reflects the *event* algorithm that is used to determine the *event* state ~~of an event~~. The *event* algorithm for each Event_Type is specified in Clause ~~13~~ *13.3*. The Event_Parameters property provides the parameters needed by the algorithm.

~~The valid combinations of Event_Type, Event_State, and Event_Parameters values are summarized in Table 12-15.~~

~~Table 12-15. Event_Types, Event_States, and their Parameters~~

~~Event_Type~~	~~Event_State~~	~~Event_Parameters~~
~~CHANGE_OF_BITSTRING~~	~~NORMAL~~ ~~OFFNORMAL~~	~~Time_Delay~~ ~~Bitmask~~ ~~List_Of_Bitstring_Values~~
~~CHANGE_OF_STATE~~	~~NORMAL~~ ~~OFFNORMAL~~	~~Time_Delay~~ ~~List_Of_Values~~
~~CHANGE_OF_VALUE~~	~~NORMAL~~	~~Time_Delay~~ ~~Bitmask~~ ~~Referenced_Property_Increment~~
~~COMMAND_FAILURE~~	~~NORMAL~~ ~~OFFNORMAL~~	~~Time_Delay~~ ~~Feedback_Property_Reference~~
~~FLOATING_LIMIT~~	~~NORMAL~~ ~~HIGH_LIMIT~~ ~~LOW_LIMIT~~	~~Time_Delay~~ ~~Setpoint_Reference~~ ~~Low_Diff_Limit~~ ~~High_Diff_Limit~~ ~~Deadband~~
~~OUT_OF_RANGE~~	~~NORMAL~~ ~~HIGH_LIMIT~~ ~~LOW_LIMIT~~	~~Time_Delay~~ ~~Low_Limit~~ ~~High_Limit~~ ~~Deadband~~
~~BUFFER_READY~~	~~NORMAL~~	~~Notification_Threshold~~
~~CHANGE_OF_LIFE_SAFETY~~	~~NORMAL~~ ~~OFFNORMAL~~ ~~LIFE_SAFETY_ALARM~~	~~Time_Delay~~ ~~List_Of_Alarm_Values~~ ~~List_Of_Life_Safety_Alarm_Values~~ ~~Mode_Property_Reference~~
~~EXTENDED~~	~~Any BACnetEventState~~	~~Vendor_Id~~ ~~Extended_Event_Type~~ ~~Parameters~~
~~UNSIGNED_RANGE~~	~~NORMAL~~ ~~HIGH_LIMIT~~ ~~LOW_LIMIT~~	~~Time_Delay~~ ~~Low_Limit~~ ~~High_Limit~~

[Change Clause 12.12.7, p. 207]

12.12.7 Event_Parameters

The Event_Parameters property, of type BACnetEventParameter, determines the algorithm used to monitor the referenced object and provides the parameter values needed for this *event* algorithm. ~~The meaning of each value in the Event_Parameters, depends on the algorithm as indicated by the Event_Type column in Table 12-15. Each of the possible parameters is described below.~~ *The parameter values specified in this property serve as event algorithm parameters and as reference type parameters reference properties whose values are used as event algorithm parameters, as defined by the respective event algorithm in Clause 13.3. The mapping to the event algorithm parameters is defined in Table 12-X.*

If the Event Enrollment object evaluates reliability only and does not apply an event algorithm, then the event algorithm NONE shall be set in this property.

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

Event Algorithm	Event Parameters	Event Algorithm Parameters
NONE	none	none
ACCESS_EVENT	List_Of_Access_Events Access_Event_Time_Reference	pAccessEvents Referent's value is pAccessEventTime
BUFFER_READY	Notification_Threshold Previous_Notification_Count	pThreshold pPreviousCount
CHANGE_OF_BITSTRING	Time_Delay Bitmask List_Of_Bitstring_Values	pTimeDelay pBitmask pAlarmValues
CHANGE_OF_CHARACTERSTRING	Time_Delay List_Of_Alarm_Values	pTimeDelay pAlarmValues
CHANGE_OF_LIFE_SAFETY	Time_Delay List_Of_Alarm_Values List_Of_Life_Safety_Alarm_Values Mode_Property_Reference	pTimeDelay pAlarmValues pLifeSafetyAlarmValues Referent's value is pMode
CHANGE_OF_STATE	Time_Delay List_Of_Values	pTimeDelay pAlarmValues
CHANGE_OF_STATUS_FLAGS	Time_Delay Selected_Flags	pTimeDelay pSelectedFlags
CHANGE_OF_VALUE	Time_Delay Bitmask Referenced_Property_Increment	pTimeDelay pBitmask pIncrement
COMMAND_FAILURE	Time_Delay Feedback_Property_Reference	pTimeDelay Referent's value is pFeedbackValue
DOUBLE_OUT_OF_RANGE	Time_Delay Low_Limit High_Limit Deadband	pTimeDelay pLowLimit pHighLimit pDeadband
EXTENDED	Vendor_Id Extended_Event_Type Parameters	pVendorId pEventType pParameters
FLOATING_LIMIT	Time_Delay Setpoint_Reference Low_Diff_Limit High_Diff_Limit Deadband	pTimeDelay Referent's value is pSetpoint pLowDiffLimit pHighDiffLimit pDeadband
OUT_OF_RANGE	Time_Delay Low_Limit High_Limit Deadband	pTimeDelay pLowLimit pHighLimit pDeadband
SIGNED_OUT_OF_RANGE	Time_Delay Low_Limit High_Limit Deadband	pTimeDelay pLowLimit pHighLimit pDeadband
UNSIGNED_OUT_OF_RANGE	Time_Delay Low_Limit High_Limit Deadband	pTimeDelay pLowLimit pHighLimit pDeadband

Event Algorithm	*Event Parameters*	*Event Algorithm Parameters*
UNSIGNED_RANGE	*Time_Delay* *Low_Limit* *High_Limit*	*pTimeDelay* *pLowLimit* *pHighLimit*

~~**Bitmask**~~ ~~This parameter, of type BIT STRING, applies to the CHANGE_OF_BITSTRING event algorithm and the CHANGE_OF_VALUE event algorithm in the special case where the referenced property is a BIT STRING datatype. It represents a bitmask that is used to indicate which bits in the referenced property are to be monitored by the algorithm. A value of one in a bit position indicates that the bit in this position in the referenced property is to be monitored by the algorithm. A value of zero in a bit position indicates that the bit in this position in the referenced property is not significant for the purpose of detecting this CHANGE_OF_BITSTRING or CHANGE_OF_VALUE.~~

~~**List_Of_Bitstring_Values**~~ ~~This parameter is a list of bitstrings that apply to the CHANGE_OF_BITSTRING event algorithm. This list of bitstrings defines the set of states for which the referenced property is OFFNORMAL. Only the bits indicated by the Bitmask are significant. If the value of the referenced property changes to one of the values in the List_Of_Bitstring_Values, then the Event_State property of the Event Enrollment object makes a transition TO-OFFNORMAL and appropriate notifications are sent.~~

~~**List_Of_Values**~~ ~~This parameter is a list of BACnetPropertyStates that apply to the CHANGE_OF_STATE event algorithm. This event algorithm applies to referenced properties that have discrete or enumerated values. The List_Of_Values is a subset of the possible values that the property may have. If the value of the referenced property changes to one of the values in the List_Of_Values, then the Event_State property of the Event Enrollment object makes a transition TO-OFFNORMAL and appropriate notifications are sent.~~

~~**Referenced_Property_Increment**~~ ~~This parameter, of type REAL, applies to the CHANGE_OF_VALUE event algorithm. It represents the increment by which the referenced property must change in order for the event to occur.~~

~~**Time_Delay**~~ ~~This parameter, of type Unsigned, applies to all event types and represents the time, in seconds, that the conditions monitored by the event algorithm must persist before an event notification is issued.~~

~~**Feedback_Property_Reference**~~ ~~This parameter, of type BACnetDeviceObjectPropertyReference, applies to the COMMAND_FAILURE algorithm. It identifies the object and property that provides the feedback to ensure that the commanded property has changed value. This property may reference only object properties that have enumerated values or are of type BOOLEAN.~~

~~**Setpoint_Reference**~~ ~~This parameter, of type BACnetDeviceObjectPropertyReference, applies to the FLOATING_LIMIT event algorithm. It indicates the setpoint reference for the reference property interval.~~

~~**Deadband,**~~ ~~These parameters, of type REAL, apply to the FLOATING_LIMIT and~~

~~High_Diff_Limit,~~ ~~Low_Diff_Limit,~~ ~~High_Limit (REAL),~~ ~~Low_Limit (REAL)~~	~~OUT_OF_RANGE event algorithms. Their use is described in the algorithms for these types in Clause 13.~~
~~Notification_Threshold~~	~~This parameter, of type Unsigned, applies to the BUFFER_READY algorithm. It specifies the value of Records_Since_Notification at which notification occurs.~~
~~List_Of_Life_Safety_Alarm_Values~~	~~This parameter is a list of BACnetLifeSafetyState that applies to the CHANGE_OF_LIFE_SAFETY algorithm. If the value of the referenced property changes to one of the values in the List_Of_Life_Safety_Alarm_Values, then the Event_State property of the Event Enrollment object makes a transition TO-OFFNORMAL and appropriate notifications are sent. The resulting event state is LIFE_SAFETY_ALARM~~
~~List_Of_Alarm_Values~~	~~This parameter is a list of BACnetLifeSafetyState that applies to the CHANGE_OF_LIFE_SAFETY algorithm. If the value of the referenced property changes to one of the values in the List_Of_Alarm_Values, then the Event_State property of the Event Enrollment object makes a transition TO-OFFNORMAL and appropriate notifications are sent. The resulting event state is OFFNORMAL.~~
~~Mode_Property_Reference~~	~~This parameter, of type BACnetDeviceObjectPropertyReference, applies to the CHANGE_OF_LIFE_SAFETY algorithm. It identifies the object and property that provides the operating mode of the referenced object providing life safety functionality(normally the Mode property). This parameter may reference only object properties that are of type BACnetLifeSafetyMode.~~
~~Vendor_Id~~	~~This parameter, of type Unsigned16, is a vendor identification code, assigned by ASHRAE, which is used to distinguish proprietary extensions to the protocol. See clause 23.~~
~~Extended_Event_Type~~	~~This parameter, of type Unsigned, is a value selected by the organization indicated by Vendor_Id, which specifies the interpretation of the Parameters parameter.~~
~~Parameters~~	~~This parameter consists of a set of data values whose interpretation is specified by the combination of Vendor_Id and Extended_Event_Type. The set of data values constitutes the set of input parameters for the proprietary algorithm.~~
~~High_Limit (Unsigned),~~ ~~Low_Limit (Unsigned)~~	~~These parameters, of type Unsigned, apply to the UNSIGNED_RANGE event algorithm. Their use is described in the algorithm for these types in Clause 13.~~
~~List_Of_Access_Events~~	~~This parameter is a list of BACnetAccessEvent values that applies to the ACCESS_EVENT algorithm. If the value of the referenced property is updated to one of the values in the List_Of_Access_Events, then the Event_State property of the Event Enrollment object makes a transition TO-NORMAL and appropriate notifications are sent.~~
~~Access_Event_Time_Reference~~	~~This parameter, of type BACnetDeviceObjectPropertyReference, applies to the ACCESS_EVENT algorithm. It identifies the object and property (normally the Access_Event_Time property) that provides the last update time of the referenced property that is monitored for access events. This parameter may only reference object properties that are of~~

~~type BACnetTimeStamp.~~

~~Selected_Flags~~	~~This parameter, of type BACnetStatusFlags, selects which flags should be monitored for the CHANGE_OF_STATUS_FLAGS algorithm.~~

[Change Clause 12.15, p. 216]

12.15 Life Safety Point Object Type

…

Life Safety Point objects that support intrinsic reporting shall apply the CHANGE_OF_LIFE_SAFETY event algorithm.

For reliability-evaluation, the FAULT_LIFE_SAFETY fault algorithm may be applied.

The Life Safety Point object type and its properties are summarized in Table 12-18 and described in detail in this subclause.

~~***NOTE: Do not confuse the Present_Value state with the Event_State property, which reflects the offnormal state of the Life Safety Point object.***~~

[Change Clause 12.16, p. 223]

12.16 Life Safety Zone Object Type

…

Life Safety Zone objects that support intrinsic reporting shall apply the CHANGE_OF_LIFE_SAFETY event algorithm.

For reliability-evaluation, the FAULT_LIFE_SAFETY fault algorithm may be applied.

~~***NOTE: Do not confuse the Present_Value state with the Event_State property, which reflects the offnormal state of the Life Safety Zone object.***~~

[Change Clause 12.17, p. 230]

12.17 Loop Object Type

…

Loop objects that support intrinsic reporting shall apply the FLOATING_LIMIT event algorithm.

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.

[Change Clause 12.18, p. 237]

12.18 Multi-state Input Object Type

…

Multi-state Input objects that support intrinsic reporting shall apply the CHANGE_OF_STATE event algorithm.

For reliability-evaluation, the FAULT_STATE fault algorithm may be applied.

The Multi-state Input object type and its properties are summarized in Table 12-21 and described in detail in this subclause.

~~***NOTE: Do not confuse the Present_Value state with the Event_State property, which reflects the offnormal state of the Multi-State Input.***~~

[Change Clause 12.19, p. 242]

12.19 Multi-state Output Object Type

…

Multi-state Output objects that support intrinsic reporting shall apply the COMMAND_FAILURE event algorithm.

The Multi-state Output object type and its properties are summarized in Table 12-22 and described in detail in this subclause.

[Change Clause 12.20, p. 246]

12.20 Multi-state Value Object Type

…

Multi-state Value objects that support intrinsic reporting shall apply the CHANGE_OF_ STATE event algorithm.

For reliability-evaluation, the FAULT_STATE fault algorithm may be applied.

The Multi-state Value object type and its properties are summarized in Table 12-23 and described in detail in this subclause.

~~***NOTE: Do not confuse the Present_Value state with the Event_State property, which reflects the offnormal state of the Multi-state Value.***~~

[Change Clause 12.23, p. 259]

12.23 Pulse Converter Object Type

…

Pulse Converter objects that support intrinsic reporting shall apply the OUT_OF_RANGE event algorithm.

The object and its properties are summarized in Table 12-27 and described in detail in this subclause.

[Change Clause 12.25, p. 271]

12.25 Trend Log Object Type

…

Event reporting (notification) may be provided to facilitate automatic fetching of log records by processes on other devices such as fileservers. Support is provided for algorithmic reporting; optionally, intrinsic reporting may be provided. *Trend Log objects that support intrinsic reporting shall apply the BUFFER_READY event algorithm.*

…

[Change Clause 12.26, p. 279]

12.26 Access Door Object Type

…

Access Door objects that support intrinsic reporting shall apply the CHANGE_OF_STATE event algorithm.

For reliability-evaluation, the FAULT_STATE fault algorithm may be applied.

The object and its properties are summarized in Table 12-30 and described in detail in this subclause.

[Change Clause 12.27, p. 286]

12.27 Event Log Object Type

…

Event reporting (notification) may be provided to facilitate automatic fetching of event log records by processes on other devices such as fileservers. Support is provided for algorithmic reporting; optionally, intrinsic reporting may be provided. *Event Log objects that support intrinsic reporting shall apply the BUFFER_READY event algorithm.*

…

[Change Clause 12.28, p. 293]

12.28 Load Control Object Type

…

The Load Control object shall exhibit restorative behavior across a restart or time change of the BACnet device in which it resides. The shed request property values shall be maintained across a device restart. Upon device restart or a time change, the object shall behave as if Start_Time were written and shall re-evaluate the state machine's state.

Load Control objects that support intrinsic reporting shall apply the CHANGE_OF_STATE event algorithm.

[Change Clause 12.28, p. 304]

12.30 Trend Log Multiple Object Type

…

Event reporting (notification) may be provided to facilitate automatic fetching of log records by processes on other devices such as fileservers. Mechanisms for both algorithmic and intrinsic reporting are provided. *Trend Log Multiple objects that support intrinsic reporting shall apply the BUFFER_READY event algorithm.*

…

[Change Clause 12.37, p. 356]

12.37 CharacterString Value Object Type

…
If a set of strings is known and fixed, then a MultiState Value object is an alternative that may provide some benefit to automated processes consuming the numeric Present_Value.

CharacterString Value objects that support intrinsic reporting shall apply the CHANGE_OF_CHARACTERSTRING event algorithm.

