ADDENDA

ANSI/ASHRAE Addendum q to
ANSI/ASHRAE Standard 135.1-2013

Method of Test for Conformance to BACnet®

Approved by ASHRAE on and by the American National Standards Institute on December 7, 2018.

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. Instructions for how to submit a change can be found on the ASHRAE® website (https://www.ashrae.org/continuous-maintenance).

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

 ISSN 1041-2336

Approved American National Standard
ANSI
www.ansi.org

SPECIAL NOTE

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

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

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

The Senior 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 modifications to Standard 135.1 presented in this addendum are the result of change proposals made pursuant to the ASHRAE continuous maintenance procedures and of deliberations within Standing Standard Project Committee 135. The proposed changes are summarized below.

135.1-2013*q*-1. Update alarm and event tests for protocol revisions 13 and higher, p. 2

In the following document, language to be added to existing clauses of ANSI/ASHRAE 135.1-2013 and Addenda is indicated through the use of *italics*, while deletions are indicated by ~~strikethrough~~. Where entirely new subclauses are proposed to be added, plain type is used throughout. Only this new and deleted text is open to comment as this time. All other material in this document is provided for context only and is not open for public review comment except as it relates to the proposed changes.

The use of placeholders like X, Y, Z, X1, X2, N, NN, x, n, ?, etc., should not be interpreted as literal values of the final published version. These placeholders will be assigned actual numbers/letters only after final publication approval of the addendum.

135.1-2013*q*-1. Update alarm and event tests for protocol revisions 13 and higher

Rationale

Addendum 135-2010*af* at Protocol_Revision 13 introduced significant changes to event and fault reporting. Addendum 135.1-2013*q* adds new and corrects existing tests to meet the requirements of the following Clauses of Addendum 135-2010*af:*

- 135-2010*af*-16 - 135-2010*af*-21
- 135-2010*af*-23
- 135-2010*af*-27 - 135-2010*af*-32

Clauses not listed above do not result in changes to 135-2013.

In addition, numerous defects, which are independent of Addendum 135-2010*af*, are fixed.

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

Thomas Kurowski, Siemens
Michael Osborne, Reliable Controls
Cliff Copass, Johnson Controls
Dattatray Pawar, Siemens
Teresa Zotti, Philips Lighting
Courtney Adams, Siemens
Christoph Zeller, Sauter

[Change **Clause 7.3.1.10**, p. 43]

7.3.1.10 Event_Enable Tests

[Change **Clause 7.3.1.10.1**, p. 43]

[Reviewer Note: This test was modified to take into account the Feedback behavior that is required by the Output objects.]

7.3.1.10.1 Event_Enable Test for TO_OFFNORMAL, ~~and~~ TO_NORMAL, *and TO_FAULT*

~~Dependencies: ConfirmedEventNotification Service Initiation Tests, 8.4; UnconfirmedEventNotification Service Initiation Tests, 8.5; ReadProperty Service Execution Tests, 9.18; WriteProperty Service Execution Tests, 9.22.~~

Purpose: To verify that notification messages are transmitted only if the bit in Event_Enable corresponding to the event transition has a value of TRUE. ~~This test applies to all event generating object types that are capable of generating TO-OFFNORMAL and TO-NORMAL event transitions.~~

Test Concept: The IUT is configured *with an event-generating object, O1,* such that the Event_Enable property *is tested in all supported states*.~~indicates that some event transitions are to trigger an event notification and some are not.~~ Each event transition is triggered and the IUT is monitored to verify that notification messages are transmitted only for those transitions for which the Event_Enable property has a value of TRUE

Configuration Requirements: *If the Event_Enable property is configurable, repeat the test with Event_Enable = (T, F, F), (F, T, F), (F, F, T).* ~~The Event_Enable property shall be configured with a value of TRUE for either the TO-OFFNORMAL transition or the TO-NORMAL transition and the other event transition shall have a value of FALSE~~. If the Event_Enable property is not configurable, follow the test steps as written and verify correct behavior for the value of the Event_Enable property. *All other properties in O1, and any supporting objects, shall be configured to allow these events to be generated.*

~~For analog objects the Limit_Enable property shall be configured with the value (TRUE, TRUE). The referenced event-triggering property shall be set to a value that results in a NORMAL condition.~~*The event-generating object shall be in a NORMAL state at the start of the test.* ~~The value of the Transitions parameter for all recipients shall be (TRUE, TRUE, TRUE).~~ *D1 is either the pTimeDelay parameter or, in case of the EXTENDED and proprietary algorithms, a vendor specific delay (may be zero).*

D2 is either the pTimeDelayNormal parameter or, in case of the EXTENDED and proprietary algorithms, a vendor specific delay (may be zero).

~~In the test description below, "X" is used to designate the event-triggering property.~~

Test Steps:

1. VERIFY ~~Event_State~~*pCurrentState* = NORMAL
2. WAIT (~~Time_Delay~~*pTimeDelay* + **Notification Fail Time**)
3. *IF (O1 contains pFeedbackValue) THEN*
   MAKE (pFeedbackValue differ from pMonitoredValue)
   ELSE IF (~~X~~*pMonitoredValue* is writable) THEN
   WRITE ~~X~~*pMonitoredValue* = (a value that is OFFNORMAL)
   ELSE
   MAKE (~~X~~*pMonitoredValue* have a value that is OFFNORMAL)
4. WAIT ~~(Time_Delay~~*D1*
5. BEFORE **Notification Fail Time**
   IF (the Transitions bit corresponding to the TO-OFFNORMAL transition is TRUE) THEN *{*
   RECEIVE ConfirmedEventNotification-Request,
   'Process Identifier' = (any valid process ID),
   'Initiating Device Identifier' = IUT,
   'Event Object Identifier' = ~~(the event-generating object configured for this test)~~*O1*,
   'Time Stamp' = (~~the current local time~~*any valid time stamp*),
   'Notification Class' = (the class corresponding to the object being tested),
   'Priority' = (the value configured to correspond to a TO-OFFNORMAL transition),
   'Event Type' = (any valid event type),
   'Message Text' = (optional, any valid message text),
   'Notify Type' = EVENT | ALARM,
   'AckRequired' = TRUE | FALSE,
   'From State' = NORMAL,
   'To State' = OFFNORMAL,
   'Event Values' = (values appropriate to the event type)
   TRANSMIT BACnet-SimpleACK-PDU
   }
   ELSE
   CHECK (verify that the IUT did not transmit an event notification message)
6. VERIFY ~~Event_State~~*pCurrentState* = OFFNORMAL
7. *IF (O1 contains pFeedbackValue) THEN*
   MAKE (pFeedbackValue equal to pMonitoredValue)
   ELSE IF (~~X~~*pMonitoredValue* is writable) THEN
   WRITE ~~X~~*pMonitoredValue* = (a value that is NORMAL)
   ELSE
   MAKE (~~X~~*pMonitoredValue* have a value that is NORMAL)
8. WAIT (~~Time_Delay~~*D2*)
9. BEFORE **Notification Fail Time**
   IF (the Transitions bit corresponding to the TO-NORMAL transition is TRUE) THEN *{*
   RECEIVE ConfirmedEventNotification-Request,
   'Process Identifier' = (any valid process ID),
   'Initiating Device Identifier' = IUT,
   'Event Object Identifier' = (~~the event-generating object configured for this test~~*O1*),

'Time Stamp' = (~~the current local time~~*any valid time stamp*),
'Notification Class' = (the class corresponding to the object being tested),
'Priority' = (the value configured to correspond to a TO-NORMAL transition),
'Event Type' = (any valid event type),
'Message Text' = *(optional, any valid message text),*
'Notify Type' = EVENT | ALARM,
'AckRequired' = TRUE | FALSE,
'From State' = OFFNORMAL,
'To State' = NORMAL,
'Event Values' = (values appropriate to the event type)
TRANSMIT BACnet-SimpleACK-PDU
}
ELSE
CHECK (verify that the IUT did not transmit an event notification message)
10. VERIFY ~~Event_State~~*pCurrentState* = NORMAL
11. IF (~~the event-triggering object~~*O1* can be placed into a fault condition) THEN {
MAKE (~~the event-triggering object~~ *a condition exist that will cause O1 to generate a TO-FAULT transition*~~change to a fault condition~~)
BEFORE **Notification Fail Time**
IF (the Transitions bit corresponding to the TO-FAULT transition is TRUE) THEN *{*
RECEIVE ConfirmedEventNotification-Request,
'Process Identifier' = (any valid process ID),
'Initiating Device Identifier' = IUT,
'Event Object Identifier' = (~~the event-generating object configured for this test~~)*O1*,
'Time Stamp' = (~~the current local time~~*any valid time stamp*),
'Notification Class' = (the class corresponding to the object being tested),
'Priority' = (the value configured to correspond to a TO-FAULT transition),
'Event Type' = *(IF (Protocol_Revision < 13) THEN*
(any valid event type)
ELSE
CHANGE_OF_RELIABILITY),
'Message Text' = *(optional, any valid message text),*
'Notify Type' = EVENT | ALARM,
'AckRequired' = TRUE | FALSE,
'From State' = NORMAL,
'To State' = FAULT,
'Event Values' = (values appropriate to the event type)
TRANSMIT BACnet-SimpleACK-PDU
}
ELSE
CHECK (verify that the IUT did not transmit an event notification message)
VERIFY ~~Event_State~~*pCurrentState* = FAULT
}

Notes to Tester: The UnconfirmedEventNotification service may be substituted for the ConfirmedEventNotification service*, in which case the TD shall skip all of the steps in which a BACnet-SimpleACK-PDU is sent*. ~~The 'Message Text' parameter is omitted in the test description because it is optional. The IUT may include this parameter in the notification messages.~~

[Change **Clause 7.3.1.10.2**, p. 45]

[Reviewer Note: There is an error pointed out by BTL Clarification Request BTL-CR-0196, of not returning the TO_NORMAL bit of the Event_Enable to TRUE in step 7.]

7.3.1.10.2 Event_Enable Tests for TO_NORMAL only Algorithms

~~Dependencies: ConfirmedEventNotification Service Initiation Tests, 8.4; UnconfirmedEventNotification Service Initiation Tests, 8.5; ReadProperty Service Execution Tests, 9.18; WriteProperty Service Execution Tests, 9.22.~~

Purpose: To verify that notification messages are transmitted only if the bit in Event_Enable corresponding to the event transition has a value of TRUE. This test applies to objects that only support generation of TO_NORMAL transitions.

Test Concept: The IUT is configured such that the Event_Enable property indicates that some event transitions are to trigger an event notification ~~and some are not~~. Each event transition is triggered and the IUT is monitored to verify that notification messages are transmitted only for those transitions for which the Event_Enable property has a value of TRUE.

Configuration Requirements: In the Notification Class object providing recipient information, the value of the Transitions parameter for all recipients shall be (TRUE, TRUE, TRUE). *The event-generating object shall be in a NORMAL state at the start of the test. D1 is either the pTimeDelayNormal parameter or, in case of the EXTENDED and proprietary algorithms, a vendor specific delay (may be zero).*

Test Steps:

1. VERIFY ~~Event_State~~*pCurrentState* = NORMAL
2. MAKE (the TO_NORMAL bit of the Event_Enable property equal to TRUE)
3. MAKE (a condition exist that ~~would~~*will* cause the object to generate a TO_NORMAL transition)
4. WAIT D1
~~4~~*5*. BEFORE **Notification Fail Time**
RECEIVE ConfirmedEventNotification-Request,
'Process Identifier' = (any valid process ID),
'Initiating Device Identifier' = IUT,
'Event Object Identifier' = (the event-generating object configured for this test),
'Time Stamp' = (~~the current local time~~*any valid time stamp*),
'Notification Class' = (the class corresponding to the object being tested),
'Priority' = (the value configured to correspond to a TO_NORMAL transition),
'Event Type' = (any valid event type),
'Message Text' = *(optional, any valid message text),*
'Notify Type' = EVENT | ALARM,
'AckRequired' = TRUE | FALSE,
'From State' = NORMAL,
'To State' = NORMAL,
'Event Values' = (values appropriate to the event type)
~~5~~*6*. TRANSMIT BACnet-Simple~~Ack~~*ACK*-PDU
~~6~~*7*. VERIFY ~~Event_State~~*pCurrentState* = NORMAL
~~7~~*8*. IF (Event_Enable can be changed such that the TO_NORMAL transition is FALSE) *THEN {*
MAKE (the TO_NORMAL bit of the Event_Enable property equal to FALSE)
MAKE (a condition exist that would cause the object to generate a TO_NORMAL transition)
*WAIT (D1 + **Notification Fail Time**)*
CHECK (verify that the IUT did not transmit an event notification message)
MAKE (the TO_NORMAL bit of the Event_Enable property equal to TRUE)
}
~~8~~*9*. IF (the event-generating object can be placed into a fault condition) THEN *{*
IF (Event_Enable can be modified) THEN
MAKE (Event_Enable TO_FAULT transition equal TRUE)
IF (Event_Enable TO_FAULT transition = TRUE) THEN *{*
MAKE (the event-triggering object change to a fault condition)
BEFORE **Notification Fail Time**
RECEIVE ConfirmedEventNotification-Request,
'Process Identifier' = (any valid process ID),
'Initiating Device Identifier' = IUT,
'Event Object Identifier' = (the event-generating object configured for this test),

'Time Stamp' = (~~the current local time~~*any valid time stamp*),
'Notification Class' = (the class corresponding to the object being tested),
'Priority' = (the value configured to correspond to a TO_FAULT transition),
'Event Type' = *(IF (Protocol_Revision < 13) THEN*
(any valid event type)
ELSE
CHANGE_OF_RELIABILITY),
'Message Text' = *(optional, any valid message text),*
'Notify Type' = EVENT | ALARM,
'AckRequired' = TRUE | FALSE,
'From State' = NORMAL,
'To State' = FAULT,
'Event Values' = (values appropriate to the event type)
TRANSMIT ~~SimpleAck-PDU~~*BACnet-SimpleACK-PDU*
VERIFY ~~Event_State~~*pCurrentState* = FAULT
MAKE (the ~~event-triggering~~*event-generating* object change to a normal condition)
BEFORE **Notification Fail Time**
RECEIVE ConfirmedEventNotification-Request,
'Process Identifier' = (any valid process ID),
'Initiating Device Identifier' = IUT,
'Event Object Identifier' = (the event-generating object configured for this test),
'Time Stamp' = (~~the current local time~~*any valid time stamp*),
'Notification Class' = (the class corresponding to the object being tested),
'Priority' = (the value configured to correspond to a TO_NORMAL transition),
'Event Type' = *(IF (Protocol_Revision < 13) THEN*
(any valid event type)
ELSE
CHANGE_OF_RELIABILITY),
'Message Text' = *(optional, any valid message text),*
'Notify Type' = EVENT | ALARM,
'AckRequired' = TRUE | FALSE,
'From State' = FAULT,
'To State' = NORMAL,
'Event Values' = (values appropriate to the event type)
TRANSMIT ~~SimpleAck-PDU~~*BACnet-SimpleACK-PDU*
VERIFY pCurrentState = NORMAL
}
}
~~9~~*10*. IF (Event_Enable can be modified) THEN
MAKE (Event_Enable TO_FAULT transition equal FALSE)
~~10~~*11*. IF (Event_Enable TO_FAULT transition = FALSE) THEN *{*
MAKE (the event-triggering object change to a fault condition)
VERIFY ~~Event_State~~*pCurrentState* = FAULT
CHECK (verify that the IUT did not transmit an event notification message)
MAKE (the event-triggering object change to a normal condition)
}

Notes to Tester: ~~For objects that do not have a Time_Delay property, the Time_Delay value used in the test shall be 0.~~ The UnconfirmedEventNotification service may be substituted for the ConfirmedEventNotification service, in which case the TD shall skip all of the steps in which a BACnet-SimpleACK-PDU is sent. ~~The 'Message Text' parameter is omitted in the test description because it is optional. The IUT may include this parameter in the notification messages~~

[Change **Clause 7.3.1.11**, p. 47]

[Reviewer Note: Clause 13.5.2 excerpt states: "The Time Stamp conveyed in the acknowledgment notification shall not be derived from the Time Stamp of the original event notification, but rather the time at which the acknowledgment notification is generated." This is new language, not present in 135-2008, but which is presented as existing text in 135-2010*af*. Test 7.3.1.11 explicitly contradicts this and needs corrections.]

7.3.1.11 Acked_Transitions Tests

~~Dependencies: ConfirmedEventNotification Service Initiation Tests, 8.4; UnconfirmedEventNotification Service Initiation Tests, 8.5; AcknowledgeAlarm Service Execution Tests, 9.1; ReadProperty Service Execution Tests, 9.18; WriteProperty Service Execution Tests, 9.22.~~

~~BACnet Reference Clauses: 12.1.28, 12.2.24, 12.3.25, 12.4.21, 12.6.23, 12.7.27, 12.8.25, 12.12.11, 12.15.20, 12.16.20,12.17.35, 12.18.18, 12.19.19, 12.20.19, 12.23.27 and 12.25.23.~~

Purpose: To verify that the Acked_Transitions property tracks whether or not an acknowledgment has been received for a previously issued event notification. It also verifies the interrelationship between Status_Flags and Event_State. ~~This test applies to Event Enrollment objects and Accumulator, Analog Input, Analog Output, Analog Value, Binary Input, Binary Output, Binary Value, Life Safety Point, Life Safety Zone, Loop, Multi-state Input, Multi-state Output, Multi-state Value, Pulse COnverter and Trend Log objects that support intrinsic reporting.~~

Test Concept: The IUT is configured such that the Event_Enable property indicates that all event transitions are to trigger an event notification. The Acked_Transitions property shall have the value (TRUE, TRUE, TRUE) indicating that all previous transitions have been acknowledged. Each event transition is triggered and the Acked_Transitions property is monitored to verify that the appropriate bit is cleared when a notification message is transmitted and reset if an acknowledgment is received.

Configuration Requirements: The Event_Enable and Acked_Transitions properties shall be configured with a value of (TRUE, TRUE, TRUE). For analog objects the Limit_Enable property shall be configured with the value (TRUE, TRUE). The referenced event-triggering property shall be set to a value that results in a NORMAL condition. The value of the Transitions parameter for all recipients shall be (TRUE, TRUE, TRUE).