For reliability-evaluation, the FAULT_CHARACTERSTRING fault algorithm may be applied.

[Change Clause 12.39, p. 365]

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.

[Change Clause 12.40, p. 370]

12.40 BitString Value Object Type

The BitString 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 BitString Value object to make any kind of bitstring data value accessible to other BACnet devices. The mechanisms by which the value is derived are not visible to the BACnet client.

BitString Value objects that support intrinsic reporting shall apply the CHANGE_OF_BITSTRING event algorithm.

[Change Clause 12.43, p. 380]

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.

[Change Clause 12.44, p. 385]

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.

[Change Clause 12.50, p. 405]

12.50 Global Group Object Type

…
A Global Group object differs from a Group object in that its members can be from anywhere in the BACnet internetwork, it supports intrinsic event reporting, and it exposes a method for sending periodic COV notifications. The Global Group object is able to monitor all referenced Status_Flags properties to detect changes to non-normal states and can initiate an event notification message conveying the values of all of the members of the group. This provides a mechanism to define a large set of property values that are made available when an event occurs.

Global Group objects that support intrinsic reporting shall apply the CHANGE_OF_STATUS_FLAGS event algorithm. The pSelectedFlags parameter used shall only have the IN_ALARM bit set.

For reliability-evaluation, the FAULT_STATUS_FLAGS fault algorithm shall be applied.

The Global Group object and its properties are summarized in Table 12-Y1 and described in detail in this subclause.
…

[Change Clause 13, p 411]

13 ALARM AND EVENT SERVICES

This clause describes the conceptual approach and application services used in BACnet to manage communication related to events. Object types relating to event management are defined in Clause 12. In general, "events" are changes of value of certain properties of certain objects, or internal status changes, that meet predetermined criteria. There are ~~three~~ *two* mechanisms provided in BACnet for managing events: change of value reporting~~, intrinsic reporting, and algorithmic change reporting~~ *and event reporting.*

Change of value (COV) reporting allows a COV-client to subscribe with a COV-server, on a permanent or temporary basis, to receive reports of some changes of value of some referenced property based on fixed criteria. Certain BACnet standard objects may optionally support COV reporting. If a standard object provides COV reporting, then changes of value of specific properties of the object, in some cases based on programmable increments, trigger COV notifications to be sent to one or more subscriber clients. Typically, COV notifications are sent to supervisory programs in COV-client devices or to operators or logging devices. Proprietary objects may support COV reporting at the implementor's option.

~~Intrinsic~~ *Event* reporting allows ~~a~~ BACnet device*s* to ~~provide~~ *contain* one or more ~~event~~ *objects (event-initiating objects)*~~sources, intrinsic to the device~~, that generate event notifications that may be directed to one or more destinations. *Event reporting comes in three forms: intrinsic reporting, algorithmic reporting and alert reporting. Typically, event notifications are sent to operators or logging devices represented by "processes" within a notification-client device.*

~~**Certain BACnet standard objects may optionally support intrinsic reporting by supporting optional properties that define the type of event to be generated and options for handling and routing of the notifications. Internal status changes and alarms may also use intrinsic reporting to generate diagnostic notifications. Proprietary objects may support intrinsic reporting at the implementor's option.**~~

~~**Algorithmic change reporting allows a notification-client to subscribe with a notification-server to receive reports of changes of value of any property of any object on the basis of predetermined, and network-visible, criteria. Any of the standardized algorithms described in Clause 13.3 may be used to establish criteria for change reporting. Once established, occurrences of change may be reported to one or more destinations based on further criteria. Algorithmic change reporting differs from intrinsic reporting in that Event Enrollment objects are used to determine the event condition(s).**~~

~~**Intrinsic and algorithmic change reporting rely on the concept of event classificationfor selective reporting of individual occurrences of events. Particular intrinsic or algorithmic change events may specify a notification class number that is directly related to a Notification Class object within the initiating device. The Notification Class object is used to specify the handling and routing of events to one or more destinations. The Notification Class object defines the priorities to be used in event-notification messages, whether acknowledgment by an application process or human operator is required, and at what time periods during the week given destinations are to be used.**~~

~~**BACnet devices that support event notification are free to define any number of unique conditions that trigger alarm or event notifications. Alarms and events are broadly classified into one of three possible groups: fault, offnormal, and normal. A "fault" condition is a malfunction, nearly always representing a failure within the automation system itself. An "offnormal" condition is a condition within the system that is not normally expected or is outside the bounds of ideal operation. A "normal" condition is anything else.**~~

~~**Event-initiating objects may identify their "state" from moment to moment as one of any number of possibly unique event states. Notifications are triggered by the "transition" of conditions for an object, usually from one unique event state to another. A transition to a particular event state may be used to identify specific unique handling for the notification(s) generated by an object, for example, controlling the destination for a notification or whether the particular transition event should require**~~

~~acknowledgment. In these contexts, all transitions that result in a state that is not normal, and not a fault, are considered to be TO-OFFNORMAL transitions. Transitions to any fault state are considered to be TO-FAULT transitions. All other transitions are, by definition, TO-NORMAL transitions.~~

~~Events may be selectively identified as belonging to the category of "alarms" or "events." Event-initiating objects indicate this distinction through the Notify_Type property. In BACnet, the singular distinction between alarms and all other events is whether the event will be reported by the GetAlarmSummary service or not. Alarms will be reported by the GetAlarmSummary service, while all other events will not be reported by GetAlarmSummary. In every other respect, BACnet makes no distinction between an alarm and an event.~~

~~None of the reporting mechanisms is preferred, as each addresses particular needs generally found in building automation and control systems. A given BACnet device may use any or all of these mechanisms to provide alarm and event management functions. However, each mechanism dictates a standardized complement of services and/or objects that are used to realize their functions.~~

[Replace Clause 13.2, p 414]

13.2 Event Reporting

Event reporting is used to detect and report conditions that are broadly categorized into one of three possible groups: fault, offnormal, and normal. A "fault" condition is a malfunction, nearly always representing a failure within the automation system itself. An "offnormal" condition is a condition within the system that is not normally expected or is outside the bounds of ideal operation. A "normal" condition is anything else.

Objects which support event reporting are called event-initiating objects. Event-initiating objects identify their "event state" from moment to moment as one of any number of possibly unique event states. Notifications are triggered by the "transition" of conditions for an object, usually from one unique event state to another. In these contexts, all states that are not normal and not fault are offnormal states, and transitions that result in an offnormal state are considered to be TO-OFFNORMAL transitions. Transitions to any fault state are considered to be TO-FAULT transitions. All other transitions are, by definition, TO-NORMAL transitions.

Intrinsic reporting consists of an object monitoring its own properties, whereas algorithmic reporting consists of an object monitoring properties of other objects. Certain BACnet standard objects may optionally support intrinsic reporting by supporting optional properties that define the type of event to be generated and options for handling and routing of the notifications. Proprietary objects may support intrinsic reporting at the implementor's option.

Algorithmic reporting allows the monitoring of objects that do not provide intrinsic reporting, or the monitoring of an object with an algorithm or algorithm parameters that differ from those configured in the object. Any of the standardized algorithms described in Clause 13.3 may be used to establish criteria for algorithmic reporting. Algorithmic reporting differs from intrinsic reporting in that Event Enrollment objects are used to determine the event condition(s).

Alert reporting allows any object to provide event reports that are unrelated to the object's event state and the intrinsic reporting algorithm of the object. Conceptually, when the need for an alert message is identified by an object, the alert is passed to an Alert Enrollment object for distribution. Alerts differ from intrinsic reporting and algorithmic reporting in that there are no standard conditions under which alerts are generated and in that alerts are stateless and cannot be acknowledged.

Events may be selectively identified as belonging to the category of "alarms" or "events." Event-initiating objects indicate this distinction through the Notify_Type property. Conceptually, alarms are events that are intended to be seen and reacted to by human operators. Operator workstation software is written, for example, to facilitate the reviewing and acknowledgement of alarms, possibly conveying the acknowledgement over the network via the AcknowledgeAlarm service. Events may or may not be of interest to operators and are typically intended for machine-to-machine communication. Applications of events include equipment interlocks, temperature overrides, the transition to a load-shedding

condition, and so on. In the BACnet protocol, the singular distinction is that alarms will be reported by the GetAlarmSummary service, while all other events will not. In every other respect, BACnet makes no distinction between an alarm and an event.

The event notification services contain a 'Time Stamp' parameter that indicates the chronological order of events. This 'Time Stamp' may be the actual time as determined by the local device clock or, if the device has no clock, a sequence number. Sequence numbers are required to increase monotonically up to their maximum value, at which point the number "wraps around" to zero. A device may have a single sequence number for all event-initiating objects, or it may have a separate sequence number for each object.

Event notifications may be specified to use either confirmed or unconfirmed services for notification messages. By providing two kinds of notification mechanisms, BACnet allows the application designer to determine the relative importance of each event and whether or not notification of its occurrence is essential or merely desirable. In the former case, notification can be carried out with a confirmed service and repeated for as many recipients as required. In the latter case, an unconfirmed service using a broadcast or multicast address may be used.

Notification Class objects provide event classification and specify the destination devices for notification messages using BACnetRecipients. The recipients may be individual devices, groups of devices with a common multicast address, or all devices reachable by a broadcast address. If a broadcast is used, the scope may be limited to all devices on a single network or it may be extended to encompass all devices on a BACnet internetwork. The Notification Class object defines the priorities to be used in event notification messages, whether acknowledgment by an application process or a human operator is required, and at what time periods during the week given destinations are to be used.

BACnet event reporting provides support for event acknowledgment whereby an operator indicates that the event transition has been reviewed. The need for acknowledgements is enabled or disabled by event class through the Notification Class object and by transition type (fault, offnormal or normal).

13.2.1Event Detection and Reporting Model

The BACnet event detection and reporting model as outlined in this clause applies to all BACnet objects, both standard and proprietary. The event algorithms are described in Clause 13.3.

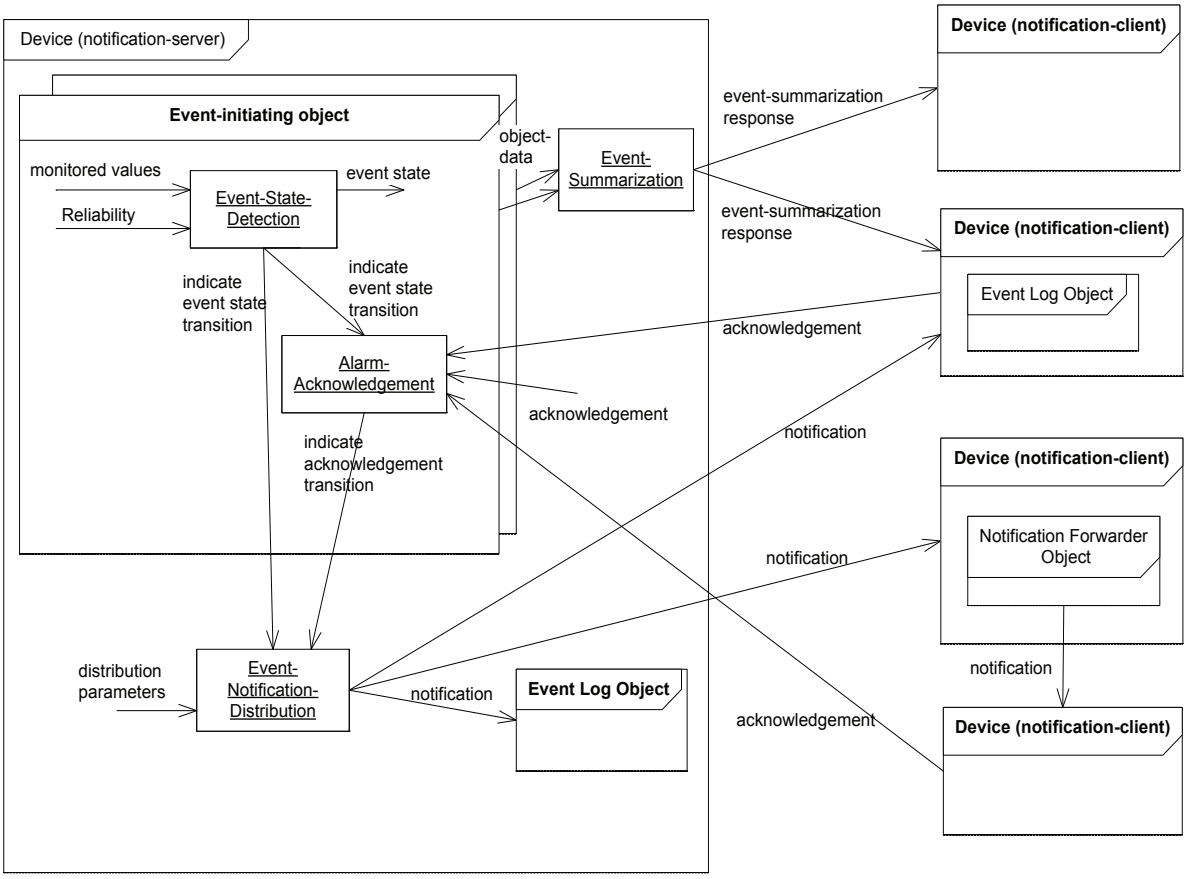

Figure 13-X1. Overview of the Event Detection and Reporting Model

The event-detection and reporting model used in BACnet is separated into four main concepts: event-state-detection, alarm-acknowledgment, event-summarization, and event-notification-distribution. The flow of data between the main parts of the model and the roles that the different objects play in event reporting are shown in Figure 13-X1.

Devices that contain event-initiating objects (notification-servers) interact with one or more devices which process the event information (notification-clients). A notification-client can receive the event information through an event notification or by querying the notification-server via an event-summarization service.

Event-state-detection consists of the monitoring of one or more values (including a Reliability value) in order to evaluate the object's event state.

An object that supports event-state-detection (intrinsic reporting or algorithmic reporting) may also support alarm-acknowledgement. Alarm-acknowledgment is an optional functionality whereby an event transition requires acknowledgment by an operator. A notification-client performs an acknowledgement by issuing an AcknowledgeAlarm service request. Additionally, the acknowledgement may be performed by means local to the device (e.g. an alarm reset button).

Event-summarization provides the ability for a device to retrieve event information independent of notifications, allowing notification-clients that have missed notifications or that are not subscribed for

notifications to easily determine the event state of all objects in a device. A device is required to support event-summarization if it is capable of containing objects that support event-state-detection.

Event-notification-distribution refers to the process involved in sending notifications of event state transitions and acknowledgment transitions using ConfirmedEventNotifications and UnconfirmedEventNotifications to a set of notification-clients and to local Event Log objects. Event-notification-distribution is provided via Notification Class objects and, optionally, Notification Forwarder objects. Devices that support event-state-detection shall support event-notification-distribution.

Objects support event-state-detection via intrinsic reporting, or event-state-detection can be provided for the object via another object performing algorithmic reporting. Notification Class, Notification Forwarder and any objects that implement algorithmic reporting shall not be permitted to implement intrinsic reporting.

The following clauses specify the Event Detection and Reporting model independent of the object types. The Event Log and Notification Forwarder object processes notifications and are included in the overview for information. These objects and their associations to event notifications are specified in Clause 12.

13.2.2Event-State-Detection

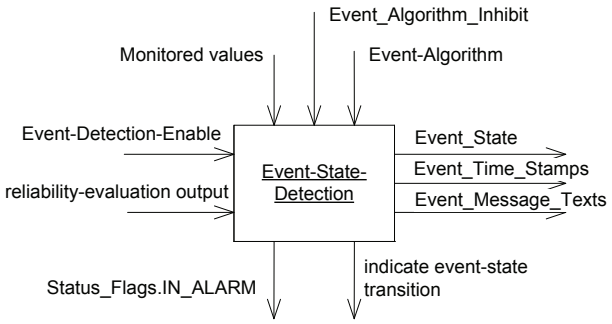

Figure 13-X2. The Event-State-Detection Process

In the event-state-detection process, the event algorithm and the Reliability property together determine the event state of an object. The event algorithm determines the normal or offnormal states and the Reliability property determines whether or not the event state will indicate a fault. Fault detection takes precedence over the detection of normal and offnormal states. As such, when Reliability has a value other than NO_FAULT_DETECTED, the event-state-detection process will determine the object's event state to be FAULT.

When the event-state-detection process is disabled via the Event_Detection_Enable, both the event algorithm and the Reliability value are ignored, and Event_State remains NORMAL.