~~In the test description below, "X" is used to designate the event-triggering property.~~

Test Steps:

~~1. WAIT (Time_Delay + **Notification Fail Time**)~~
~~2~~*1*. VERIFY ~~Event_State~~*pCurrentState* = NORMAL
~~3~~*2*. VERIFY Acked_Transitions = (TRUE, TRUE, TRUE)
~~4~~*3*. *IF (Protocol_Revision is present AND Protocol_Revision e 13) THEN*
VERIFY ~~Status_Flags~~*pStatusFlags* = (FALSE, FALSE, ?, ?)
~~5~~*4*. IF (~~X~~*pMonitoredValue* is writable) THEN
WRITE ~~X~~*pMonitoredValue* = (a value that is OFFNORMAL)
ELSE
MAKE (~~X~~*pMonitoredValue* have a value that is OFFNORMAL)
~~6~~*5*. WAIT (~~Time_Delay~~*pTimeDelay*)
~~7~~*6*. BEFORE **Notification Fail Time**
RECEIVE ConfirmedEventNotification-Request,

'Process Identifier' =	(*PI1:* any valid process ID),
'Initiating Device Identifier' =	IUT,
'Event Object Identifier' =	(the event-generating object configured for this test),
'Time Stamp' =	(*Toffnormal:* any valid time stamp),
'Notification Class' =	(the class corresponding to the object being tested),
'Priority' =	(*Poffnormal:* the value configured to correspond to a TO-OFFNORMAL transition),
'Event Type' =	(any valid event type),
'Message Text' =	*(optional, any valid message text),*
'Notify Type' =	(the notify type configured for this event),
'AckRequired' =	TRUE,

'From State' = NORMAL,
'To State' = OFFNORMAL,
'Event Values' = (values appropriate to the event type)

~~8~~*7*. TRANSMIT BACnet-SimpleACK-PDU

~~9~~*8*. VERIFY ~~Event_State~~*pCurrentState* = OFFNORMAL

~~10~~*9*. VERIFY Acked_Transitions = (FALSE, TRUE, TRUE)

~~11~~*10*. *IF (Protocol_Revision is present AND Protocol_Revision e 13) THEN*
VERIFY ~~Status_Flags~~*pStatusFlags* = (TRUE, FALSE, ?, ?)

~~12~~*11*. IF (~~X~~*pMonitoredValue* is writable) THEN
WRITE ~~X~~*pMonitoredValue* = (a value that is NORMAL)
ELSE
MAKE (~~X~~*pMonitoredValue* have a value that is NORMAL)

~~13~~*12*. WAIT (~~Time_Delay~~*pTimeDelayNormal*)

~~14~~*13*. BEFORE **Notification Fail Time**
RECEIVE ConfirmedEventNotification-Request,
'Process Identifier' = (*PI2:* any valid process ID),
'Initiating Device Identifier' = IUT,
'Event Object Identifier' = (the event-generating object configured for this test),
'Time Stamp' = (*Tnormal:* any valid time stamp),
'Notification Class' = (the class corresponding to the object being tested),
'Priority' = (*Pnormal:* the value configured to correspond to a TO-NORMAL transition),
'Event Type' = (any valid event type),
'Message Text' = *(optional, any valid message text),*
'Notify Type' = (the notify type configured for this event),
'AckRequired' = TRUE,
'From State' = ~~OFNORMAL~~*OFFNORMAL*,
'To State' = NORMAL,
'Event Values' = (values appropriate to the event type)

~~15~~*14*. TRANSMIT BACnet-SimpleACK-PDU

~~16~~*15*. VERIFY ~~Event_State~~*pCurrentState* = NORMAL

~~17~~*16*. VERIFY Acked_Transitions = (FALSE, TRUE, FALSE)

~~18~~*17*. *IF (Protocol_Revision is present AND Protocol_Revision e 13) THEN*
VERIFY ~~Status_Flags~~*pStatusFlags* = (FALSE, FALSE, ?,?)

~~19~~*18*. IF (the event-triggering object can be placed into a fault condition) THEN *{*

~~20.~~ MAKE (~~the event-triggering object change to a fault condition)~~*a condition exist that will cause the object to generate a TO_FAULT transition)*

~~21.~~ BEFORE **Notification Fail Time**
RECEIVE ConfirmedEventNotification-Request,
'Process Identifier' = (*PI3:* any valid process ID),
'Initiating Device Identifier' = IUT,
'Event Object Identifier' = (the event-generating object configured for this test),
'Time Stamp' = (*Tfault:* any valid time stamp),
'Notification Class' = (the class corresponding to the object being tested),
'Priority' = (*Pfault:* the value configured to correspond to a TO-FAULT transition),
'Event Type' = *(IF (Protocol_Revision < 13) THEN*
(any valid event type)
ELSE
CHANGE_OF_RELIABILITY),
'Message Text' = *(optional, any valid message text),*
'Notify Type' = (the notify type configured for this event),
'AckRequired' = TRUE,
'From State' = NORMAL,
'To State' = FAULT,
'Event Values' = (values appropriate to the event type)

~~22~~*19*. TRANSMIT BACnet-SimpleACK-PDU

~~23~~*20*. VERIFY ~~Event_State~~*pCurrentState* = FAULT

~~24~~*21*. VERIFY Acked_Transitions = (FALSE, FALSE, FALSE)
~~25. VERIFY Status_Flags = (TRUE, TRUE, ?, ?)~~
~~26. MAKE (the event triggering object change to a normal condition)~~
~~27. BEFORE **Notification Fail Time**~~
~~RECEIVE ConfirmedEventNotification-Request,~~
~~'Process Identifier' = (any valid process ID),~~
~~'Initiating Device Identifier' = IUT,~~
~~'Event Object Identifier' = (the event-generating object configured for this test),~~
~~'Time Stamp' = (any valid time stamp)~~
~~'Notification Class' = (the class corresponding to the object being tested),~~
~~'Priority' = (the value configured to correspond to a TO-NORMAL transition),~~
~~'Event Type' = (any valid event type),~~
~~'Notify Type' = (the notify type configured for this event),~~
~~'AckRequired' = TRUE,~~
~~'From State' = FAULT,~~
~~'To State' = NORMAL,~~
~~'Event Values' = (values appropriate to the event type)~~
~~28. TRANSMIT BACnet-SimpleACK-PDU~~
~~29. VERIFY Event_State = NORMAL~~
~~30. VERIFY Acked_Transitions = (FALSE, FALSE, FALSE)~~
~~31. VERIFY Status_Flags = (FALSE, FALSE, ?, ?)~~
~~32~~ *22*. TRANSMIT AcknowledgeAlarm-Request,
'Acknowledging Process Identifier' = (~~the value of the 'Process Identifier' in step 21~~*PI3*,
'Event Object Identifier' = (*the event-generating object configured for this test*~~the 'Event Object Identifier' in step 21~~),
'Event State Acknowledged' = FAULT,
'Time Stamp' = (*Tfault*~~'Time Stamp' in step 21~~),
'*Acknowledgement Source*' = *(a character string),*
'Time of Acknowledgment' = (the TD's current time~~)~~
~~33~~ *23*. RECEIVE BACnet-SimpleACK-PDU
~~34~~ *24*. IF (Protocol_Revision is present and Protocol_Revision ≥ 1) THEN
BEFORE **Notification Fail Time**
RECEIVE ConfirmedEventNotification-Request,
'Process Identifier' = (~~the value of the 'Process Identifier' in step 21~~*PI3*),
'Initiating Device Identifier' = IUT,
'Event Object Identifier' = (~~the 'Event Object Identifier' in step 21~~*the event-generating object configured for this test*),
'Time Stamp' = (~~the 'Time Stamp' in step 21~~ *Tfault*),
'Notification Class' = (~~the 'Notification Class' in step 21~~*the class corresponding to the object being tested*),
'Priority' = (~~the 'Priority' in step 21~~*Pfault*),
'Event Type' = ~~(the 'Event Type' in step 21),~~
IF (Protocol_Revision < 13)
(any valid event type)
ELSE
CHANGE_OF_RELIABILITY,
'Message Text' = *(optional, any valid message text),*
'Notify Type' = ACK_NOTIFICATION,
'To State' = FAULT
ELSE
BEFORE **Notification Fail Time**
. RECEIVE ConfirmedEventNotification-Request,
'Process Identifier' = (~~the value of the 'Process Identifier' in step 21~~*PI3*),
'Initiating Device Identifier' = IUT,
'Event Object Identifier' = (~~the 'Event Object Identifier' in step 21~~*the event-generating object configured for this test*),

'Time Stamp' = (~~the current time or sequence number~~*Tfault*),
'Notification Class' = (~~the 'Notification Class' in step 21~~*the class corresponding to the object being tested*),
'Priority' = (~~the 'Priority' in step 21~~*Pfault*),
'Event Type' = (~~the 'Event Type' in step 21~~*any valid event type*),
'Message Text' = *(optional, any valid message text),*
'Notify Type' = ACK_NOTIFICATION