When the monitoring of values is disabled by Event_Algorithm_Inhibit, the result of the event algorithm will be ignored and all TO-OFFNORMAL and TO-NORMAL transitions will be disabled with the exception of TO-NORMAL transitions from FAULT. Event_Algorithm_Inhibit does not impact transitions to or from the fault state.

For intrinsic reporting in standard object types, the event algorithm is implied by the object type. For algorithmic reporting, the Event_Enrollment object contains the Event_Type property which indicates the event algorithm. An object, standard or proprietary, shall use only a single event algorithm. For objects in which the event algorithm is configurable, there is an expectation that the event algorithm does not change dynamically.

In objects that support event-state-detection, the reporting of changes in Reliability cannot be disabled while event-state-detection is enabled. Instead, the object may be stopped from detecting faults through the Reliability_Evaluation_Inhibit property. If an object that supports event-state-detection does not

have a Reliability property, then the reliability evaluation is assumed to indicate NO_FAULT_DETECTED and no fault transitions shall occur.

A transition of the event state is indicated to the Alarm-Acknowledgement process (see Clause 13.2.2.1.5) and the event-notification-distribution process (see Clause 13.2.5).

13.2.2.1 Event-State-Detection State Machine

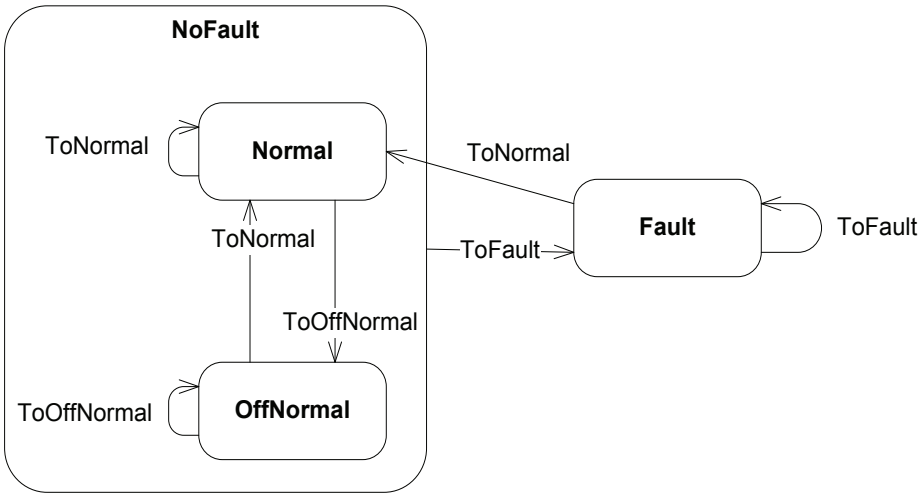

Figure 13-X3. Event-State-Detection State Machine

The state machine in Figure 13-X3 shows all possible transitions of the general event-state-detection model. Not all ToNormal or ToOffNormal transitions are supported by every event algorithm.

If the Event_Detection_Enable property is FALSE, then this state machine is not evaluated. In this case, no transitions shall occur, Event_State shall be set to NORMAL, and Event_Time_Stamps, Event_Message_Text and Acked_Transitions shall be set to their respective initial conditions.

It is important to note that, in the following clauses, transitions from one Event_State to the same Event_State value are indicated by the event algorithm or by reliability-evaluation.

13.2.2.1.1 Normal

In the Normal state reliability-evaluation indicates a value of NO_FAULT_DETECTED and either the event algorithm indicates a normal event state or the Event_Algorithm_Inhibit is TRUE.

ToOffNormal

If reliability-evaluation indicates a value of NO_FAULT_DETECTED and the event algorithm indicates an offnormal event state and Event_Algorithm_Inhibit is FALSE,

then perform the corresponding transition actions (see Clause 13.2.2.1.4) and enter the OffNormal state.

ToFault

If reliability-evaluation indicates a value other than NO_FAULT_DETECTED,

then perform the corresponding transition actions and enter the Fault state.

ToNormal

If reliability-evaluation indicates a value of NO_FAULT_DETECTED and the event algorithm indicates a transition to the Normal state and Event_Algorithm_Inhibit is FALSE,

then perform the corresponding transition actions and re-enter the Normal state.

13.2.2.1.2 OffNormal

In the OffNormal state, reliability-evaluation indicates a value of NO_FAULT_DETECTED and the event algorithm indicates an offnormal event state and Event_Algorithm_Inhibit is FALSE. (Note that the OffNormal state includes all event states other than NORMAL and FAULT).

ToOffNormal

If reliability-evaluation indicates a value of NO_FAULT_DETECTED and the event algorithm indicates a transition to the OffNormal state and Event_Algorithm_Inhibit is FALSE,

then perform the corresponding transition actions and re-enter the OffNormal state.

ToFault

If reliability-evaluation indicates a value other than NO_FAULT_DETECTED,

then perform the corresponding transition actions and enter the Fault state.

ToNormal

If reliability-evaluation indicates a value of NO_FAULT_DETECTED and the event algorithm indicates a normal event state,
or
if reliability-evaluation indicates a value of NO_FAULT_DETECTED and Event_Algorithm_Inhibit is TRUE,

then perform the corresponding transition actions and enter the Normal state.

13.2.2.1.3 Fault

In the Fault state reliability-evaluation indicates a value other than NO_FAULT_DETECTED.

ToNormal

If reliability-evaluation indicates a value of NO_FAULT_DETECTED,

then perform the corresponding transition actions and enter the Normal state.

ToFault

If reliability-evaluation indicates a different Reliability value and the new Reliability value is not NO_FAULT_DETECTED or reliability-evaluation indicates a transition to the Fault state with the same Reliability value,

then perform the corresponding transition actions and re-enter the Fault state.

13.2.2.1.4 Transition Actions

This clause describes the actions to be taken when a transition of the event-state-detection state machine occurs. The actions are the same for all transitions and they shall be executed even if the transition does not change the event state (e.g., to the ToOffNormal from the OffNormal state).

Store the new event state in the event-initiating object's Event_State property. Note that the Event_State property shall reflect the specific BACnetEventState returned by the event algorithm (i.e. it is not acceptable to set Event_State to OFFNORMAL when the returned value is HIGH_LIMIT).

Store the time of the transition in the corresponding entry of the Event_Time_Stamps property.

Store the message text that is generated for distribution with the notification in the corresponding entry of the Event_Message_Texts property, if present.

Indicate the transition to the Alarm-Acknowledgement process (see Clause 13.2.3) and the event-notification-distribution process (see Clause 13.2.5).

13.2.2.1.5 Inhibiting Detection of Offnormal Conditions

The Event_Algorithm_Inhibit property temporarily overrides the event algorithm thus maintaining a normal Event_State regardless of the existance of offnormal conditions. The effect of this property on the Event_State property is shown in Figure 13-X4.

Upon Event_Algorithm_Inhibit changing to TRUE, the event shall transition to the NORMAL state if not already there. While Event_Algorithm_Inhibit remains TRUE, no transitions shall occur except those into and out of FAULT. Upon Event_Algorithm_Inhibit changing to FALSE, any condition shall hold for the its regular time delay after the change to FALSE before a transition is generated.

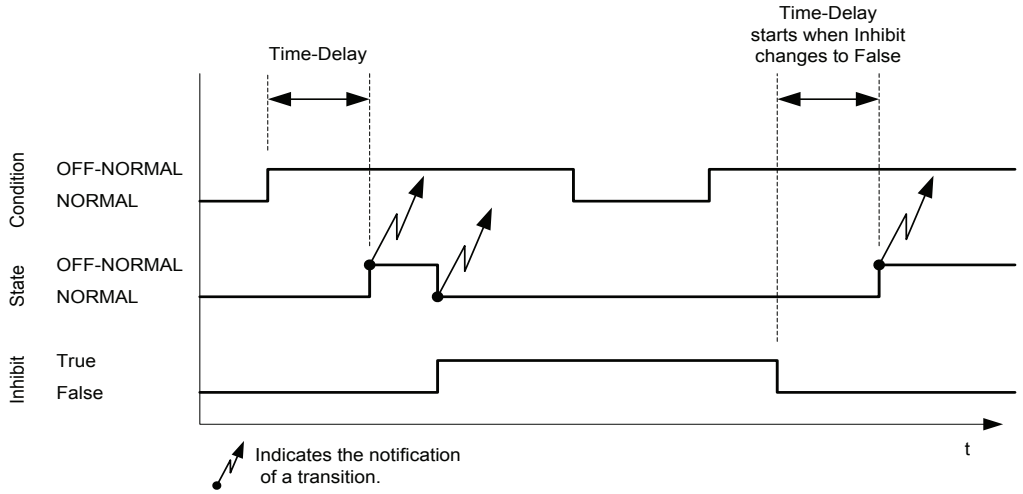

Figure 13-X4. Effect of Event_Algorithm_Inhibit on Event_State

13.2.3Alarm-Acknowledgment

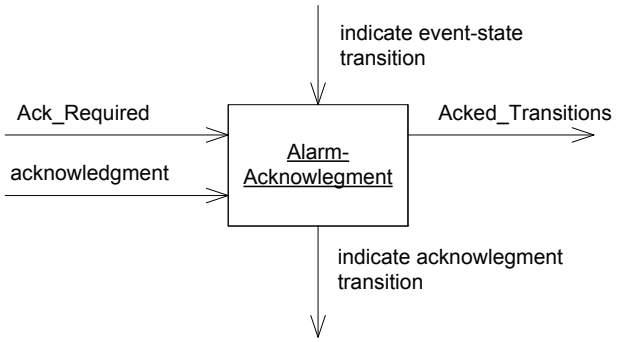

Figure 13-X5. Alarm-Acknowledgement

The alarm-acknowledgement process, shown in Figure 13-X5, is responsible for maintaining the acknowledgment state for each of the transition types (TO-NORMAL, TO-FAULT, TO-OFFNORMAL) in the Acked_Transitions property and for indicating acknowledgement transitions to the event-notification-distribution process.

Event state transitions are received from the event-state-detection state machine. Acknowledgement indications are received from the AcknowledgeAlarm service procedure and from a means local to the device (e.g. reset button) and acknowledgment transitions are indicated to the event-notification-distribution process (see Clause 13.2.5). Every device that supports acknowledgable transitions shall support execution of the AcknowledgeAlarm service.

Whether or not an acknowledgement is required is determined by the Ack_Required property from the referenced Notification Class object.

When an event state transition is received, the corresponding bit in Acked_Transitions is either set or cleared. If the corresponding bit in Ack_Required is set, then the bit in Acked_Transitions is cleared, otherwise it is set.

When an acknowledgement indication is received, the corresponding bit in Acked_Transitions is set and an Acknowledgment transition is indicated to the event-notification-distribution process.

Modification of Ack_Required does not change the value of Acked_Transitions properties of associated objects.

13.2.4Event-Summarization

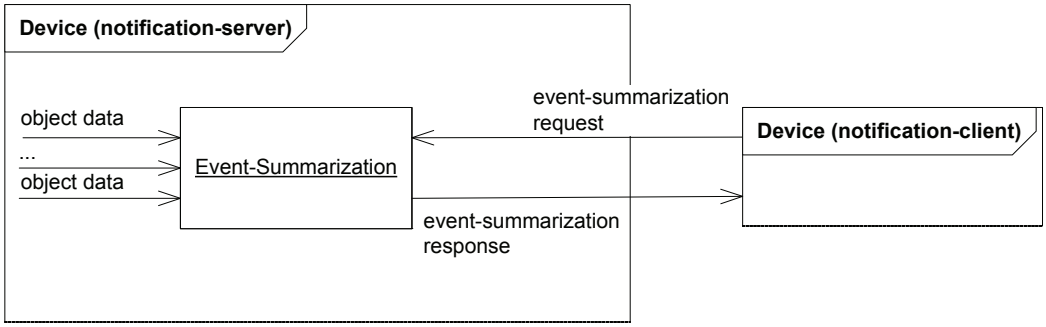

Figure 13-X6. Event-Summarization

Event-summarization (see Figure 13-X6) provides the means by which a notification-client can efficiently determine the current state of multiple event-initiating objects in a device when it has not received all of the event notifications (either because it was offline, the network is down, or the client is not in the distribution list for the event-initiating objects). The different BACnet services that can be used for event-summarization are compared in Table 13-X1.

Table 13-X1: Event-Summarization Services

Service	Selected objects	Response
GetAlarmSummary (see Clause 13.10)	All event-initiating objects where: Event_Detection_Enable is TRUE and Event_State is not NORMAL and Notify_Type equal to ALARM	List of Object_Identifier Event_State Acked_Transitions
GetEnrollmentSummary (see Clause 13.11)	All event-initiating objects where: Event_Detection_Enable is TRUE and Acknowledgement Filter matches and Enrollment Filter matches (optional) and Event State Filter matches (optional) and Event Type Filer matches (optional) and Priority Filter matches (optional) and Notification Class Filter matches(optional)	List of Object_Identifier Event Type Event_State Priority Notification_Class
GetEventInformation (see Clause 13.12)	All event-initiating objects where: Event_Detection_Enable is TRUE and Event_State is not NORMAL or any bit in the Acked_Transitions property is not set	List of Object_Identifier Event_State Acked_Transitions Event_Time_Stamps Notify_Type Event_Enable Event Priorities

Notification-servers are required to support execution of the GetEventInformation service. Support for the execution of the GetAlarmSummary and GetEnrollmentSummary services is optional.

It is important to note that the results returned by the summarization services ignore the value of the event-initiating objects' Event_Enable properties and the notification filtering fields in the Recipient_List property of related Notification Class objects. This results in the set of objects returned by the services being different than the set of objects a notification-client would be made aware of via the event notification services.

13.2.5Event-Notification-Distribution

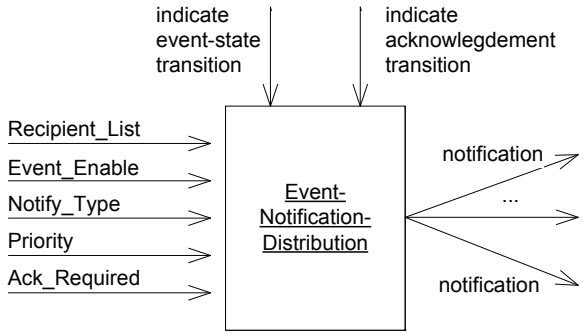

Figure 13-X7. Event-Notification-Distribution

The event-notification-distribution process (see Figure 13-X7) is responsible for distributing event transition notifications to all local objects and to notification-clients. Event transition notifications and acknowledgement notifications are distributed to notification-clients via the ConfirmedEventNotification and UnconfirmedEventNotification services.

The Event_Enable and Notify_Type inputs are provided by the event-initiating object, and the Recipient_List, Priority, and Ack_Required inputs are provided by the Notification Class object referenced by the event-initiating object.

When an event state transition or an acknowledgment transition is indicated, notifications are distributed to the notification-clients specified by the Recipient_List input. Additional information provided by the Recipient_List input controls the distribution of the notification. For acknowledgment notifications, the transition type of notification (TO_FAULT, TO_NORMAL, or TO_OFFNORMAL) is the same as the transition type of the event transition that is being acknowledged.

The external distribution of notifications can be disabled by the Event_Enable property. The notifications are disabled if the bit corresponding to the transition is set to FALSE. While the Event_Enable property can take on any combination of three bits, the two most used values are (T, T, T), indicating that all transitions are to be distributed, or (F, F, F), indicating that no transitions are to be distributed.

Distribution of notifications to local objects (e.g., Event Log objects) in the device is not controlled by the Recipient_List information unless otherwise indicated (e.g. Notification Forwarder objects). It is a local matter whether or not the distribution of notifications to local objects is affected by the Event_Enable property (e.g., whether or not Event Log objects record the notifications).

13.2.5.1 Notification Forwarding

Notification forwarding is related to, but separate from, the distribution of event notifications by the Notification Class object.

The forwarding of notifications, whether by a local or remote Notification Forwarder object, modifies the notification distribution as setup by a Notification Class. The Notification Class distributes the notification to the device that contains the Notification Forwarder object, and the Notification Forwarder object re-sends the notification with recipient information from the Notification Forwarder object. See Clause 12.X for a detailed description of notification forwarding as implemented by Notification Forwarding objects.

13.2.5.2 Service Parameters of Event Notification Service Requests

The event-notification-distribution process generates event notifications and the service parameters for those notifiations are generated accoirding to Table 13-X2.