~~35~~ *25.* TRANSMIT BACnet-SimpleACK-PDU
~~36~~ *26.* VERIFY Acked_Transitions = (FALSE, TRUE, FALSE)
}
~~37~~ *27.* TRANSMIT AcknowledgeAlarm-Request,
'Acknowledging Process Identifier' = (*PI2*~~the value of the 'Process Identifier' in step 27~~),
'Event Object Identifier' = (~~the 'Event Object Identifier' in step 27~~*the event-generating object configured for this test*),
'Event State Acknowledged' = NORMAL,
'Time Stamp' = (~~the 'Time Stamp in step 27~~*Tnormal)*,
'*Acknowledgement Source*' = *(a character string),*
'Time of Acknowledgment' = (the TD's current time)
~~38~~ *28.* RECEIVE BACnet-SimpleACK-PDU
~~39~~ *29.* IF (Protocol_Revision is present and Protocol_Revision ≥ 1) THEN
BEFORE **Notification Fail Time**
RECEIVE ConfirmedEventNotification-Request,
'Process Identifier' = (~~the value of the 'Process Identifier' in step 27~~*PI2*),
'Initiating Device Identifier' = IUT,
'Event Object Identifier' = (~~the 'Event Object Identifier' in step 27~~*the event-generating object configured for this test*),
'Time Stamp' = (~~the current time or sequence number~~*Tnormal*),
'Notification Class' = (~~the 'Notification Class' in step 27~~*the class corresponding to the object being tested*),
'Priority' = (~~the 'Priority' in step 27~~*Pnormal*),
'Event Type' = (~~the 'Event Type' in step 27~~*any valid event type*),
'Message Text' = *(optional, any valid message text),*
'Notify Type' = ACK_NOTIFICATION,
'To State' = NORMAL
ELSE
BEFORE **Notification Fail Time**
RECEIVE ConfirmedEventNotification-Request,
'Process Identifier' = (*PI2*~~the value of the 'Process Identifier' in step 27~~),
'Initiating Device Identifier' = IUT,
'Event Object Identifier' = (*the event-generating object configured for this test*~~the 'Event Object Identifier' in step 27~~),
'Time Stamp' = (~~the current time or sequence number~~*Tnormal*),
'Notification Class' = (*the class corresponding to the object being tested*~~the 'Notification Class' in step 27~~),
'Priority' = (*Pnormal*~~the 'Priority' in step 27~~),
'Event Type' = (*any valid event type*~~the 'Event Type' in step 27~~),
'Message Text' = *(optional, any valid message text),*
'Notify Type' = ACK_NOTIFICATION
~~40~~ *30.* TRANSMIT BACnet-SimpleACK-PDU
~~41~~ *31.* VERIFY Acked_Transitions = (FALSE, TRUE, TRUE)
~~42~~ *32.* TRANSMIT AcknowledgeAlarm-Request,
'Acknowledging Process Identifier' = (*PI1*~~the value of the 'Process Identifier' in step 7~~),
'Event Object Identifier' = (*the event-generating object configured for this test*~~the 'Event Object Identifier' in step 7~~),
'Event State Acknowledged' = OFFNORMAL,
'Time Stamp' = (~~the 'Time Stamp in step 7~~*Toffnormal*),

'*Acknowledgement Source*' = *(a character string),*
'Time of Acknowledgment' = (the TD's current time)

~~43~~ *33*. RECEIVE BACnet-SimpleACK-PDU

~~44~~ *34*. IF (Protocol_Revision is present and Protocol_Revision ≥ 1) THEN
BEFORE **Notification Fail Time**
RECEIVE ConfirmedEventNotification-Request,
'Process Identifier' = (*PI1*~~the value of the 'Process Identifier' in step 7~~),
'Initiating Device Identifier' = IUT,
'Event Object Identifier' = (~~the 'Event Object Identifier' in step 7~~*the event-generating object configured for this test*),
'Time Stamp' = (~~the current time or sequence number~~*Toffnormal*),
'Notification Class' = (~~the 'Notification Class' in step 7~~*the class corresponding to the object being tested*),
'Priority' = (~~the 'Priority' in step 7~~*Poffnormal*),
'Event Type' = (~~the 'Event Type' in step 7~~*any valid event type*),
'Message Text' = *(optional, any valid message text),*
'Notify Type' = ACK_NOTIFICATION,
'To State' = OFFNORMAL
ELSE
BEFORE **Notification Fail Time**
RECEIVE ConfirmedEventNotification-Request,
'Process Identifier' = (~~the value of the 'Process Identifier' in step 7~~*PI1*),
'Initiating Device Identifier' = IUT,
'Event Object Identifier' = (the *the event-generating object configured for this test*~~current time or sequence number~~),
'Time Stamp' = (~~the current time or sequence number~~*Toffnormal*),
'Notification Class' = (~~the 'Notification Class' in step 7~~*the class corresponding to the object being tested*),
'Priority' = (~~the 'Priority' in step 7~~*Poffnormal*),
'Event Type' = (~~the 'Event Type' in step 7~~*any valid event type*),
'Message Text' = *(optional, any valid message text),*
'Notify Type' = ACK_NOTIFICATION

~~45~~ *35*. TRANSMIT BACnet-SimpleACK-PDU

~~46~~ *36*. VERIFY Acked_Transitions = (TRUE, TRUE, TRUE)

Notes to Tester: The UnconfirmedEventNotification service may be substituted for the ConfirmedEventNotification service*, in which case the TD shall skip all of the steps in which a BACnet-SimpleACK-PDU is sent*.~~The 'Message Text' parameter is omitted in the test description because it is optional. The IUT may include this parameter in the notification messages~~

[Change **Clause 7.3.1.12**, p. 51]

7.3.1.12 Notify_Type Test

~~Dependencies: ConfirmedEventNotification Service Initiation Tests, 8.4; UnconfirmedEventNotification Service Initiation Tests, 8.5; WriteProperty Service Execution Tests, 9.22.~~

~~BACnet Reference Clauses: 12.1.29, 12.2.25, 12.3.26, 12.4.22, 12.6.24, 12.7.28, 12.8.26, 12.12.6, 12.15.21, 12.16.21, 12.17.36, 12.18.19, 12.19.20, 12.20.20, 12.23.28 and 12.25.24.~~

Purpose: To verify that the value of the Notify_Type property determines whether an event notification is transmitted as an alarm or as an event.~~ This test applies to Event Enrollment objects and objects that support intrinsic reporting.~~

Configuration Requirements: The IUT shall be configured with two event-generation objects, ~~E1~~ *E1* and *E2* ~~E2~~. Object E1 shall be configured with a Notify_Type of ALARM and E2 shall be configured with a Notify_Type of EVENT. The value of the Transitions parameter for all recipients shall be (TRUE, TRUE, TRUE)

In the test description below *X1* ~~X1~~ and *X2* ~~X2~~ are used to designate the ~~event-triggering property~~*pMonitoredValue algorithm parameter* linked to ~~E1~~ *E1* and ~~E2~~ *E2*, respectively. *X1 and X2 shall be set to a value that results in a NORMAL condition of E1 and E2, respectively.*

Test Steps:

1. VERIFY ~~(E1~~*E1*~~)~~, ~~Event_State~~*pCurrentState* = NORMAL
2. VERIFY ~~(E2~~ *E2*~~)~~, ~~Event_State~~*pCurrentState* = NORMAL
3. WAIT (~~Time_Delay~~*pTimeDelay* + **Notification Fail Time**)
4. IF (X_1 is writable) THEN
   WRITE ~~X1~~ *X1* = (a value that will cause a transition in ~~E1~~ *E1*)
   ELSE
   MAKE (~~X1~~ *X1* a value that will cause a transition in ~~E1~~ *E1*)
5. IF (the transition is not a FAULT transition*) THEN*
   IF (the transition is a TO-OFFNORMAL transition) THEN
   WAIT (~~Time_Delay~~*pTimeDelay*)
   ELSE
   WAIT (pTimeDelayNormal)
6. BEFORE **Notification Fail Time**
   RECEIVE ConfirmedEventNotification-Request,

'Process Identifier' =	(any valid process ID),
'Initiating Device Identifier' =	IUT,
'Event Object Identifier' =	~~(E1~~ *E1*~~)~~,
'Time Stamp' =	(~~the current local time~~*any valid time stamp*),
'Notification Class' =	(the class corresponding to the object being tested),
'Priority' =	(any valid value),
'Event Type' =	(any valid event type),
'Message Text' =	*(optional, any valid message text),*
'Notify Type' =	ALARM,
'AckRequired' =	TRUE | FALSE,
'From State' =	NORMAL,
'To State' =	(any valid value),
'Event Values' =	(values appropriate to the event type)

7. TRANSMIT ~~SimpleAck-PDU~~*BACnet-SimpleACK-PDU*
8. IF (X_2 is writable) THEN
   WRITE ~~X2~~ *X2* = (a value that will cause a transition in ~~E2~~ *E2*)
   ELSE
   MAKE (~~X2~~ *X2* a value that will cause a transition in ~~E2~~ *E2*)
9. IF (the transition is not a FAULT transition*) THEN*
   IF (the transition is a TO-OFFNORMAL transition) THEN
   WAIT (~~Time_Delay~~*pTimeDelay*)
   ELSE
   WAIT (pTimeDelayNormal)
10. BEFORE **Notification Fail Time**
    RECEIVE ConfirmedEventNotification-Request,

'Process Identifier' =	(any valid process ID),
'Initiating Device Identifier' =	IUT,
'Event Object Identifier' =	~~(E2~~ *E2*~~)~~,
'Time Stamp' =	(~~the current local time~~*any valid time stamp*),
'Notification Class' =	(the class corresponding to the object being tested),
'Priority' =	(any valid value),
'Event Type' =	(any valid event type),
'Message Text' =	*(optional, any valid message text),*
'Notify Type' =	EVENT,
'AckRequired' =	TRUE | FALSE,

'From State' = NORMAL,
'To State' = (any valid value),
'Event Values' = (values appropriate to the event type)

11. TRANSMIT ~~SimpleAck-PDU~~*BACnet-SimpleACK-PDU*

Notes to Tester: If Notify_Type is writable this test may be performed with one event generating object by changing Notify_Type from ALARM to EVENT in order to cover both cases. ~~The 'Message Text' parameter is omitted in the test description because it is optional. The IUT may include this parameter in the notification messages~~

[Remove **Clause 7.3.1.13**, p. 52]

[Reviewer Note: Test 7.3.1.13 is extremely long (58 steps) and tests more than the Limit_Enable property's ability to enable or disable the reporting of out-of-range events. The proposed change breaks the existing test into two tests; one for LowLimitEnable (7.3.1.13.1), the other for HighLimitEnable (7.3.1.13.2). In addition, new Integer objects have been added to the 135 standard that may have a writable Limit_Enable property and these objects are not covered by the exsting 7.3.1.13 test.]

~~**7.3.1.13 Limit_Enable Test**~~

~~Dependencies: ConfirmedEventNotification Service Initiation Tests, 8.4; UnconfirmedEventNotification Service Initiation Tests, 8.5; ReadProperty Service Execution Tests, 9.18; WriteProperty Service Execution Tests, 9.22.~~

~~Purpose: To verify that the Limit_Enable property correctly enables or disables reporting of out-of range events. This test applies to objects with a Limit_Enable property.~~

~~Test Concept: The event triggering property is manipulated to cause both the high limit and the low limit to be exceeded for each possible combination of values for Limit_Enable. The resulting event notification messages are monitored to verify that they are transmitted only for circumstances where the associated event limit is enabled.~~

~~Configuration Requirements: Configure the object with High_Limit, Low_Limit, and Deadband values such that High_Limit – Deadband > Low_Limit and both the Low_Limit and High_Limit values are within the valid range of values for Present_Value. If the device cannot be configured with limit values that meet these conditions, then this test shall be skipped. The Event_Enable property shall be set to (TRUE, ?, TRUE) for this test. If the Event_Enable cannot be configured such that the TO-NORMAL and the TO-OFFNORMAL transitions are TRUE, then this test shall be skipped.~~

~~In the test description below, "X" is used to designate the event-triggering property.~~

~~Test Steps:~~

~~1. IF Limit_Enable can be made to be equal (TRUE, TRUE) THEN~~
~~2. IF Limit_Enable is writable THEN~~
~~WRITE Limit_Enable = (TRUE, TRUE)~~
~~ELSE~~
~~MAKE (Limit_Enable = (TRUE, TRUE))~~
~~3. WAIT (Time_Delay + **Notification Fail Time**)~~
~~4. VERIFY Event_State = NORMAL~~
~~5. IF (X is writable) THEN~~
~~WRITE X = (a value that exceeds High_Limit)~~
~~ELSE~~
~~MAKE (X a value that exceeds High_Limit)~~
~~6. WAIT (Time_Delay)~~
~~7. BEFORE **Notification Fail Time**~~
~~RECEIVE ConfirmedEventNotification Request,~~
~~'Process Identifier' = (any valid process ID),~~
~~'Initiating Device Identifier' = IUT,~~
~~'Event Object Identifier' = (the object configured for this test),~~

~~'Time Stamp' = (the current local time),~~
~~'Notification Class' = (the class corresponding to the object being tested),~~
~~'Priority' = (the value configured to correspond to a TO-OFFNORMAL transition),~~
~~'Event Type' = OUT_OF_RANGE,~~
~~'Notify Type' = ALARM | EVENT,~~
~~'AckRequired' = TRUE | FALSE,~~
~~'From State' = NORMAL,~~
~~'To State' = HIGH_LIMIT,~~
~~'Event Values' = (values appropriate to the event type)~~

~~8. TRANSMIT SimpleAck-PDU~~
~~9. IF (X is writable) THEN~~
~~WRITE X = (a value that is lower than Low_Limit)~~
~~ELSE~~
~~MAKE (X a value that is lower than Low_Limit)~~
~~10. WAIT (Time_Delay)~~
~~11. BEFORE **Notification Fail Time**~~
~~RECEIVE ConfirmedEventNotification-Request,~~
~~'Process Identifier' = (any valid process ID),~~
~~'Initiating Device Identifier' = IUT,~~
~~'Event Object Identifier' = (the object configured for this test),~~
~~'Time Stamp' = (the current local time),~~
~~'Notification Class' = (the class corresponding to the object being tested),~~
~~'Priority' = (the value configured to correspond to a TO-NORMAL transition),~~
~~'Event Type' = OUT_OF_RANGE,~~
~~'Notify Type' = ALARM | EVENT,~~
~~'AckRequired' = TRUE | FALSE,~~
~~'From State' = HIGH_LIMIT,~~
~~'To State' = NORMAL,~~
~~'Event Values' = (values appropriate to the event type)~~
~~12. TRANSMIT SimpleAck-PDU~~
~~13. WAIT (Time_Delay)~~
~~14. BEFORE **Notification Fail Time**~~
~~RECEIVE ConfirmedEventNotification-Request,~~
~~'Process Identifier' = (any valid process ID),~~
~~'Initiating Device Identifier' = IUT,~~
~~'Event Object Identifier' = (the object configured for this test),~~
~~'Time Stamp' = (the current local time),~~
~~'Notification Class' = (the class corresponding to the object being tested),~~
~~'Priority' = (the value configured to correspond to a TO-OFFNORMAL transition),~~
~~'Event Type' = OUT_OF_RANGE,~~
~~'Notify Type' = ALARM | EVENT,~~
~~'AckRequired' = TRUE | FALSE,~~
~~'From State' = NORMAL,~~
~~'To State' = LOW_LIMIT,~~
~~'Event Values' = (values appropriate to the event type)~~
~~15. TRANSMIT SimpleAck-PDU~~
~~16. IF (X is writable) THEN~~
~~WRITE X = (a value that is between Low_Limit + deadband and High_Limit)~~
~~ELSE~~
~~MAKE (X a value that is between than Low_Limit + deadband and High_Limit)~~
~~17. WAIT (Time_Delay)~~
~~18. BEFORE **Notification Fail Time**~~
~~RECEIVE ConfirmdEventNotification-Request,~~
~~'Process Identifier' = (any valid process ID),~~
~~'Initiating Device Identifier' = IUT,~~
~~'Event Object Identifier' = (the object configured for this test),~~

~~'Time Stamp' = (the current local time),~~
~~'Notification Class' = (the class corresponding to the object being tested),~~
~~'Priority' = (the value configured to correspond to a TO-NORMAL transition),~~
~~'Event Type' = OUT_OF_RANGE,~~
~~'Notify Type' = ALARM | EVENT,~~
~~'AckRequired' = TRUE | FALSE,~~
~~'From State' = LOW_LIMIT,~~
~~'To State' = NORMAL,~~
~~'Event Values' = (values appropriate to the event type)~~
~~19. TRANSMIT SimpleAck-PDU~~
~~20. IF Limit_Enable can be made to equal (FALSE, TRUE) THEN~~
~~21. IF Limit_Enable is writable THEN~~
~~WRITE Limit_Enable = (FALSE, TRUE)~~
~~ELSE~~
~~MAKE (Limit_Enable = (FALSE,TRUE))~~
~~22. IF (X is writable) THEN~~
~~WRITE X = (a value that exceeds High_Limit)~~
~~ELSE~~
~~MAKE (X a value that exceeds High_Limit)~~
~~23. WAIT (Time_Delay)~~
~~24. BEFORE **Notification Fail Time**~~
~~RECEIVE ConfirmedEventNotification-Request,~~
~~'Process Identifier' = (any valid process ID),~~
~~'Initiating Device Identifier' = IUT,~~
~~'Event Object Identifier' = (the object configured for this test),~~
~~'Time Stamp' = (the current local time),~~
~~'Notification Class' = (the class corresponding to the object being tested),~~
~~'Priority' = (the value configured to correspond to a TO-OFFNORMAL transition),~~
~~'Event Type' = OUT_OF_RANGE,~~
~~'Notify Type' = ALARM | EVENT,~~
~~'AckRequired' = TRUE | FALSE,~~
~~'From State' = NORMAL,~~
~~'To State' = HIGH_LIMIT,~~
~~'Event Values' = (values appropriate to the event type)~~
~~25. TRANSMIT SimpleAck-PDU~~
~~26. IF (X is writable) THEN~~
~~WRITE X = (a value that is between Low_Limit and High_Limit-Deadband)~~
~~ELSE~~
~~MAKE (X a value that is between Low_Limit and High_Limit-Deadband)~~
~~27. WAIT (Time_Delay)~~
~~28. BEFORE **Notification Fail Time**~~
~~RECEIVE ConfirmedEventNotification-Request,~~
~~'Process Identifier' = (any valid process ID),~~
~~'Initiating Device Identifier' = IUT,~~
~~'Event Object Identifier' = (the object configured for this test),~~
~~'Time Stamp' = (the current local time),~~
~~'Notification Class' = (the class corresponding to the object being tested),~~
~~'Priority' = (the value configured to correspond to a TO-OFFNORMAL transition),~~
~~'Event Type' = OUT_OF_RANGE,~~
~~'Notify Type' = ALARM | EVENT,~~
~~'AckRequired' = TRUE | FALSE,~~
~~'From State' = HIGH_LIMIT,~~
~~'To State' = NORMAL,~~
~~'Event Values' = (values appropriate to the event type)~~
~~29. TRANSMIT SimpleAck-PDU~~
~~30. IF (X is writable) THEN~~