Table 13-X2: Event Notification Service Parameter Values

Service Parameter	Event State Transition (all transitions)	Acknowledgement Transition
Process Identifier	From Recipient_List entry	From Recipient_List entry
Initiating Device Identifier	Device object identifier of the device which contains the event-initiating object	Device object identifier of the device which contains the event-initiating object
Event Object Identifier	Object identifier of the event-initiating object	Object identifier of the event-initiating object
Time Stamp	Value of the Event_Time_Stamps array entry that corresponds to the 'To State'.	The time at which the AcknowledgeAlarm service is executed or If acknowledged locally, the time that the acknowledgement is performed.
Notification Class	Value of the event-initiating object's Notification_Class property.	Value of the event-initiating object's Notification_Class property.
Priority	Value of the Priority property entry that corresponds to the 'To State'	Value of the Priority property entry that corresponds to the 'To State'.
Event Type	When 'To State' or 'From State' is FAULT, set to CHANGE_OF_RELIABILITY, Otherwise the value associated with the event-initiating object's event algorithm.	not present
Message Text	Optional The value is a local matter and is reflected in the Event_Message_Texts array, if the property exists.	Optional The value is a local matter.
Notify Type	Value of Notify_Type	ACK_NOTIFICATION
AckRequired	Value of the Ack_Required bit that corresponds to 'To State'.	Not present
From State	Value of Event_State before this transition	Not present
To State	Value of property Event_State after this transition	The 'To State' parameter from the transition being acknowledged
Event Values	As defined for the Event_Type	not present

13.2.5.3 Fault Event Notifications

For all transitions to, or from, the FAULT state, the corresponding event notification shall use the Event Type CHANGE_OF_RELIABILITY.

Table 13-X3. CHANGE_OF_RELIABILITY Notification Parameters

Parameter	Value
reliability	The value of the Reliability property of the event-initiating object.
status-flags	The value of the Status_Flags property of the event-initiating object.
property-values	As specified in Table 13-X4.

The content of the property-values parameter of CHANGE_OF_RELIABILITY event notifications depends on the type of the event-initiating object. The property values required to be conveyed are specified in Table 13-X4, and shall be included in the order shown in the table. Object types that are not included in table 13-X4 are not required to include any extra properties in CHANGE_OF_RELIABILITY notifications.

Additional properties of the event-initiating object may be conveyed in the property-values parameter following the properties which are required by this standard. The selection of additional properties to include in the event notification is a local matter.

In the case of the Event Enrollment object, the first property in the property-values parameter shall be the Event Enrollment object's Object_Property_Reference property. The second property is the property that is referenced by the Event Enrollment object's Object_Property_Reference property. All other properties (required and additional properties) shall be from the monitored object. This is the only case where the properties conveyed in the CHANGE_OF_RELIABILITY are not from the event-initiating object.

Table 13-X4. Properties Reported in CHANGE_OF_RELIABILITY Notifications

Object Type	Properties
Access Door	Door_Alarm_State Present_Value
Access Point	Access_Event Access_Event_Tag Access_Event_Time Access_Event_Credential
Access Zone	Occupancy_State
Accumulator	Pulse_Rate Present_Value
Analog Input, Analog Output, Analog Value, Binary Input, Binary Value, BitString Value, CharacterString Value, Integer Value, Large Analog Value, Multi-state Input, Multi-state Value, Positive Integer Value, Pulse Converter	Present_Value
Binary Output, Multi-state Output	Present_Value Feedback_Value
Credential Data Input	Update_Time Present_Value[1]
Event Enrollment	Object_Property_Reference Value of property referenced by Object_Property_Reference[2] Reliability[2]

	Status_Flags[2]
Life Safety Point, Life Safety Zone	Present_Value Mode Operation_Expected
Load Control	Present_Value Requested_Shed_Level Actual_Shed_Level
Loop	Present_Value Controlled_Variable_Value[2] Setpoint[2]
Program	Program_State Reason_For_Halt[2,3] Description_Of_Halt[2,3]

[1] This value may be excluded from the property-value parameter due to security requirements.
[2] This property is, or may be, from a referenced object. If the value is not known by the event-initiating object, then it shall not be included in the property-value parameter.
[3] These properties are optional and are included only if present in the object.

13.2.5.4 Alarm and Event Priority Classification

Alarms and events traversing the BACnet network need prioritization to assure that important information reaches its destination and is acted upon quickly. To assure alarm prioritization at the network level, the Network Priority as defined in Clause 6.2.2 shall be set as a function of the alarm and event priority as defined in Table 13-X5. Annex M provides additional clarity and examples of specific messages and priorities.

Table 13-X5. Alarm and Event Priority - Network Priority Association

Alarm and Event Priority	Network Priority
00 – 63	Life Safety message
64 - 127	Critical Equipment message
128 - 191	Urgent message
192 - 255	Normal message

[Replace Clause 13.3 and all sub-clauses, p. 422]

13.3 Event Algorithms

Table 13-Y1 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-Y1. Standardized Event Algorithms

Event Algorithm	Clause
NONE	13.3.Y8
ACCESS_EVENT	13.3.Y3
BUFFER_READY	13.3.7
CHANGE_OF_BITSTRING	13.3.1
CHANGE_OF_CHARACTERSTRING	13.3.Y7
CHANGE_OF_LIFE_SAFETY	13.3.8
CHANGE_OF_STATE	13.3.2
CHANGE_OF_STATUS_FLAGS	13.3.Y2
CHANGE_OF_VALUE	13.3.3
COMMAND_FAILURE	13.3.4

DOUBLE_OUT_OF_RANGE	13.3.Y4
EXTENDED	13.3.Y1
FLOATING_LIMIT	13.3.5
OUT_OF_RANGE	13.3.6
SIGNED_OUT_OF_RANGE	13.3.Y5
UNSIGNED_OUT_OF_RANGE	13.3.Y6
UNSIGNED_RANGE	13.3.9

Event algorithms monitor a value and evaluate whether the condition for a transition of event state exists. The result of the evaluation, indicated to the Event-State-Detection process, may be a transition to a new event state, a transition to the same event state, or no transition. The final determination of the Event_State property value is the responsibility of the Event-State-Detection process and is subject to additional conditions. See Clause 13.2.

Each of the event algorithms defines its input parameters, the allowable normal and offnormal states, the conditions for transitions between those states, and the notification parameters conveyed in event notifications for the algorithm.

When executing an event algorithm, all conditions defined for the algorithm shall be evaluated in the order as presented for the algorithm. Some algorithms specify optional conditions, marked as "Optional:" Whether or not an implementation uses these conditions is a local matter. If no condition evaluates to true, then no transition shall be indicated to the Event-State-Detection process.

13.3.1 CHANGE_OF_BITSTRING Event Algorithm

The CHANGE_OF_BITSTRING event algorithm detects whether the monitored value of type BIT STRING equals a value that is listed as an alarm value, after applying a bitmask.

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 BIT STRING, 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}.

pAlarmValues — This parameter, of type List Of BIT STRING, represents a list of values that are considered offnormal values.

pBitmask — This parameter, of type BIT STRING, represents the bitmask that defines the bits of pMonitoredValue that are significant for comparison with values of pAlarmValues.
This value is bit-wise ANDed with the pMonitoredValue before comparison with pAlarmValues.

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 are:

(a) If pCurrentState is NORMAL, and pMonitoredValue, after applying pBitmask, is equal to any of the values contained in pAlarmValues for pTimeDelay, then indicate a transition to the OFFNORMAL event state.
(b) If pCurrentState is OFFNORMAL, and pMonitoredValue, after applying pBitmask, is not equal to any of the values contained in pAlarmValues for pTimeDelayNormal, then indicate a transition to the NORMAL event state.
(c) Optional: If pCurrentState is OFFNORMAL, and pMonitoredValue, after applying pBitmask, is equal to one of the values contained in pAlarmValues that is different from the value that caused the last transition to OFFNORMAL and remains equal to that value for pTimeDelay, then indicate a transition to the OFFNORMAL event state.

Figure 13-Y1 depicts those transitions of Figure 13-X3 that this event algorithm may indicate:

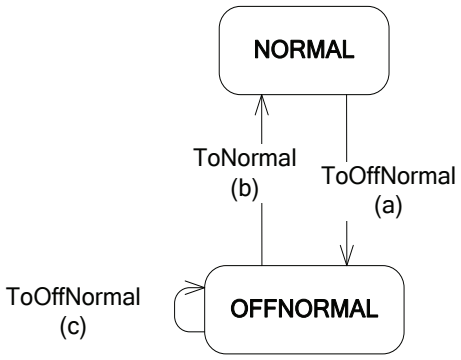

Figure 13-Y1. Transitions indicated by CHANGE_OF_BITSTRING algorithm

The notification parameters of this algorithm are:

Referenced_Bitstring	**This notification parameter, of type BIT STRING, conveys the value of pMonitoredValue.**
Status_Flags	**This notification parameter, of type BACnetStatusFlags, conveys the value of pStatusFlags.**

13.3.2 CHANGE_OF_STATE Event Algorithm

The CHANGE_OF_STATE event algorithm detects whether the monitored value equals a value that is listed as an alarm value. The monitored value may be of any discrete or enumerated data type, including Boolean.

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 is a discrete value that represents the current value of the monitored property. The datatype of the value of this parameter shall be one of the options of BACnetPropertyStates.**
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}.**
pAlarmValues	**This parameter is a list of discrete values that represent the offnormal values. The datatype of the values of this parameter and of pMonitoredValue shall be the same.**
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 are:

(a) If pCurrentState is NORMAL, and pMonitoredValue is equal to any of the values contained in pAlarmValues for pTimeDelay, then indicate a transition to the OFFNORMAL event state.
(b) If pCurrentState is OFFNORMAL, and pMonitoredValue is not equal to any of the values contained in pAlarmValues for pTimeDelayNormal, then indicate a transition to the NORMAL event state.
(c) Optional: If pCurrentState is OFFNORMAL, and pMonitoredValue is equal to one of the values contained in pAlarmValues that is different from the value that caused the last transition to OFFNORMAL, and remains equal to that value for pTimeDelay, then indicate a transition to the OFFNORMAL event state.

Figure 13-Y2 depicts those transitions of Figure 13-X3 that this event algorithm may indicate:

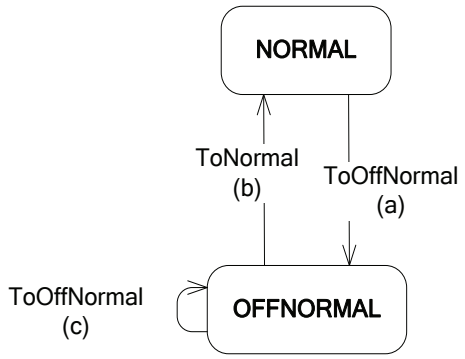

Figure 13-Y2. Transitions indicated by CHANGE_OF_STATE algorithm

The notification parameters of this algorithm are:

New_State — **This notification parameter, of type BACnetPropertyStates, conveys the value of pMonitoredValue.**

Status_Flags — **This notification parameter, of type BACnetStatusFlags, conveys the value of pStatusFlags.**

13.3.3 CHANGE_OF_VALUE Event Algorithm

The CHANGE_OF_VALUE event algorithm, for monitored values of datatype REAL, detects whether the absolute value of the monitored value changes by an amount equal to or greater than a positive REAL increment.

The CHANGE_OF_VALUE event algorithm, for monitored values of datatype BIT STRING, detects whether the monitored value changes in any of the bits specified by a bitmask.

For detection of change, the value of the monitored value when a transition to the 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 REAL or BIT STRING, 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}.**

pIncrement — **This parameter, of type REAL, shall provide a value if and only if pMonitoredValue is of type REAL. It represents the positive increment by which the monitored value of type REAL must change for a new transition.**

pBitmask — **This parameter, of type BIT STRING, shall provide a value if and only if pMonitoredValue is of type BIT STRING. It represents the bitmask that defines the bits of pMonitoredValue that are significant for detecting a change of value.**
This value is bit-wise ANDed with the pMonitoredValue before comparison with the value that has caused the last transition to NORMAL.

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 REAL, are:

(a) If pCurrentState is NORMAL, and the absolute value of pMonitoredValue changes by an amount equal or greater than pIncrement for pTimeDelayNormal, then indicate a transition to the NORMAL event state.

The conditions evaluated by this event algorithm, for a monitored value of type BIT STRING, are:

(a) If pCurrentState is NORMAL, and any of the significant bits of pMonitoredValue change state and remain changed for pTimeDelayNormal, then indicate a transition to the NORMAL event state.

Figure 13-Y3 depicts those transitions of Figure 13-X3 that this event algorithm may indicate:

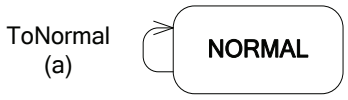

Figure 13-Y3. Transitions indicated by CHANGE_OF_VALUE algorithm

The notification parameters of this algorithm are:

New_Value	**This notification parameter, of type REAL or BIT STRING, conveys the value of pMonitoredValue.** **If pMonitoredValue is of type BIT STRING, then the Changed_Bits option is used.** **If pMonitoredValue is of type REAL, then the Changed_Value option is used.**
Status_Flags	**This notification parameter, of type BACnetStatusFlags, conveys the value of pStatusFlags.**

13.3.4 COMMAND_FAILURE Event Algorithm

The COMMAND_FAILURE event algorithm detects whether the monitored value and the feedback value disagree for a time period. It may be used, for example, to verify that a process change has occurred after writing a property.

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 ABSTRACT-SYNTAX.&Type, 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}.**
pFeedbackValue	**This parameter, of type ABSTRACT-SYNTAX.&Type, represents the feedback value used for comparison with the pMonitoredValue. The datatype of the value of this parameter shall be the same as the datatype of the value of pMonitoredValue.**
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 are:

(a) If pCurrentState is NORMAL, and pFeedbackValue is not equal to pMonitoredValue for pTimeDelay, then indicate a transition to the OFFNORMAL event state.
(b) If pCurrentState is OFFNORMAL, and pMonitoredValue is equal to pMonitoredValue for pTimeDelayNormal, then indicate a transition to the NORMAL event state.

Figure 13-Y4 depicts those transitions of Figure 13-X3 that this event algorithm may indicate:

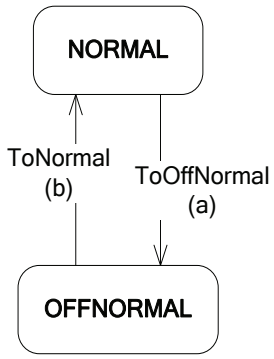

Figure 13-Y4. Transitions indicated by COMMAND_FAILURE algorithm

The notification parameters of this algorithm are:

Command_Value	**This notification parameter, of type ABSTRACT-SYNTAX.&Type, conveys the value of pMonitoredValue.**
Status_Flags	**This notification parameter, of type BACnetStatusFlags, conveys the value of pStatusFlags.**
Feedback_Value	**This notification parameter, of type ABSTRACT-SYNTAX.&Type, conveys the value of pFeedbackValue.**

13.3.5FLOATING_LIMIT Event Algorithm

The FLOATING_LIMIT event algorithm detects whether the monitored value exceeds a range defined by a setpoint, a high difference limit, a low difference limit and a deadband.

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 REAL, 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}.**
pSetpoint	**This parameter, of type REAL, represents the value that defines the reference interval.**
pLowDiffLimit	**This parameter, of type REAL, represents, when subtracted from pSetpoint, the lower limit of the range considered normal.**
pHighDiffLimit	**This parameter, of type REAL, represents, when added to pSetpoint, the higher limit of the range considered normal.**
pDeadband	**This parameter, of type REAL, represents the deadband that is applied to the respective limit before a return to Normal event state is indicated.**
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 provided for this parameter, then it takes on the value of the pTimeDelay parameter.**

Figure 13-Y5 shows the relationship of the various parameters used in the FLOATING_LIMIT algorithm. In this figure, pTimeDelay is assumed to have a value of zero.