~~WRITE X = (a value that is lower than Low_Limit)~~
~~ELSE~~
~~MAKE (X a value that is lower than Low_Limit)~~
~~31. WAIT (Time_Delay + **Notification Fail Time**)~~
~~32. CHECK (verify that no notification message was transmitted)~~
~~33. IF (X is writable) THEN~~
~~WRITE X = (a value that is between than Low_Limit+Deadband and High_Limit)~~
~~ELSE~~
~~MAKE (X a value that is lower than Low_Limit+Deadband and High_Limit)~~
~~34. WAIT (Time_Delay + **Notification Fail Time**)~~
~~35. IF Limit_Enable can be made to equal (TRUE, FALSE) THEN~~
~~36. IF Limit_Enable is writable THEN~~
~~WRITE Limit_Enable = (TRUE, FALSE)~~
~~ELSE~~
~~MAKE (Limit_Enable = (TRUE, FALSE))~~
~~37. IF (X is writable) THEN~~
~~WRITE X = (a value that exceeds High_Limit)~~
~~ELSE~~
~~MAKE (X a value that exceeds High_Limit)~~
~~38. WAIT (Time_Delay + **Notification Fail Time**)~~
~~39. CHECK (Verify that no notification message was transmitted)~~
~~40. IF (X is writable) THEN~~
~~WRITE X = (a value that is lower than Low_Limit)~~
~~ELSE~~
~~MAKE (X a value that is lower than Low_Limit)~~
~~41. WAIT (Time_Delay)~~
~~42. BEFORE **Notification Fail Time**~~
~~RECEIVE ConfirmedEventNotification-Request,~~
~~'Process Identifier' = (any valid process ID),~~
~~'Initiating Device Identifier' = IUT,~~
~~'Event Object Identifier' = (the object configured for this test),~~
~~'Time Stamp' = (the current local time),~~
~~'Notification Class' = (the class corresponding to the object being tested),~~
~~'Priority' = (the value configured to correspond to a TO-OFFNORMAL transition),~~
~~'Event Type' = OUT_OF_RANGE,~~
~~'Notify Type' = ALARM | EVENT,~~
~~'AckRequired' = TRUE | FALSE,~~
~~'From State' = NORMAL,~~
~~'To State' = LOW_LIMIT,~~
~~'Event Values' = (values appropriate to the event type)~~
~~43. TRANSMIT SimpleAck-PDU~~
~~44. IF (X is writable) THEN~~
~~WRITE X = (a value that is between Low_Limit + Deadband and High_Limit)~~
~~ELSE~~
~~MAKE (X a value that is between Low_Limit + Deadband and High_Limit)~~
~~45. WAIT (Time_Delay)~~
~~46. BEFORE **Notification Fail Time**~~
~~RECEIVE ConfirmedEventNotification-Request,~~
~~'Process Identifier' = (any valid process ID),~~
~~'Initiating Device Identifier' = IUT,~~
~~'Event Object Identifier' = (the object configured for this test),~~
~~'Time Stamp' = (the current local time),~~
~~'Notification Class' = (the class corresponding to the object being tested),~~
~~'Priority' = (the value configured to correspond to a TO-NORMAL transition),~~
~~'Event Type' = OUT_OF_RANGE,~~
~~'Notify Type' = ALARM | EVENT,~~

~~'AckRequired' = TRUE | FALSE,~~
~~'From State' = LOW_LIMIT,~~
~~'To State' = NORMAL,~~
~~'Event Values' = (values appropriate to the event type)~~
~~47. IF Limit_Enable can be made to equal (FALSE, FALSE) THEN~~
~~48. IF Limit_Enable is writable THEN~~
~~WRITE Limit_Enable = (FALSE, FALSE)~~
~~ELSE~~
~~MAKE (Limit_Enable = (FALSE, FALSE))~~
~~49. IF (X is writable) THEN~~
~~WRITE X = (a value that exceeds High_Limit)~~
~~ELSE~~
~~MAKE (X a value that exceeds High_Limit)~~
~~50. WAIT (Time_Delay + **Notification Fail Time**)~~
~~51. CHECK (verify that no notification message was transmitted)~~
~~52. IF (X is writable) THEN~~
~~WRITE X = (a value that is lower than Low_Limit)~~
~~ELSE~~
~~MAKE (X a value that is lower than Low_Limit)~~
~~53. WAIT (Time_Delay + **Notification Fail Time**)~~
~~54. CHECK (verify that no notification message was transmitted)~~
~~55. IF (X is writable) THEN~~
~~WRITE X = (a value that is between Low_Limit and High_Limit)~~
~~ELSE~~
~~MAKE (X a value that is between Low_Limit and High_Limit)~~
~~56. WAIT (Time_Delay + **Notification Fail Time**)~~
~~57. CHECK (verify that no notification message was transmitted)~~

~~Notes to Tester: The UnconfirmedEventNotification service may be substituted for the ConfirmedEventNotification service in which case the TD shall skip all of the steps in which a BACnet SimpleACK-PDU is sent. The 'Message Text' parameter is omitted in the test description because it is optional. The IUT may include this parameter in the notification messages.~~

[Add new **Clause 7.3.1.13**, p. 52]

7.3.1.13 Limit_Enable Tests

7.3.1.13.1 Limit_Enable Test, LowLimitEnable

Purpose: To verify that the LowLimitEnable flag in the Limit_Enable property correctly enables or disables reporting of out of range events. This test applies to objects with a Limit_Enable property.

Test Concept: The LowLimitEnable flag is set to true in the Limit_Enable property, and the event-triggering property is manipulated to cause the low limit to be exceeded. This should generate an event notification and make Event_State = Low_Limit. After the event-triggering property is returned to a normal value, the LowLimitEnable flag is set to false, and the event-triggering property is again manipulated to exceed the low limit. No event notification should be observed, and the Event_State must have a value of normal.

Configuration Requirements: Configure the object with pHighLimit, pLowLimit, and pDeadband values such that pLowLimit + pDeadband < pHighLimit and both the pLowLimit and pHighLimit values are within the valid range of values for the event-triggering property. If the device cannot be configured with limit values that meet these conditions, then this test shall be skipped. The Event_Enable property shall be set to (TRUE, ?, TRUE) for this test. If the Event_Enable property cannot be configured such that the TO-NORMAL and the TO-OFFNORMAL transitions are TRUE, this test shall be skipped.

Test Steps:

1. MAKE pLimitEnable = (TRUE, ?)
2. VERIFY pCurrentState = NORMAL
3. MAKE (pMonitoredValue a value less than pLowLimit)
4. WAIT (pTimeDelay)
5. BEFORE **Notification Fail Time**
   RECEIVE ConfirmedEventNotification-Request,
   'Process Identifier' = (any valid process ID),
   'Initiating Device Identifier' = IUT,
   'Event Object Identifier' = (the object configured for this test),
   'Time Stamp' = (any valid time stamp),
   'Notification Class' = (the class corresponding to the object being tested),
   'Priority' = (the value configured to correspond to a TO-OFFNORMAL transition),
   'Event Type' = OUT_OF_RANGE,
   'Message Text' = (optional, any valid message text),
   'Notify Type' = ALARM | EVENT,
   'AckRequired' = TRUE | FALSE,
   'From State' = NORMAL,
   'To State' = LOW_LIMIT,
   'Event Values' = (values appropriate to the event type)
6. TRANSMIT BACnet-SimpleACK-PDU
7. VERIFY pCurrentState = LOW_LIMIT
8. MAKE (pMonitoredValue a value that is between pLowLimit + pDeadband and pHighLimit)
9. WAIT (pTimeDelayNormal)
10. BEFORE **Notification Fail Time**
    RECEIVE ConfirmedEventNotification-Request,
    'Process Identifier' = (any valid process ID),
    'Initiating Device Identifier' = IUT,
    'Event Object Identifier' = (the object configured for this test),
    'Time Stamp' = (any valid time stamp),
    'Notification Class' = (the class corresponding to the object being tested),
    'Priority' = (the value configured to correspond to a TO-NORMAL transition),
    'Event Type' = OUT_OF_RANGE,
    'Message Text' = (optional, any valid message text),
    'Notify Type' = ALARM | EVENT,
    'AckRequired' = TRUE | FALSE,
    'From State' = LOW_LIMIT,
    'To State' = NORMAL,
    'Event Values' = (values appropriate to the event type)
11. TRANSMIT BACnet-SimpleACK-PDU
12. MAKE pLimitEnable = (FALSE, ?)
13. VERIFY pCurrentState = NORMAL
14. MAKE (pMonitoredValue a value less than pLowLimit)
15. WAIT (pTimeDelay + **Notification Fail Time**)
16. CHECK (verify that no notification message was transmitted)
17. VERIFY pCurrentState = NORMAL

Notes to Tester: The UnconfirmedEventNotification service may be substituted for the ConfirmedEventNotification service in which case the TD shall skip all of the steps in which a SimpleACK-PDU is sent.

7.3.1.13.2 Limit_Enable Test, HighLimitEnable

Purpose: To verify that the HighLimitEnable flag in the Limit_Enable property correctly enables or disables reporting of out of range events. This test applies to objects with a Limit_Enable property.

Test Concept: The HighLimitEnable flag is set to true in the Limit_Enable property and the event-triggering property is manipulated to cause the high limit to be exceeded. This should generate an event notification and make Event_State = High_Limit. After the event-triggering property is returned to a normal value, the HighLimitEnable flag is set to false, and

the event-triggering property is again manipulated to exceed the high limit. No event notification should be observed, and the Event_State must have a value of normal.

Configuration Requirements: Configure the object with pHighLimit, pLowLimit, and pDeadband values such that pHighLimit - pDeadband > pLowLimit and both the pLowLimit and pHighLimit values are within the valid range of values for the event triggering property. If the device cannot be configured with limit values that meet these conditions, then this test shall be skipped. The Event_Enable property shall be set to (TRUE, ?, TRUE) for this test. If the Event_Enable property cannot be configured such that the TO-NORMAL and the TO-OFFNORMAL transitions are TRUE, this test shall be skipped.

Test Steps:

1. MAKE pLimitEnable = (?, TRUE)
2. VERIFY pCurrentState = NORMAL
3. MAKE (pMonitoredValue a value that exceeds pHighLimit)
4. WAIT (pTimeDelay)
5. BEFORE **Notification Fail Time**
   RECEIVE ConfirmedEventNotification-Request,
   'Process Identifier' = (any valid process ID),
   'Initiating Device Identifier' = IUT,
   'Event Object Identifier' = (the object configured for this test),
   'Time Stamp' = (any valid time stamp),
   'Notification Class' = (the class corresponding to the object being tested),
   'Priority' = (the value configured to correspond to a TO-OFFNORMAL transition),
   'Event Type' = OUT_OF_RANGE,
   'Message Text' = (optional, any valid message text),
   'Notify Type' = ALARM | EVENT,
   'AckRequired' = TRUE | FALSE,
   'From State' = NORMAL,
   'To State' = HIGH_LIMIT,
   'Event Values' = (values appropriate to the event type)
6. TRANSMIT BACnet-SimpleACK-PDU
7. VERIFY pCurrentState = HIGH_LIMIT
8. MAKE (pMonitoredValue a value that is between pLowLimit and pHighLimit - pDeadband)
9. WAIT (pTimeDelayNormal)
10. BEFORE **Notification Fail Time**
    RECEIVE ConfirmedEventNotification-Request,
    'Process Identifier' = (any valid process ID),
    'Initiating Device Identifier' = IUT,
    'Event Object Identifier' = (the object configured for this test),
    'Time Stamp' = (any valid time stamp),
    'Notification Class' = (the class corresponding to the object being tested),
    'Priority' = (the value configured to correspond to a TO-NORMAL transition),
    'Event Type' = OUT_OF_RANGE,
    'Message Text' = (optional, any valid message text),
    'Notify Type' = ALARM | EVENT,
    'AckRequired' = TRUE | FALSE,
    'From State' = HIGH_LIMIT,
    'To State' = NORMAL,
    'Event Values' = (values appropriate to the event type)
11. TRANSMIT BACnet-SimpleACK-PDU
12. MAKE pLimitEnable = (?, FALSE)
13. VERIFY pCurrentState = NORMAL
14. MAKE (pMonitoredValue a value that exceeds pHighLimit)
15. WAIT (pTimeDelay + **Notification Fail Time**)
16. CHECK (verify that no notification message was transmitted)
17. VERIFY pCurrentState = NORMAL

Notes to Tester: The UnconfirmedEventNotification service may be substituted for the ConfirmedEventNotification service in which case the TD shall skip all of the steps in which a SimpleACK-PDU is sent.

[Add new **Clause 7.3.1.X4**, p. 59]

7.3.1.X4 Event_Message_Texts Tests

Purpose: To verify that the value of the Event_Message_Texts property is updated when an object generates an event notification.

Test Concept: Read the Event_Message_Texts from the object. Transition the object through each event state, which is enabled in the object, saving the Message Text parameter from the received notification. Verify that the Event_Message_Texts updates with the Event_Message_Texts value received from the notification.

Configuration Requirements: The IUT shall be configured with an event-generation object, O1, which shall be in a NORMAL Event_State at the beginning of the test. If the algorithm of the object does not support NORMAL to NORMAL transitions, then the TO-OFFNORMAL bit of the Event_Enable shall be TRUE. If the IUT does not contain any objects which can transition to any offnormal state, then this test shall be skipped.

Test Steps:

1. READ EMT = Event_Message_Texts
2. IF (Event_Enable is (TRUE, ?, ?)) THEN {
3. IF (pMonitoredValue is writable) THEN
   WRITE pMonitoredValue = (a value that is offnormal)
   ELSE
   MAKE (pMonitoredValue a value that is offnormal)
4. WAIT (pTimeDelay)
5. BEFORE **Notification Fail Time**
   RECEIVE ConfirmedEventNotification-Request,
   'Process Identifier' = (any valid process ID),
   'Initiating Device Identifier' = IUT,
   'Event Object Identifier' = O1,
   'Time Stamp' = (any valid time stamp),
   'Notification Class' = (the class corresponding to the object being tested),
   'Priority' = (the configured TO_OFFNORMAL priority),
   'Event Type' = (any valid event type),
   'Message Text' = (M: any valid value placed into EMT[1]),
   'Notify Type' = Notify_Type,
   'AckRequired' = (the configured value for the TO_OFFNORMAL transition),
   'From State' = NORMAL,
   'To State' = (any valid offnormal state),
   'Event Values' = (values appropriate to the event type)
6. VERIFY Event_Message_Texts = EMT
   }
7. IF (Event_Enable is (?, ?, TRUE)) THEN {
8. IF (pMonitoredValue is writable) THEN
   WRITE pMonitoredValue = (a value that will result in a TO_NORMAL transition)
   ELSE
   MAKE (pMonitoredValue a value that will result in a TO_NORMAL transition)
9. WAIT (pTimeDelayNormal)

10. BEFORE **Notification Fail Time**
RECEIVE ConfirmedEventNotification-Request,
'Process Identifier' = (any valid process ID),
'Initiating Device Identifier' = IUT,
'Event Object Identifier' = O1,
'Time Stamp' = (any valid time stamp),
'Notification Class' = (the class corresponding to the object being tested),
'Priority' = (the configured TO_NORMAL priority),
'Event Type' = (any valid event type),
'Message Text' = (M: any valid value placed into EMT[3]),
'Notify Type' = Notify_Type,
'AckRequired' = (the configured value for the TO_NORMAL transition),
'From State' = (any valid value),
'To State' = NORMAL,
'Event Values' = (values appropriate to the event type)
11. VERIFY Event_Message_Texts = EMT
}
12. IF (Event_Enable is (?, TRUE, ?)) THEN {
13. MAKE (a condition exist that will cause O1 to generate a TO-FAULT transition)
14. BEFORE **Notification Fail Time**
RECEIVE ConfirmedEventNotification-Request,
'Process Identifier' = (any valid process ID),
'Initiating Device Identifier' = IUT,
'Event Object Identifier' = O1,
'Time Stamp' = (any valid time stamp),
'Notification Class' = (the class corresponding to the object being tested),
'Priority' = (the configured TO_FAULT priority),
'Event Type' = *(IF (Protocol_Revision < 13) THEN*
(any valid event type)
ELSE
CHANGE_OF_RELIABILITY),
'Message Text' = M: any valid value placed into EMT[2]),
'Notify Type' = Notify_Type,
'AckRequired' = (the configured value for the TO_FAULT transition),
'From State' = (any valid value),
'To State' = FAULT,
'Event Values' = (values appropriate to the event type)
15. VERIFY Event_Message_Texts = EMT
}

[Add new **Clause 7.3.1.X5**, p. 59]

7.3.1.X5 Event_Message_Texts_Config Test

Purpose: To verify that the Message Text parameter of generated event notifications is controlled via the Event_Message_Texts_Config property.

Test Concept: Select an object, O1, in the IUT that supports the Event_Message_Texts_Config property. Make O1 perform each supported event transition (i.e., to-offnormal, to-normal, and to-fault). Verify that the 'Message Text' parameter matches the associated Event_Message_Texts_Config value. Note that due to the use of text substitution codes, the resulting text might not be an exact match.