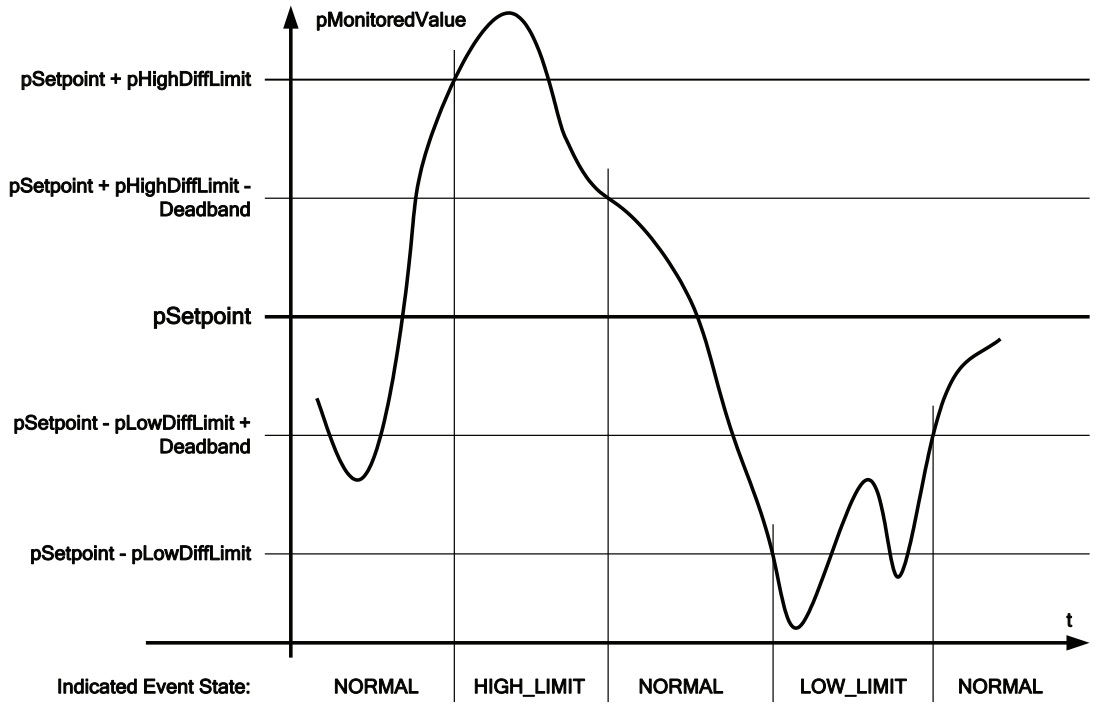

Figure 13-Y5. FLOATING_LIMIT parameter relationship

The conditions evaluated by this event algorithm are:

(a) If pCurrentState is NORMAL, and pMonitoredValue is greater than (pSetpoint + pHighDiffLimit) for pTimeDelay, then indicate a transition to the HIGH_LIMIT event state.
(b) If pCurrentState is NORMAL, and pMonitoredValue is less than (pSetpoint - pLowDiffLimit) for pTimeDelay, then indicate a transition to the LOW_LIMIT event state.

(c) Optional: If pCurrentState is HIGH_LIMIT, and pMonitoredValue is less than (pSetpoint - pLowDiffLimit) for pTimeDelay, then indicate a transition to the LOW_LIMIT event state.
(d) If pCurrentState is HIGH_LIMIT, and pMonitoredValue is less than (pSetpoint + pHighDiffLimit - pDeadband) for pTimeDelayNormal, then indicate a transition to the NORMAL event state.

(e) Optional: If pCurrentState is LOW_LIMIT, and pMonitoredValue is greater than (pSetpoint + pHighDiffLimit) for pTimeDelay, then indicate a transition to the HIGH_LIMIT event state.
(f) If pCurrentState is LOW_LIMIT, and pMonitoredValue is greater than (pSetpoint - pLowDiffLimit + pDeadband) for pTimeDelayNormal, then indicate a transition to the NORMAL event state.

If any of the optional conditions are supported, then all optional conditions shall be supported.

Figure 13-Y6 depicts those transitions of Figure 13-X3 that this event algorithm may indicate:

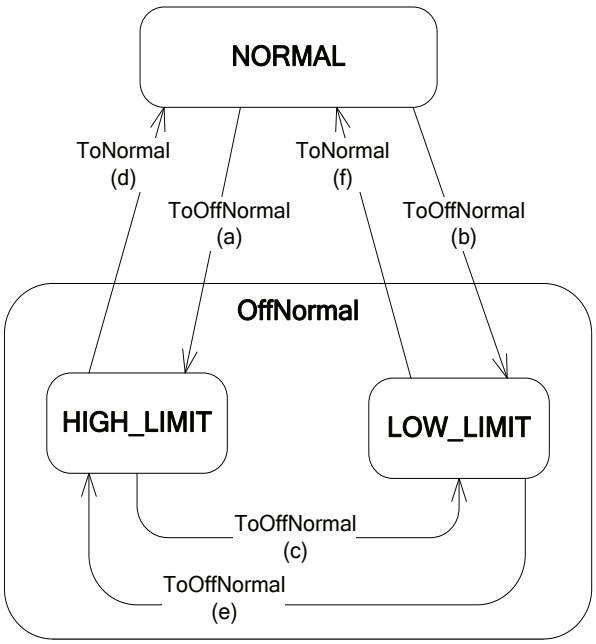

Figure 13-Y6. Transitions indicated by FLOATING_LIMIT algorithm

The notification parameters of this algorithm are:

Reference_Value	**This notification parameter, of type REAL, conveys the value of pMonitoredValue.**
Status_Flags	**This notification parameter, of type BACnetStatusFlags, conveys the value of pStatusFlags.**
Setpoint_Value	**This notification parameter, of type REAL, conveys the value of pSetpoint.**
Error_Limit	**This notification parameter, of type REAL, conveys the value of pLowDiffLimit if** **a) the new state is LOW_LIMIT, or** **b) pCurrentState is LOW_LIMIT and the new state is NORMAL** **This notification parameter conveys the value of pHighDiffLimit if** **a) the new state is HIGH_LIMIT, or** **b) pCurrentState is HIGH_LIMIT and the new state is NORMAL**

13.3.6OUT_OF_RANGE Event Algorithm

The OUT_OF_RANGE event algorithm detects whether the monitored value exceeds a range defined by a high limit and a low limit. Each of these limits may be enabled or disabled. If disabled, the normal range has no higher limit or no lower limit. In order to reduce jitter of the resulting event state, a deadband is applied when the value is in the process of returning to the normal range.

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 REAL, 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}.
pLowLimit	**This parameter, of type REAL, represents the lower limit of the range considered normal.**
pHighLimit	**This parameter, of type REAL, represents the higher limit of the range considered normal.**
pDeadband	**This parameter, of type REAL, represents the deadband that is applied to the respective limit before a return to Normal event state is indicated.**
pLimitEnable	**This parameter, of type BACnetLimitEnable, represents two flags, HighLimitEnable and LowLimitEnable, that separately enable (TRUE) or disable (FALSE) the respective limits applied by the event algorithm. If the value of this parameter is not provided, then both flags shall be set to TRUE (1).**
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 provided for this parameter, then it takes on the value of the pTimeDelay parameter.**

The conditions evaluated by this event algorithm are:

(a) If pCurrentState is NORMAL, and the HighLimitEnable flag of pLimitEnable is TRUE, and pMonitoredValue is greater than pHighLimit for pTimeDelay, then indicate a transition to the HIGH_LIMIT event state.
(b) If pCurrentState is NORMAL, and the LowLimitEnable flag of pLimitEnable is TRUE, and pMonitoredValue is less than pLowLimit for pTimeDelay, then indicate a transition to the LOW_LIMIT event state.

(c) If pCurrentState is HIGH_LIMIT, and the HighLimitEnable flag of pLimitEnable is FALSE, then indicate a transition to the NORMAL event state.
(d) Optional: If pCurrentState is HIGH_LIMIT, and the LowLimitEnable flag of pLimitEnable is TRUE, and pMonitoredValue is less than pLowLimit for pTimeDelay, then indicate a transition to the LOW_LIMIT event state.
(e) If pCurrentState is HIGH_LIMIT, and pMonitoredValue is less than (pHighLimit - pDeadband) for pTimeDelayNormal, then indicate a transition to the NORMAL event state.

(f) If pCurrentState is LOW_LIMIT, and the LowLimitEnable flag of pLimitEnable is FALSE, then indicate a transition to the NORMAL event state.
(g) Optional: If pCurrentState is LOW_LIMIT, and the HighLimitEnable flag of pLimitEnable is TRUE, and pMonitoredValue is greater than pHighLimit for pTimeDelay, then indicate a transition to the HIGH_LIMIT event state.
(h) If pCurrentState is LOW_LIMIT, and pMonitoredValue is greater than (pLowLimit + pDeadband) for pTimeDelayNormal, then indicate a transition to the NORMAL event state.

If any of the optional conditions are supported, then all optional conditions shall be supported.

Figure 13-Y7 depicts those transitions of Figure 13-X3 that this event algorithm may indicate:

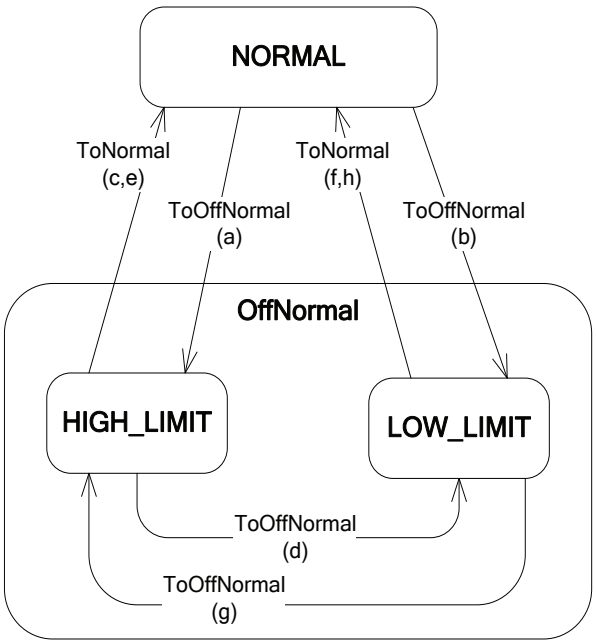

Figure 13-Y7. Transitions indicated by OUT_OF_RANGE algorithm

The notification parameters of this algorithm are:

Exceeding_Value	**This notification parameter, of type REAL, conveys the value of pMonitoredValue.**
Status_Flags	**This notification parameter, of type BACnetStatusFlags, conveys the value of pStatusFlags.**
Deadband	**This notification parameter, of type REAL, conveys the value of pDeadband.**
Exceeded_Limit	**This notification parameter, of type REAL, conveys the value of pLowLimit if** **a) the new state is LOW_LIMIT, or** **b) pCurrentState is LOW_LIMIT and the new state is NORMAL** **This notification parameter conveys the value of pHighLimit if** **a) the new state is HIGH_LIMIT, or** **b) pCurrentState is HIGH_LIMIT and the new state is NORMAL**

13.3.7 BUFFER_READY Event Algorithm

The BUFFER_READY event algorithm detects whether a defined number of records have been added to a log buffer since start of operation or the previous notification, whichever is most recent.

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 Unsigned32, represents the current total count of records in the log buffer referenced by pLogBuffer.**
pLogBuffer	**This parameter, of type BACnetDeviceObjectPropertyReference, represents the reference the log buffer property for which this algorithm is applied.**
pThreshold	**This parameter, of type Unsigned, represents the number of records that,**

when added to the log buffer, will result in a transition to NORMAL. If this parameter has a value of 0, then no transitions will be indicated by the algorithm.

pPreviousCount — **This parameter, of type Unsigned32, represents the value of pMonitoredValue at the time the most recent transition to NORMAL was indicated. Upon initialization of the event algorithm, this parameter shall be set to the value of pMonitoredValue. When a transition to NORMAL is indicated, this parameter shall be updated to the value of pMonitoredValue.**

The conditions evaluated by this event algorithm are:

(a) If pCurrentState is NORMAL, and pMonitoredValue is greater than or equal to pPreviousCount, and (pMonitoredValue - pPreviousCount) is greater than or equal to pThreshold and pThreshold is greater than 0, then indicate a transition to the NORMAL event state.
(b) If pCurrentState is NORMAL, and pMonitoredValue is less than pPreviousCount, and (pMonitoredValue - pPreviousCount + 2^{32} - 1) is greater than or equal to pThreshold and pThreshold is greater than 0, then indicate a transition to the NORMAL event state.

Figure 13-Y8 depicts those transitions of Figure 13-X3 that this event algorithm may indicate:

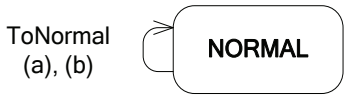

Figure 13-Y8. Transitions indicated by BUFFER_READY algorithm

The notification parameters of this algorithm are:

Buffer_Property — **This notification parameter, of type BACnetDeviceObjectPropertyReference, conveys the value of pLogBuffer.**

Previous_Notification — **This notification parameter, of type Unsigned32, conveys the value of pPreviousCount.**

Current_Notification — **This notification parameter, of type Unsigned32, conveys the value of pMonitoredValue.**

13.3.8 CHANGE_OF_LIFE_SAFETY Event Algorithm

The CHANGE_OF_LIFE_SAFETY event algorithm detects whether the monitored value equals a value that is listed as an alarm value or life safety alarm value. Event state transitions are also indicated if the value of the mode parameter changed since the last transition indicated. In this case, any time delays are overridden and the transition is indicated immediately.

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 BACnetLifeSafetyState, represents the current value of the monitored property.**

pMode — **This parameter, of type BACnetLifeSafetyMode, represents the current**

	life safety mode of operation.
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}.**
pOperationExpected	**This parameter, of type BACnetLifeSafetyOperation, represents the currently expected life safety operation.**
pAlarmValues	**This parameter, of type List Of BACnetPropertyStates, represents a list of values that are considered offnormal values.**
pLifeSafetyAlarmValues	**This parameter, of type List Of BACnetPropertyStates, represents a list of values that are considered life safety alarm values.**
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 are:

(a) If pCurrentState is NORMAL, and pMonitoredValue is equal to any of the values contained in pAlarmValues, and remains within the set of values of pAlarmValues either for pTimeDelay or for pMode changes, then indicate a transition to the OFFNORMAL event state.
(b) If pCurrentState is NORMAL, and pMonitoredValue is equal to any of the values contained in pLifeSafetyAlarmValues, and remains within the set of values of pLifeSafetyAlarmValues either for pTimeDelay or for pMode changes, then indicate a transition to the LIFE_SAFETY_ALARM event state.
(c) If pCurrentState is NORMAL, and pMode changes, then indicate a transition to the NORMAL event state.

(d) If pCurrentState is OFFNORMAL, and pMonitoredValue is not equal to any of the values contained in pAlarmValues and pLifeSafetyAlarmValues either for pTimeDelayNormal or for pMode changes, then indicate a transition to the NORMAL event state.
(e) If pCurrentState is OFFNORMAL, and pMonitoredValue is equal to any of the values contained in pLifeSafetyAlarmValues, and remains within the set of values of pLifeSafetyAlarmValues either for pTimeDelay or for pMode changes, then indicate a transition to the LIFE_SAFETY_ALARM event state.
(f) Optional: If pCurrentState is OFFNORMAL, and pMonitoredValue is equal to one of the values contained in pAlarmValues that is different from the value causing the last transition to OFFNORMAL, and remains equal to that value for pTimeDelay, then indicate a transition to the OFFNORMAL event state.
(g) If pCurrentState is OFFNORMAL, and pMode changes, then indicate a transition to the OFFNORMAL event state.

(h) If pCurrentState is LIFE_SAFETY_ALARM, and pMonitoredValue is not equal to any of the values contained in pAlarmValues and pLifeSafetyAlarmValues either for pTimeDelayNormal or for pMode changes, then indicate a transition to the NORMAL event state.
(i) If pCurrentState is LIFE_SAFETY_ALARM, and pMonitoredValue is equal to any of the values contained in pAlarmValues, and remains within the set of values of pAlarmValues either

for pTimeDelay or for pMode changes, then indicate a transition to the OFFNORMAL event state.

(j) Optional: If pCurrentState is LIFE_SAFETY_ALARM, and pMonitoredValue is equal to one of the values contained in pLifeSafetyAlarmValues that is different from the value causing the last transition to LIFE_SAFETY_ALARM, and remains equal to that value for pTimeDelay, then indicate a transition to the LIFE_SAFETY_ALARM event state.

(k) If pCurrentState is LIFE_SAFETY_ALARM, and pMode changes, then indicate a transition to the LIFE_SAFETY_ALARM event state.

If any of the optional conditions are supported, then all optional conditions shall be supported.

Figure 13-Y9 depicts those transitions of Figure 13-X3 that this event algorithm may indicate:

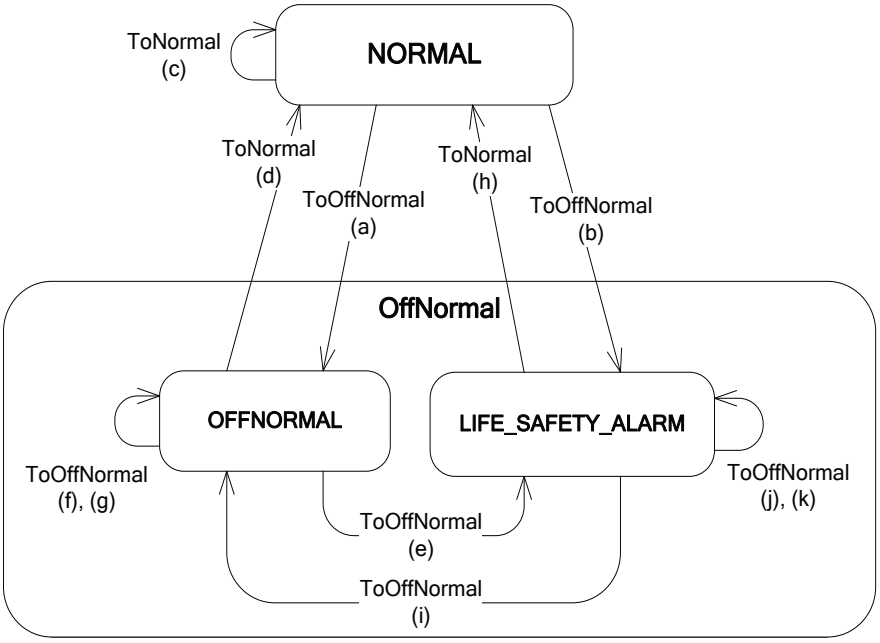

Figure 13-Y9. Transitions indicated by CHANGE_OF_LIFE_SAFETY algorithm

The notification parameters of this algorithm are:

New_State	**This notification parameter, of type BACnetLifeSafetyState, conveys the value of pMonitoredValue.**
New_Mode	**This notification parameter, of type BACnetLifeSafetyMode, conveys the value of pMode.**
Status_Flags	**This notification parameter, of type BACnetStatusFlags, conveys the value of pStatusFlags.**
Operation_Expected	**This notification parameter, of type BACnetLifeSafetyOperation, conveys the value of pOperationExpected.**

13.3.9UNSIGNED_RANGE Event Algorithm

The UNSIGNED_RANGE event algorithm detects whether the monitored value exceeds a range defined by a high limit and a low limit.

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 Unsigned, 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}.**
pLowLimit	**This parameter, of type Unsigned, represents the lower limit of the range considered normal.**
pHighLimit	**This parameter, of type Unsigned, represents the higher limit of the range considered normal.**
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 provided for this parameter, then it takes on the value of the pTimeDelay parameter.**

The conditions evaluated by this event algorithm are:

(a) If pCurrentState is NORMAL, and pMonitoredValue is greater than pHighLimit for pTimeDelay, then indicate a transition to the HIGH_LIMIT event state.
(b) If pCurrentState is NORMAL, and pMonitoredValue is less than pLowLimit for pTimeDelay, then indicate a transition to the LOW_LIMIT event state.

(c) Optional: If pCurrentState is HIGH_LIMIT, and pMonitoredValue is less than pLowLimit for pTimeDelay, then indicate a transition to the LOW_LIMIT event state.
(d) If pCurrentState is HIGH_LIMIT, and pMonitoredValue is equal to or less than pHighLimit for pTimeDelayNormal, then indicate a transition to the NORMAL event state.

(e) Optional: If pCurrentState is LOW_LIMIT, and pMonitoredValue is greater than pHighLimit for pTimeDelay, then indicate a transition to the HIGH_LIMIT event state.
(f) If pCurrentState is LOW_LIMIT, and pMonitoredValue is equal to or greater than pLowLimit, for pTimeDelayNormal, then indicate a transition to the NORMAL event state.

If any of the optional conditions are supported, then all optional conditions shall be supported.

Figure 13-Y10 depicts those transitions of Figure 13-X3 that this event algorithm may indicate:

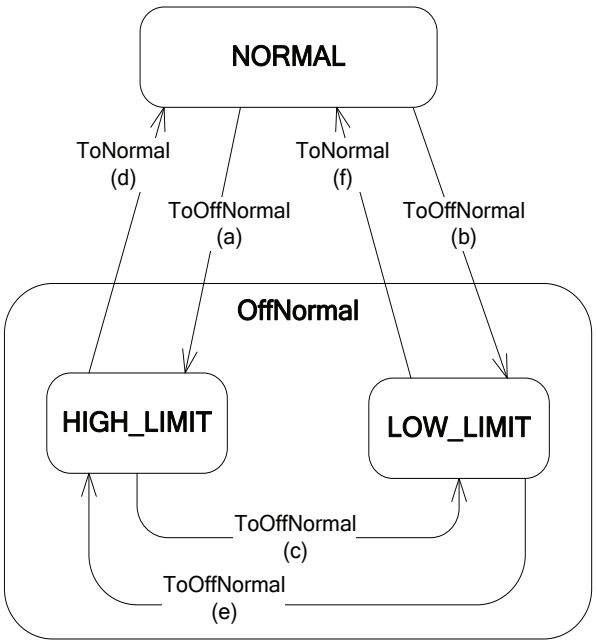

Figure 13-Y10. Transitions indicated by UNSIGNED_RANGE algorithm

The notification parameters of this algorithm are:

Exceeding_Value **This notification parameter, of type Unsigned, conveys the value of pMonitoredValue.**

Status_Flags **This notification parameter, of type BACnetStatusFlags, conveys the value of pStatusFlags.**

Exceeded_Limit **This notification parameter, of type Unsigned, conveys the value of pLowLimit if**
a) the new state is LOW_LIMIT, or
b) pCurrentState is LOW_LIMIT and the new state is NORMAL
This notification parameter conveys the value of pHighLimit if
a) the new state is HIGH_LIMIT, or
b) pCurrentState is HIGH_LIMIT and the new state is NORMAL

13.3.10 EXTENDED Event Algorithm

The EXTENDED event algorithm detects event conditions based on a proprietary event algorithm. The proprietary event algorithm uses parameters and conditions defined by the vendor. The algorithm is identified by a vendor-specific event type that is in the scope of the vendor's vendor identification code. The algorithm may, at the vendor's discretion, indicate a new event state, a transition to the same event state, or no transition to the Event-State-Detection. The indicated new event states may be NORMAL, and any OffNormal event state. FAULT event state may not be indicated by this algorithm. For the purpose of proprietary evaluation of unreliability conditions that may result in FAULT event state, a FAULT_EXTENDED fault algorithm shall be used.

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.**

pVendorId **This parameter, of type Unsigned16, represents the vendor identification**

code for the event type of the vendor-proprietary event algorithm.

pEventType — **This parameter, of type Unsigned, represents the vendor-proprietary event algorithm.**

pParameters — **This parameter is a sequence of primitive or constructed values that represent algorithm parameters whose interpretation is specific to the proprietary event algorithm.**
Values of this parameter may be used to provide values of the Parameters notification parameter.

The conditions evaluated by this event algorithm are vendor-specific, and are identified by pVendorId and pEventType. This algorithm may support multiple offnormal event states, including proprietary offnormal event states.

Figure 13-Y11 depicts those transitions of Figure 13-X3 that this event algorithm may indicate. The particular offnormal states shown are for illustration only.

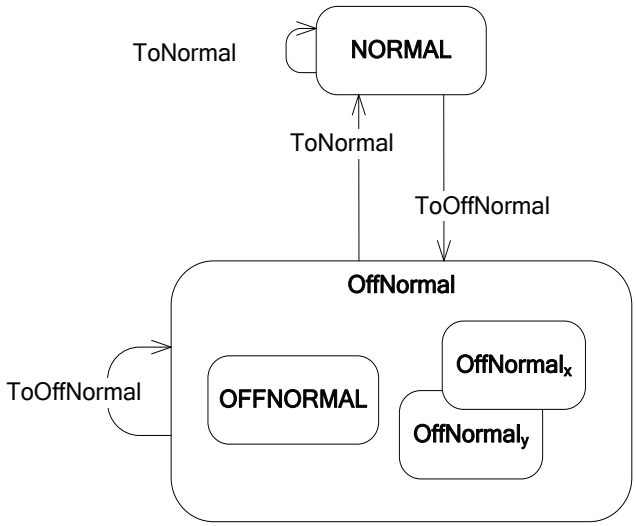

Figure 13-Y11. Transitions indicated by EXTENDED algorithm

The notification parameters of this algorithm are:

Vendor_Id — **This notification parameter, of type Unsigned16, conveys the value of pVendorId.**

Extended_Event_Type — **This notification parameter, of type Unsigned, conveys the value of pEventType.**

Parameters — **This notification parameter is a sequence of primitive or constructed values whose content and interpretation is specific to the proprietary event algorithm.**

13.3.11 CHANGE_OF_STATUS_FLAGS Event Algorithm

The CHANGE_OF_STATUS_FLAGS event algorithm detects whether a significant flag of the monitored value of type BACnetStatusFlags has the value TRUE.

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 BACnetStatusFlags, represents the current value of the monitored property.**
pSelectedFlags	**This parameter, of type BACnetStatusFlags, represents the flags of pMonitoredValue that are selected to be significant for evaluation. A value of TRUE in a flag indicates that the corresponding flag in pMonitoredValue is significant for evaluation. A value of FALSE in a flag indicates that the corresponding flag in pMonitoredValue is not significant for evaluation.**
pPresentValue	**This optional parameter, of type ABSTRACT-SYNTAX.&Type, represents the current value of the Present_Value property of the object containing the property that provides the value of the pMonitoredValue parameter. This parameter may be omitted if it is determined to be too large. The method of such determination is a local matter.**
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 provided for this parameter, then it takes on the value of the pTimeDelay parameter.**

The conditions evaluated by this event algorithm are:

(a) If pCurrentState is NORMAL, and pMonitoredValue has a value of TRUE in any of its flags that also has a value of TRUE in the corresponding flag in pSelectedFlags for pTimeDelay, then indicate a transition to the OFFNORMAL event state.
(b) If pCurrentState is OFFNORMAL, and pMonitoredValue has none of its flags set to TRUE that also has a value of TRUE in the corresponding flag in the pSelectedFlags event parameter for pTimeDelayNormal, then indicate a transition to the NORMAL event state.
(c) If pCurrentState is OFFNORMAL, and the set of selected flags of pMonitoredValue that have a value of TRUE changes, then indicate a transition to the OFFNORMAL event state.

Figure 13-Y12 depicts those transitions of Figure 13-X3 that this event algorithm may indicate:

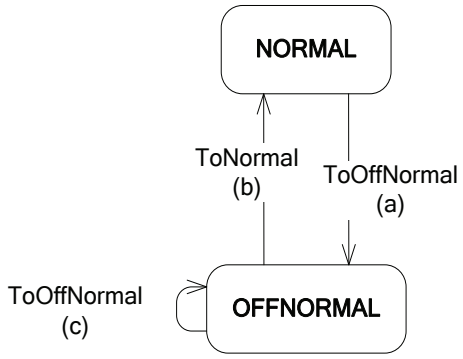

Figure 13-Y12. Transitions indicated by CHANGE_OF_STATUS_FLAGS algorithm

The notification parameters of this algorithm are:

Present_Value	**This optional notification parameter, of type ABSTRACT-SYNTAX.&Type, conveys the value of pPresentValue. This parameter is optional and may be omitted if it is not provided to the algorithm, or if it is determined to be too large. The method of such determination is a local matter.**

Referenced_Flags	**This notification parameter, of type BACnetStatusFlags, conveys the value of pMonitoredValue.**

13.3.12 ACCESS_EVENT Event Algorithm

The ACCESS_EVENT event algorithm detects whether the access event time has changed and the new access event value equals a value that is listed to cause a transition to NORMAL.

For detection of change, the access event time 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 access event time 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 BACnetAccessEvent, 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}.**
pAccessEvents	**This parameter, of type List Of BACnetAccessEvent, represents a list of values that are considered access event values that shall cause a transition indication.**
pAccessEventTag	**This parameter, of type Unsigned, represents the current value of the Access_Event_Tag property of the object containing the property that provides the value of the pMonitoredValue parameter.**
pAccessEventTime	**This parameter, of type BACnetTimeStamp, represents the update time of the monitored access event value.**
pAccessCredential	**This parameter, of type BACnetDeviceObjectReference, represents the current value of the Access_Event_Credential property of the object containing the property that provides the value of the pMonitoredValue parameter.**
pAccessFactor	**This optional parameter, of type BACnetDeviceObjectReference, represents the current value of the Access_Event_Authentication_Factor property of the object containing the property that provides the value of the pMonitoredValue parameter. This parameter may be omitted if it is determined to be too large, or omitted for security reasons. The method of such determination is a local matter.**

The conditions evaluated by this event algorithm are:

(a) If pCurrentState is NORMAL, and pAccessEventTime changes, and pMonitoredValue is equal to any of the values contained in pAccessEvents, then indicate a transition to the NORMAL event state.

Figure 13-Y13 depicts those transitions of Figure 13-X3 that this event algorithm may indicate:

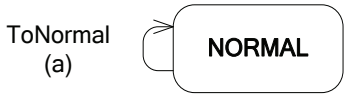

Figure 13-Y13. Transitions indicated by ACCESS_EVENT algorithm

The notification parameters of this algorithm are:

Access_Event	**This notification parameter, of type BACnetAccessEvent, conveys the value of pMonitoredValue.**
Status_Flags	**This notification parameter, of type BACnetStatusFlags, conveys the value of pStatusFlags.**
Access_Event_Tag	**This notification parameter, of type Unsigned, conveys the value of pAccessEventTag.**
Access_Event_Time	**This notification parameter, of type BACnetTimeStamp, conveys the value of pAccessEventTime.**
Access_Credential	**This notification parameter, of type BACnetDeviceObjectReference, conveys the value of pAccessCredential.**
Authentication_Factor	**This optional notification parameter, of type BACnetAuthenticationFactor, conveys the value of pAccessFactor. This parameter may be omitted if it is not provided to the algorithm, is determined to be too large for the event notification, or omitted for security reasons. The method of such determination is a local matter.**

13.3.13 DOUBLE_OUT_OF_RANGE Event Algorithm

The DOUBLE_OUT_OF_RANGE event algorithm detects whether the monitored value exceeds a range defined by a high limit and a low limit. Each of these limits may be enabled or disabled. If disabled, the normal range has no lower limit or no higher limit respectively. In order to reduce jitter of the resulting event state, a deadband is applied when the value is in the process of returning to the normal range.

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 Double, 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}.**
pLowLimit	**This parameter, of type Double, represents the lower limit of the range considered normal**
pHighLimit	**This parameter, of type Double, represents the higher limit of the range considered normal.**

pDeadband	**This parameter, of type Double, represents the deadband that is applied to the respective limit before a return to Normal event state is indicated.**
pLimitEnable	**This parameter, of type BACnetLimitEnable, represents two flags, HighLimitEnable and LowLimitEnable, that separately enable (TRUE) or disable (FALSE) the respective limits applied by the event algorithm. If the value of this parameter is not provided, then both flags shall be set to TRUE (1).**
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 provided for this parameter, then it takes on the value of the pTimeDelay parameter.**

The conditions evaluated by this event algorithm are:

(a) If pCurrentState is NORMAL, and the HighLimitEnable flag of pLimitEnable is TRUE, and pMonitoredValue is greater than pHighLimit for pTimeDelay, then indicate a transition to the HIGH_LIMIT event state.
(b) If pCurrentState is NORMAL, and the LowLimitEnable flag of pLimitEnable is TRUE, and pMonitoredValue is less than pLowLimit for pTimeDelay, then indicate a transition to the LOW_LIMIT event state.

(c) If pCurrentState is HIGH_LIMIT, and the HighLimitEnable flag of pLimitEnable is FALSE, then indicate a transition to the NORMAL event state.
(d) Optional: If pCurrentState is HIGH_LIMIT, and the LowLimitEnable flag of pLimitEnable is TRUE, and pMonitoredValue is less than pLowLimit for pTimeDelay, then indicate a transition to the LOW_LIMIT event state.
(e) If pCurrentState is HIGH_LIMIT, and pMonitoredValue is less than (pHighLimit - pDeadband) for pTimeDelayNormal, then indicate a transition to the NORMAL event state.

(f) If pCurrentState is LOW_LIMIT, and the LowLimitEnable flag of pLimitEnable is FALSE, then indicate a transition to the NORMAL event state.
(g) Optional: If pCurrentState is LOW_LIMIT, and the HighLimitEnable flag of pLimitEnable is TRUE, and pMonitoredValue is greater than pHighLimit for pTimeDelay, then indicate a transition to the HIGH_LIMIT event state.
(h) If pCurrentState is LOW_LIMIT, and pMonitoredValue is greater than (pLowLimit + pDeadband) for pTimeDelayNormal, then indicate a transition to the NORMAL event state.

If any of the optional conditions are supported, then all optional conditions shall be supported.

Figure 13-Y14 depicts those transitions of Figure 13-X3 that this event algorithm may indicate:

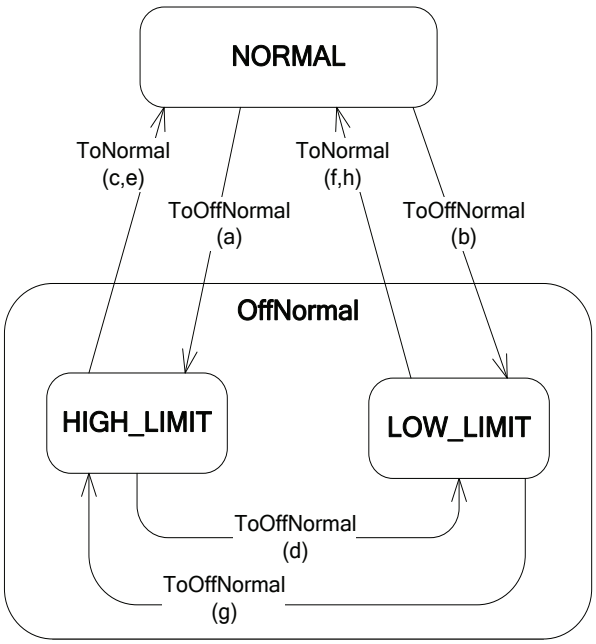

Figure 13-Y14. Transitions indicated by DOUBLE_OUT_OF_RANGE algorithm

The notification parameters of this algorithm are:

Exceeding_Value	**This notification parameter, of type Double, conveys the value of pMonitoredValue.**
Status_Flags	**This notification parameter, of type BACnetStatusFlags, conveys the value of pStatusFlags.**
Deadband	**This notification parameter, of type Double, conveys the value of pDeadband.**
Exceeded_Limit	**This notification parameter, of type Double, conveys the value of pLowLimit if** **a) the new state is LOW_LIMIT, or** **b) pCurrentState is LOW_LIMIT and the new state is NORMAL** **This notification parameter conveys the value of pHighLimit if** **a) the new state is HIGH_LIMIT, or** **b) pCurrentState is HIGH_LIMIT and the new state is NORMAL**

13.3.14 SIGNED_OUT_OF_RANGE Event Algorithm

The SIGNED_OUT_OF_RANGE event algorithm detects whether the monitored value exceeds a range defined by a high limit and a low limit. Each of these limits may be enabled or disabled. If disabled, the normal range has no lower limit or no higher limit respectively. In order to reduce jitter of the resulting event state, a deadband is applied when the value is in the process of returning to the normal range.

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 INTEGER, 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}.
pLowLimit	**This parameter, of type INTEGER, represents the lower limit of the range considered normal.**
pHighLimit	**This parameter, of type INTEGER, represents the higher limit of the range considered normal.**
pDeadband	**This parameter, of type Unsigned, represents the deadband that is applied to the respective limit before a return to Normal event state is indicated.**
pLimitEnable	**This parameter, of type BACnetLimitEnable, represents two flags, HighLimitEnable and LowLimitEnable, that separately enable (TRUE) or disable (FALSE) the respective limits applied by the event algorithm. If the value of this parameter is not provided, then both flags shall be set to TRUE (1).**
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 provided for this parameter, then it takes on the value of the pTimeDelay parameter.**

The conditions evaluated by this event algorithm are:

- **(a) If pCurrentState is NORMAL, and the HighLimitEnable flag of pLimitEnable is TRUE, and pMonitoredValue is greater than pHighLimit for pTimeDelay, then indicate a transition to the HIGH_LIMIT event state.**
- **(b) If pCurrentState is NORMAL, and the LowLimitEnable flag of pLimitEnable is TRUE, and pMonitoredValue is less than pLowLimit for pTimeDelay, then indicate a transition to the LOW_LIMIT event state.**
- **(c) If pCurrentState is HIGH_LIMIT, and the HighLimitEnable flag of pLimitEnable is FALSE, then indicate a transition to the NORMAL event state.**
- **(d) Optional: If pCurrentState is HIGH_LIMIT, and the LowLimitEnable flag of pLimitEnable is TRUE, and pMonitoredValue is less than pLowLimit for pTimeDelay, then indicate a transition to the LOW_LIMIT event state.**
- **(e) If pCurrentState is HIGH_LIMIT, and pMonitoredValue is less than (pHighLimit - pDeadband) for pTimeDelayNormal, then indicate a transition to the NORMAL event state.**
- **(f) If pCurrentState is LOW_LIMIT, and the LowLimitEnable flag of pLimitEnable is FALSE, then indicate a transition to the NORMAL event state.**
- **(g) Optional: If pCurrentState is LOW_LIMIT, and the HighLimitEnable flag of pLimitEnable is TRUE, and pMonitoredValue is greater than pHighLimit for pTimeDelay, then indicate a transition to the HIGH_LIMIT event state.**
- **(h) If pCurrentState is LOW_LIMIT, and pMonitoredValue is greater than (pLowLimit + pDeadband) for pTimeDelayNormal, then indicate a transition to the NORMAL event state.**

If any of the optional conditions are supported, then all optional conditions shall be supported.

Figure 13-Y15 depicts those transitions of Figure 13-X3 that this event algorithm may indicate:

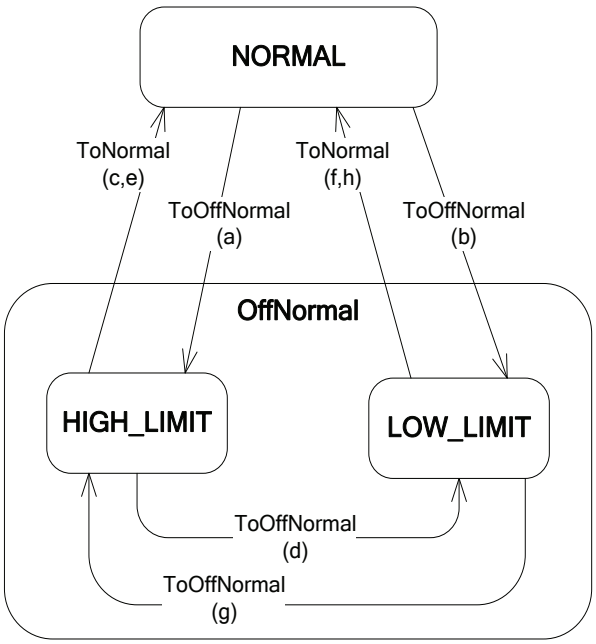

Figure 13-Y15: Transitions indicated by SIGNED_OUT_OF_RANGE algorithm

The notification parameters of this algorithm are:

Exceeding_Value	**This notification parameter, of type INTEGER, conveys the value of pMonitoredValue.**
Status_Flags	**This notification parameter, of type BACnetStatusFlags, conveys the value of pStatusFlags.**
Deadband	**This notification parameter, of type Unsigned, conveys the value of pDeadband.**
Exceeded_Limit	**This notification parameter, of type INTEGER, conveys the value of pLowLimit if** **a) the new state is LOW_LIMIT, or** **b) pCurrentState is LOW_LIMIT and the new state is NORMAL** **This notification parameter conveys the value of pHighLimit if** **a) the new state is HIGH_LIMIT, or** **b) pCurrentState is HIGH_LIMIT and the new state is NORMAL**

13.3.15 UNSIGNED_OUT_OF_RANGE Event Algorithm

The UNSIGNED_OUT_OF_RANGE event algorithm detects whether the monitored value exceeds a range defined by a high limit and a low limit. Each of these limits may be enabled or disabled. If disabled, the normal range has no lower limit or no higher limit respectively. In order to reduce jitter of the resulting event state, a deadband is applied when the value is in the process of returning to the normal range.

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 Unsigned, 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}.**
pLowLimit	**This parameter, of type Unsigned, represents the lower limit of the range considered normal.**
pHighLimit	**This parameter, of type Unsigned, represents the higher limit of the range considered normal.**
pDeadband	**This parameter, of type Unsigned, represents the deadband that is applied to the respective limit before a return to Normal event state is indicated.**
pLimitEnable	**This parameter, of type BACnetLimitEnable, represents two flags, HighLimitEnable and LowLimitEnable, that separately enable (TRUE) or disable (FALSE) the respective limits applied by the event algorithm. If the value of this parameter is not provided, then both flags shall be set to TRUE (1).**
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 provided for this parameter, then it takes on the value of the pTimeDelay parameter.**

The conditions evaluated by this event algorithm are:

- **(a) If pCurrentState is NORMAL, and the HighLimitEnable flag of pLimitEnable is TRUE, and pMonitoredValue is greater than pHighLimit for pTimeDelay, then indicate a transition to the HIGH_LIMIT event state.**
- **(b) If pCurrentState is NORMAL, and the LowLimitEnable flag of pLimitEnable is TRUE, and pMonitoredValue is less than pLowLimit for pTimeDelay, then indicate a transition to the LOW_LIMIT event state.**
- **(c) If pCurrentState is HIGH_LIMIT, and the HighLimitEnable flag of pLimitEnable is FALSE, then indicate a transition to the NORMAL event state.**
- **(d) Optional: If pCurrentState is HIGH_LIMIT, and the LowLimitEnable flag of pLimitEnable is TRUE, and pMonitoredValue is less than pLowLimit for pTimeDelay, then indicate a transition to the LOW_LIMIT event state.**
- **(e) If pCurrentState is HIGH_LIMIT, and pMonitoredValue is less than (pHighLimit - pDeadband) for pTimeDelayNormal, then indicate a transition to the NORMAL event state.**
- **(f) If pCurrentState is LOW_LIMIT, and the LowLimitEnable flag of pLimitEnable is FALSE, then indicate a transition to the NORMAL event state.**
- **(g) Optional: If pCurrentState is LOW_LIMIT, and the HighLimitEnable flag of pLimitEnable is TRUE, and pMonitoredValue is greater than pHighLimit for pTimeDelay, then indicate a transition to the HIGH_LIMIT event state.**
- **(h) If pCurrentState is LOW_LIMIT, and pMonitoredValue is greater than (pLowLimit + pDeadband) for pTimeDelayNormal, then indicate a transition to the NORMAL event state.**

If any of the optional conditions are supported, then all optional conditions shall be supported.

Figure 13-Y16 depicts those transitions of Figure 13-X3 that this event algorithm may indicate:

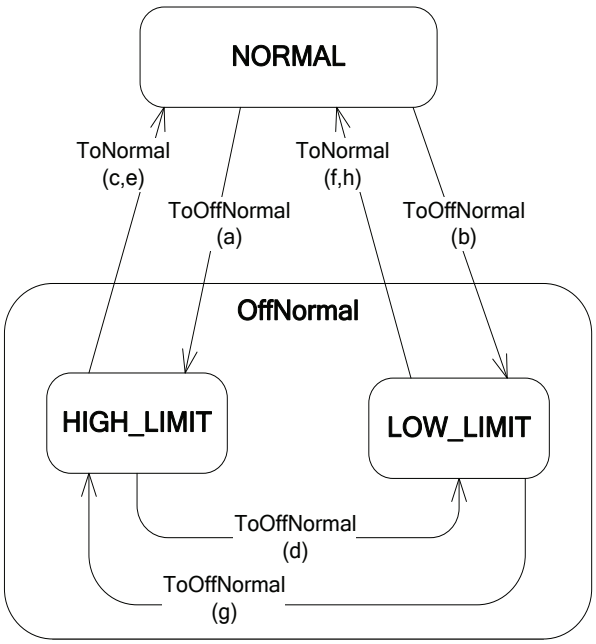

Figure 13-Y16. Transitions indicated by UNSIGNED_OUT_OF_RANGE algorithm

The notification parameters of this algorithm are:

Exceeding_Value	**This notification parameter, of type Unsigned, conveys the value of pMonitoredValue.**
Status_Flags	**This notification parameter, of type BACnetStatusFlags, conveys the value of pStatusFlags.**
Deadband	**This notification parameter, of type Unsigned, conveys the value of pDeadband.**
Exceeded_Limit	**This notification parameter, of type Unsigned, conveys the value of pLowLimit if** **a) the new state is LOW_LIMIT, or** **b) pCurrentState is LOW_LIMIT and the new state is NORMAL** **This notification parameter conveys the value of pHighLimit if** **a) the new state is HIGH_LIMIT, or** **b) pCurrentState is HIGH_LIMIT and the new state is NORMAL**

13.3.16 CHANGE_OF_CHARACTERSTRING Event Algorithm

The CHANGE_OF_CHARACTERSTRING event algorithm detects whether the monitored value matches a character string that is listed as an alarm value. Alarm values are of type BACnetOptionalCharacterString, and may also be NULL or an empty character string.

A "match" of the monitored value with an alarm value is defined as follows:

(a) If the alarm value string is NULL, then it is not considered a match.
(b) If the alarm value string is empty (of zero length), then it is considered a match if and only if the monitored value is also an empty string.
(c) If the alarm value string is not empty, then it is considered a match if the alarm value string appears in any position within the monitored value string. For character-matching purposes, character case shall be significant, and so a match must be an exact match character by character.

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 CharacterString, 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}.**
pAlarmValues	**This parameter, of type List Of BACnetOptionalCharacterString, represents a list of character strings that are considered alarm values.**
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 are:

(a) If pCurrentState is NORMAL, and pMonitoredValue matches any of the values contained in pAlarmValues for pTimeDelay, then indicate a transition to the OFFNORMAL event state.
(b) If pCurrentState is OFFNORMAL, and pMonitoredValue does not match any of the values contained in pAlarmValues for pTimeDelayNormal, then indicate a transition to the NORMAL event state.
(c) If pCurrentState is OFFNORMAL, and pMonitoredValue matches one of the values contained in pAlarmValues that is different from the value that caused the last transition to OFFNORMAL, and remains equal to that value for pTimeDelay, then indicate a transition to the OFFNORMAL event state.

Figure 13-Y17 depicts those transitions of Figure 13-X3 that this event algorithm may indicate:

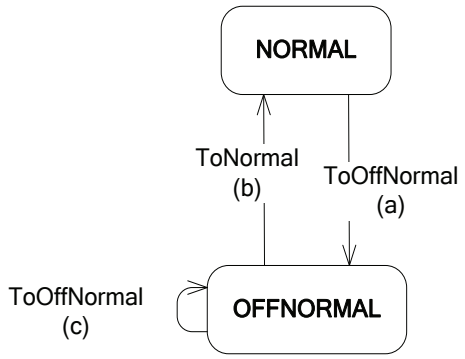

Figure 13-Y17. Transitions indicated by CHANGE_OF_CHARACTERSTRING algorithm

The notification parameters for this algorithm are:

Changed_Value	**This notification parameter, of type CharacterString, conveys the value of pMonitoredValue.**
Status_Flags	**This notification parameter, of type BACnetStatusFlags, conveys the value of**

pStatusFlags.

Alarm_Value — **This notification parameter, of type CharacterString, conveys the character string of pAlarmValues related to the event state transition reported:**

(a) for transitions to OFFNORMAL, the character string of pAlarmValues that matches pMonitoredValue,

(b) for transitions to NORMAL, the character string of pAlarmValues that match pMonitoredValue at the time of the most recent transition to OFFNORMAL.

13.3.17 NONE Event Algorithm

This event algorithm has no parameters, no conditions, and does not indicate any transitions of event state.

The NONE algorithm is used when only fault detection is in use by an object.

[Replace Clause 13.4 and its sub-clause, p. 304]

13.4 Fault Algorithms

Certain object types may optionally support a fault algorithm which has externally visible inputs and is performed as part of the object's reliability-evaluation process. This clause defines the standard fault algorithms. To determine which algorithm is applied by which object type, see the object type definitions in Clause 12.

Table 13-Y2 lists the fault algorithms that are specified in this standard. The fault algorithms are indicated by the BACnetFaultType value of the same name.

Table 13-Y2. 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 algorithms monitor a value and evaluate whether the condition for transition of reliability exists. The result of the evaluation, indicated to the reliability-evaluation process, may be a transition to a new reliability, a transition to the same reliability, or no transition. The final determination of the Reliability property value is the responsibility of the reliability-evaluation process and is subject to additional conditions. See Clause 13.2.

Each of the fault algorithms defines its input parameters, the allowable reliability values, and the conditions for transitions between those values.

When evaluating the monitored value, all conditions defined for the algorithm shall be evaluated in the order as presented for the algorithm. Some algorithms specify optional conditions, marked as "Optional:" Whether or not an implementation uses these conditions is a local matter. If no condition evaluates to true, then no transition shall be indicated to the reliability-evaluation process.

13.4.1NONE Fault Algorithm

The NONE fault algorithm is a placeholder for the case where no fault algorithm is applied by the object.

This fault algorithm has no parameters, no conditions, and does not indicate any transitions of reliability.

13.4.2FAULT_CHARACTERSTRING Fault Algorithm

The FAULT_CHRACTERSTRING event algorithm detects whether the monitored value matches a character string that is listed as a fault value. Fault values are of type BACnetOptionalCharacterString and may also be NULL or an empty character string.

A "match" of the monitored value with a fault value is defined as follows:

(a) If the fault value is NULL, then it is not considered a match.
(b) If the fault value string is empty (of zero length), then it is considered a match if and only if the monitored value is also an empty string.
(c) If the fault value string is not empty, then it is considered a match if the fault value string appears in any position within the monitored value string. For character-matching purposes, character case shall be significant, and so a match must be an exact match character by character.

The parameters of this fault algorithm are:

pCurrentReliability	**This parameter, of type BACnetReliability, represents the current value of the Reliability property of the object that applies the fault algorithm.**
pMonitoredValue	**This parameter, of type CharacterString, represents the value monitored by this algorithm.**
pFaultValues	**This parameter, of type List Of BACnetOptionalCharacterString, represents a list of character strings that are considered fault values. This parameter shall not contain string values that are present in the pAlarmValues parameter of the CHANGE_OF_CHARACTERSTRING algorithm performed by the same object. NULL values may be present in this parameter regardless of the content of pAlarmValues.**

The conditions evaluated by this fault algorithm are:

(a) If pCurrentReliability is NO_FAULT_DETECTED, and pMonitoredValue matches one of the values in pFaultValues, then indicate a transition to the MULTI_STATE_FAULT reliability.
(b) If pCurrentReliability is MULTI_STATE_FAULT, and pMonitoredValue does not match any of the values contained in pFaultValues, then indicate a transition to the NO_FAULT_DETECTED reliability.
(c) Optional: If pCurrentReliability is MULTI_STATE_FAULT, and pMonitoredValue matches one of the values contained in pFaultValues that is different from the value that caused the last transition to MULTI_STATE_FAULT, then indicate a transition to the MULTI_STATE_FAULT reliability.

Figure 13-Y18 depicts the reliability transitions that this fault algorithm may indicate:

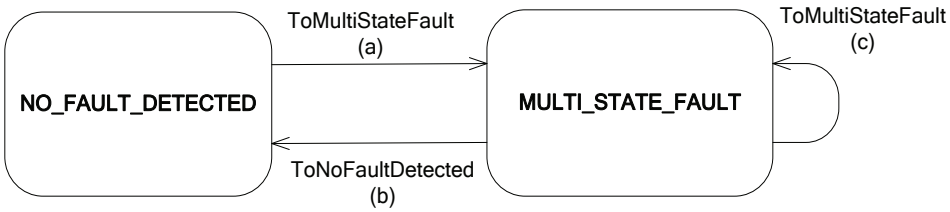

Figure 13-Y18: Transitions indicated by FAULT_CHARACTERSTRING algorithm

13.4.3FAULT_EXTENDED Fault Algorithm

The FAULT_EXTENDED fault algorithm detects fault conditions based on a proprietary fault algorithm. The proprietary fault algorithm uses parameters and conditions defined by the vendor. The algorithm is identified by a vendor-specific fault type that is in the scope of the vendor's vendor identification code. The algorithm may, at the vendor's discretion, indicate a new reliability, a transition to the same reliability, or no transition to the reliability-evaluation process.

The parameters of this fault algorithm are:

pCurrentReliability	**This parameter, of type BACnetReliability, represents the current value of the Reliability property of the object that applies the fault algorithm.**
pVendorId	**This parameter, of type Unsigned16, represents the vendor identification code for the fault type of the vendor-proprietary fault algorithm.**
pFaultType	**This parameter, of type Unsigned, represents the vendor-proprietary fault algorithm.**
pParameters	**This parameter represents a sequence of primitive or constructed values whose interpretation is specific to the proprietary fault algorithm.**

The conditions evaluated and transitions indicated by this fault algorithm are vendor-specific and are identified by pVendorId and pFaultType.

Figure 13-Y19 depicts the reliability transitions that this fault algorithm may indicate. The particular unreliable values shown are for illustration only.

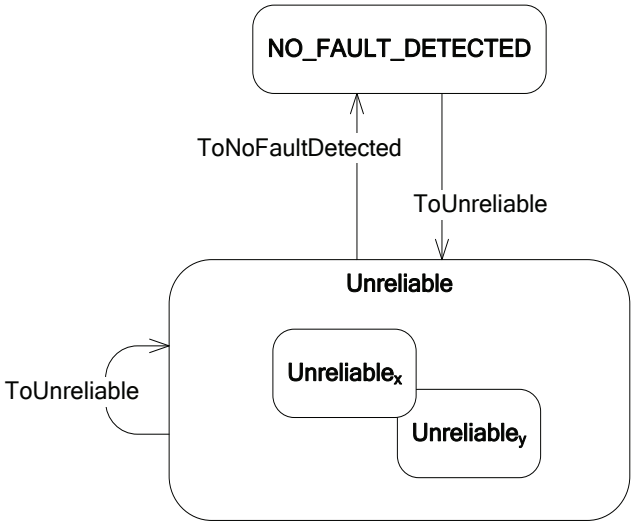

Figure 13-Y19. Transitions indicated by FAULT_EXTENDED algorithm

13.4.4FAULT_LIFE_SAFETY Fault Algorithm

The FAULT_LIFE_SAFETY fault algorithm detects whether the monitored value equals a value that is listed as a fault value. The monitored value is of type BACnetLifeSafetyState. If internal operational reliability is unreliable, then the internal reliability takes precedence over evaluation of the monitored value.

In addition, this algorithm monitors a life safety mode value. If reliability is MULTI_STATE_FAULT, then new transitions to MULTI_STATE_FAULT are indicated upon change of the mode value.

The parameters of this fault algorithm are:

pCurrentReliability	**This parameter, of type BACnetReliability, represents the current value of the Reliability property of the object that applies the fault algorithm.**
pMonitoredValue	**This parameter, of type BACnetLifeSafetyState, represents the value monitored by this algorithm.**
pMode	**This parameter, of type BACnetLifeSafetyMode, represents the life safety mode value monitored by this algorithm.**
pFaultValues	**This parameter, of type List Of BACnetLifeSafetyState, represents a list of values that are considered fault values. This parameter shall not contain values that are present in the pAlarmValues parameter of the associated CHANGE_OF_LIFE_SAFETY algorithm performed by the same object.**

The conditions evaluated by this fault algorithm are:

(a) If pCurrentReliability is NO_FAULT_DETECTED, and pMonitoredValue is equal to any of the values in pFaultValues, then indicate a transition to the MULTI_STATE_FAULT reliability.
(b) If pCurrentReliability is MULTI_STATE_FAULT, and pMonitoredValue is not equal to any of the values contained in pFaultValues, then indicate a transition to the NO_FAULT_DETECTED reliability
(c) If pCurrentReliability is MULTI_STATE_FAULT, and pMonitoredValue is equal to any of the values contained in pFaultValues, and pMode has changed since the last transition to

MULTI_STATE_FAULT, then indicate a transition to the MULTI_STATE_FAULT reliability.

(d) Optional: If pCurrentReliability is MULTI_STATE_FAULT, and pMonitoredValue is equal to one of the values contained in pFaultValues that is different from the value causing the last transition to MULTI_STATE_FAULT, then indicate a transition to the MULTI_STATE_FAULT reliability.

Figure 13-Y20 depicts the reliability transitions that this fault algorithm may indicate:

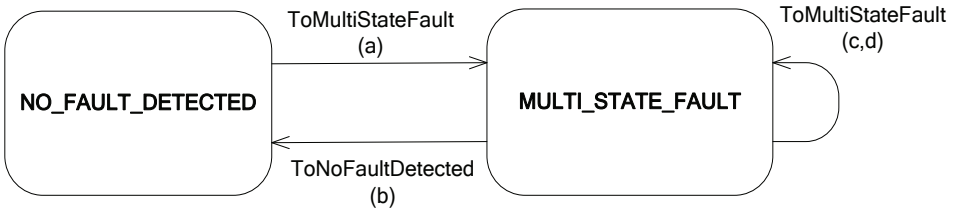

Figure 13-Y20. Transitions indicated by FAULT_LIFE_SAFETY algorithm

13.4.5FAULT_STATE Fault Algorithm

The FAULT_STATE fault algorithm detects whether the monitored value equals a value that is listed as a fault value. The monitored value may be of any discrete or enumerated data type, including Boolean. If internal operational reliability is unreliable, then the internal reliability takes precedence over evaluation of the monitored value.

The parameters of this fault algorithm are:

pCurrentReliability	**This parameter, of type BACnetReliability, represents the current value of the Reliability property of the object that applies the fault algorithm.**
pMonitoredValue	**This parameter is a discrete value that represents the current value of the monitored property. The datatype of the value of this parameter shall be one of the options of BACnetPropertyStates.**
pFaultValues	**This parameter is a list of discrete values that represent the fault values. The datatype of the values of this parameter and of pMonitoredValue shall be the same.**

The conditions evaluated by this fault algorithm are:

(a) If pCurrentReliability is NO_FAULT_DETECTED, and pMonitoredValue is equal to any of the values in pFaultValues, then indicate a transition to the MULTI_STATE_FAULT reliability.

(b) If pCurrentReliability is MULTI_STATE_FAULT, and pMonitoredValue is not equal to any of the values contained in pFaultValues, then indicate a transition to the NO_FAULT_DETECTED reliability.

(c) Optional: If pCurrentReliability is MULTI_STATE_FAULT, and pMonitoredValue is equal one of the values contained in pFaultValues that is different from the value that caused the last transition to MULTI_STATE_FAULT, then indicate a transition to the MULTI_STATE_FAULT reliability.

Figure 13-Y21 depicts the reliability transitions that this fault algorithm may indicate:

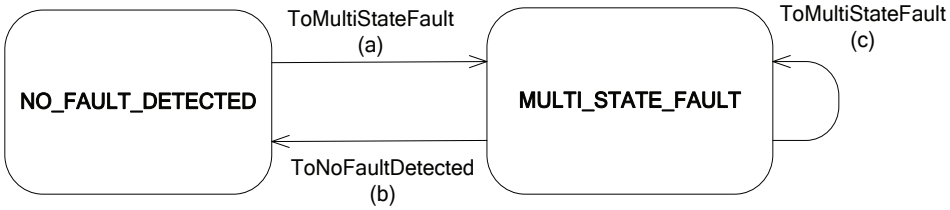

Figure 13-Y21. Transitions indicated by FAULT_STATE algorithm

13.4.6FAULT_STATUS_FLAGS Fault Algorithm

The FAULT_STATUS_FLAGS fault algorithm detects whether the monitored status flags are indicating a fault condition.

The parameters of this fault algorithm are:

pCurrentReliability	**This parameter, of type BACnetReliability, represents the current value of the Reliability property of the object that applies the fault algorithm.**
pMonitoredValue	**This parameter, of type BACnetStatusFlags, is the status flags value monitored by this algorithm.**

The conditions evaluated by this fault algorithm are:

(a) If pCurrentReliability is NO_FAULT_DETECTED, and the FAULT bit in pMonitoredValue is TRUE, then indicate a transition to the MEMBER_FAULT reliability.
(b) If pCurrentReliability is MEMBER_FAULT, and the FAULT bit in pMonitoredValue is FALSE, then indicate a transition to the NO_FAULT_DETECTED reliability.

Figure 13-Y21 depicts the reliability transitions that this fault algorithm may indicate:

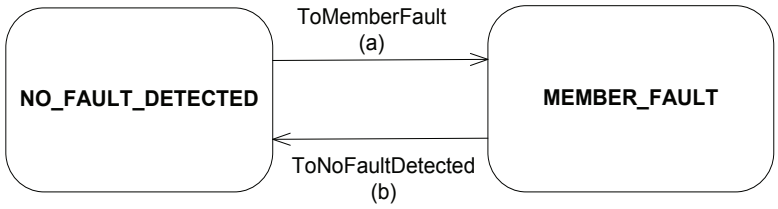

Figure 13-Y21. Transitions indicated by FAULT_STATUS_FLAGS algorithm

[Change Clause 13.8.1.8, p. 441 and Clause 13.9.1.8, p. 444]

13.[8/9].1.8 Event Type

This parameter, of type BACnetEventType, shall specify the type of event that has occurred. ~~Event types that are defined in this standard may be found in Table 12-15.~~

[Change Clause 13.8.1.14, p. 441 and Clause 13.9.1.14, p. 444]

13.[8/9].1.14 Event Values

This parameter, of type BACnetNotificationParameters, shall convey a set of values relevant to the particular event and whose content depends on the *event* type.~~of object that initiated the notification (see Tables 13-2, 13-3, and 13-4).~~ This parameter shall only be present if the 'Notify Type' parameter is EVENT or ALARM. ~~The returned set of values may be either a list of property values representing~~

~~properties of the object specified by the 'Event Object Identifier' or one of the standard event type notification parameter sets defined in Table 13-4.~~

[Change Clause 21, BACnetParameter production, p. 600]

```
BACnetEventParameter ::= CHOICE {

-- These choices have a one-to-one correspondence with the Event_Type enumeration with the exception
     of the
-- complex-event-type, which is used for proprietary event types.
     ...
     change-of-status-flags     [18] SEQUENCE {
                                          time-delay          [0] Unsigned,
                                          selected-flags      [1] BACnetStatusFlags
                                          }
                                          },
     -- context tag [19] is not used, see note below
     none                       [20] NULL
     }
-- CHOICE [6] has been intentionally omitted. It parallels the complex-event-type CHOICE [6] of the
-- BACnetNotificationParameters production which was introduced to allow the addition of proprietary
     event
-- algorithms whose event parameters are not necessarily network-visible.
-- CHOICE [19] has been intentionally omitted. It parallels the change-of-reliability event type
     CHOICE[19]
-- of the BACnetNotificationParameters production which was introduced for the notification of event
     state
-- changes to FAULT and from FAULT, which does not have event parameters.
```

[Change Clause 21, BACnetEventType production, p. 603]

```
BACnetEventType ::= ENUMERATED {
   ...
   change-of-status-flags      (18),
   change-of-reliability           (19),
   none                        (20),
   ...
   }
```

[Change Clause 21, BACnetNotificationParameters production, p. 608]

```
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
          }
          },
   change-of-reliability          [19] SEQUENCE {
          reliability          [0] BACnetReliability,
          status-flags         [1] BACnetStatusFlags,
          property-values      [2] SEQUENCE OF BACnetPropertyValue
          }
```

-- context tag [20] is not used, see note below

}

-- CHOICE [20] has been intentionally omitted. It parallels the 'none' event type CHOICE[20] of
-- the BACnetEventParameter production which was introduced for the case an object
-- does not apply an event algorithm

[Change Clause 21, BACnetReliability production, p. 626]

BACnetReliability ::= ENUMERATED {
...
member-fault (13),
monitored-object-fault (14),
...
}

[Change Clause K.2.2 BIBB Alarm and Event-Notification Internal-B (AE-N-I-B), p. 855]

K.2.2 BIBB – Alarm and Event-Notification Internal-B (AE-N-I-B)

Device B generates notifications about alarms and other events. *Devices claiming support for AE-N-I-B shall also claim support for AE-INFO-B.*

BACnet Service	Initiate	Execute
ConfirmedEventNotification	x	
UnconfirmedEventNotification	x	

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.