Configuration Requirements: If possible, configure each entry in the Event_Message_Texts_Config property of Object O1 to be distinct. ES1 shall be the state to which O1 transitions. D1 is either the pTimeDelay, or pTimeDelayNormal parameter, or 0 (for transitions to and from FAULT state) depending on the event transition. ESINDEX shall be the array index associated

with ES1 (1 for offnormal states, 2 for FAULT, and 3 for NORMAL). The notification class for O1 is configured for UnconfirmedEventNotification.

Test Steps:

1. MAKE (a condition exist which will cause O1 to transition to ES1)
2. WAIT D1
3. BEFORE **Notification Fail Time**
   RECEIVE UnconfirmedEventNotification-Request
   'Process Identifier' = (any valid process identifier),
   'Initiating Device Identifier' = IUT,
   'Event Object Identifier' = O1,
   'Time Stamp' = (any valid time stamp),
   'Notification Class' = (the notification class configured for O1),
   'Priority' = (any valid priority),
   'Event Type' = (any standard event type),
   'Message Text' = T1,
   'Notify Type' = ALARM | EVENT,
   'AckRequired' = TRUE | FALSE,
   'From State' = (any valid event state),
   'To State' = ES1,
   'Event Values' = (any values appropriate to the event type)
4. CHECK (T1 is equivalent to Event_Message_Texts_Config[ESINDEX] with any text substitutions as defined by the vendor)

Notes to Tester: This behavior can alternately be tested using the ConfirmedEventNotification service, but it is not necessary to test both.

[Add new **Clause 7.3.1.X6**, p. 59]

7.3.1.X6 Event_Algorithm_Inhibit Tests

7.3.1.X6.1 Event_Algorithm_Inhibit Test

Purpose: To verify that Event_Algorithm_Inhibit property in objects with intrinsic or algorithmic reporting controls whether or not the event state detection algorithm is executed.

Test Concept: Select an event generating object, O1, which supports the Event_Algorithm_Inhibit property and does not support the Event_Algorithm_Inhibit_Ref property. With Event_Algorithm_Inhibit set to FALSE, make a condition exist that should result in an event transition to a normal or offnormal state. Verify that a transition occurs and that a notification is generated. Set Event_Algorithm_Inhibit to TRUE. If not already in a NORMAL state, verify that the object transitions to NORMAL. Make a condition exist that should result in an event transition, if the object Event_Algorithm_Inhibit were FALSE. If O1 supports fault detection, make a fault condition exist and verify that object detects the fault and transitions to FAULT.

Configuration Requirements: O1 is configured to detect and report unconfirmed events, is in the NORMAL state, and, if supported, is configured to detect fault conditions. D1 is either the pTimeDelay, or pTimeDelayNormal parameter, or 0 (for transitions to and from FAULT state) depending on the event transition.

Test Steps:

1. VERIFY pCurrentState = NORMAL
2. VERIFY Event_Algorithm_Inhibit = FALSE
3. MAKE (a condition exist which will result in a transition of O1. If possible, 'To State' shall be an offnormal event state)
4. WAIT D1

5. BEFORE **Notification Fail Time**
   RECEIVE UnconfirmedEventNotification-Request
   'Process Identifier' = (PID1: any valid process identifier),
   'Initiating Device Identifier' = IUT,
   'Event Object Identifier' = O1,
   'Time Stamp' = (any valid time stamp),
   'Notification Class' = (the notification class configured for O1),
   'Priority' = (the value configured for the transition),
   'Event Type' = (ET1: any valid event type),
   'Message Text' = (optional, any valid message text),
   'Notify Type' = (value from the Notify_Type property configured for O1),
   'AckRequired' = TRUE | FALSE,
   'From State' = NORMAL,
   'To State' = (ES1: any event state appropriate to the event type),
   'Event Values' = (any values appropriate to the event type)
6. WRITE Event_Algorithm_Inhibit = TRUE
7. IF (ES1 <> NORMAL) THEN
   BEFORE **Notification Fail Time**
   RECEIVE UnconfirmedEventNotification-Request
   'Process Identifier' = PID1,
   'Initiating Device Identifier' = IUT,
   'Event Object Identifier' = O1,
   'Time Stamp' = (any valid time stamp),
   'Notification Class' = (the notification class configured for O1),
   'Priority' = (the value configured for the transition),
   'Event Type' ET1,
   'Message Text' = (optional, any valid message text),
   'Notify Type' = (value from the Notify_Type property configured for O1),
   'AckRequired' = TRUE | FALSE,
   'From State' = ES1,
   'To State' = NORMAL,
   'Event Values' = (any values appropriate to the event type)
8. VERIFY pCurrentState = NORMAL
9. MAKE (a condition exist which would result in a transition of O1 other than TO-FAULT, if Event_Algorithm_Inhibit were FALSE)
10. WAIT D1
11. WAIT **Notification Fail Time**
12. CHECK (that the IUT did not send any event notifications for O1)
13. VERIFY pCurrentState = NORMAL
14. IF (O1 supports fault detection) THEN
    MAKE (a fault condition exist for O1)
    BEFORE **Notification Fail Time**
    RECEIVE UnconfirmedEventNotification-Request
    'Process Identifier' = PID1,
    'Initiating Device Identifier' = IUT,
    'Event Object Identifier' = O1,
    'Time Stamp' = (any valid time stamp),
    'Notification Class' = (the notification class configured for O1),
    'Priority' = (the value configured for the transition),
    'Event Type' = CHANGE_OF_RELIABILITY,
    'Message Text' = (optional, any valid message text),
    'Notify Type' = (value from the Notify_Type property configured for O1),
    'AckRequired' = TRUE | FALSE,
    'From State' = NORMAL,
    'To State' = FAULT,
    'Event Values' = (any values appropriate for CHANGE_OF_RELIABILITY)

MAKE (remove the fault condition)
WAIT (pTimeDelayNormal)
BEFORE **Notification Fail Time**
RECEIVE UnconfirmedEventNotification-Request

'Process Identifier' =	PID1,
'Initiating Device Identifier' =	IUT,
'Event Object Identifier' =	O1,
'Time Stamp' =	(any valid time stamp),
'Notification Class' =	(the notification class configured for O1),
'Priority' =	(the value configured for the transition),
'Event Type' =	CHANGE_OF_RELIABILITY,
'Message Text' =	(optional, any valid message text),
'Notify Type' =	(value from the Notify_Type property configured for O1),
'AckRequired' =	TRUE | FALSE,
'From State' =	FAULT,
'To State' =	NORMAL,
'Event Values' =	(any values appropriate for CHANGE_OF_RELIABILITY)

Notes to Tester: This behavior can alternately be tested using the ConfirmedEventNotification service, but it is not necessary to test both.

7.3.1.X6.2 Event_Algorithm_Inhibit Summarization Test

Purpose: To verify that event generating objects are reported by summarization routines as needed even when the algorithm has been inhibited.

Test Concept: Select an event generating object, O1, which is configured for event reporting and is configured to need acknowledgement for either the TO_NORMAL or TO_OFFNORMAL transition. The TO_FAULT bit being FALSE in Acked_Transitions is not suitable as the testable point in this test because Event_Algorithm_Inhibit does not influence detection and reporting of FAULT. Similarly, a transition from FAULT is not suitable for this test. Verify that the event is reported when the device responds to a GetEventInformation request.

Configuration Requirements: O1 is configured such that it requires at least one of its transitions to require operator acknowledgement. O1's algorithm is inhibited. The number of event generating objects in the IUT that have an Event_State other than NORMAL, or which have an Acked_Transitions other than (T, T, T), is such that they can all be reported in a single GetEventInformation-ACK response.

Test Steps:

1. AT = READ Acked_Transitions
2. CHECK (AT <> (T, T, T))
2. VERIFY Acked_Transitions = (?, T, ?)
3. VERIFY Event_Algorthm_Inhibit = TRUE
4. TRANSMIT GetEventInformation
5. RECEIVE GetEventInformation-Ack,
   'List of Event Summaries' = (list of object identifiers which includes O1)
   'More Events' = FALSE

7.3.1.X6.3 Event_Algorithm_Inhibit Acknowledgement Test

Purpose: To verify that event generating objects can be acknowledged when the algorithm has been inhibited.

Test_Concept: Select an event generating object, O1, which is configured for event reporting, is configured to need acknowledgement for at least one of its transitions, and its Acked_Transitions property is not (T, T, T). Verify that the IUT accepts an acknowledgment for the transition that requires acknowledgement. The TO_FAULT bit in Acked_Transitions is not suitable as the testable point in this test because Event_Algorithm_Inhibit does not influence detection and reporting of FAULT. Similarly, a transition from FAULT is not suitable for this test.

Configuration Requirements: O1 is configured such that it requires at least one of its transitions to require operator acknowledgement. O1's algorithm is inhibited. The number of event generating objects in the IUT that have an Event_State other than NORMAL, or which have an Acked_Transitions other than (T, T, T), is such that they can all be reported in a single GetEventInformation-ACK response. For this test, ES_TO_ACK is the Event_State that is to be acknowledged, TS_TO_ACK is the timestamp associated with that transition. The IUT is configured such that TD will receive a confirmed notification when O1 transitions.

Test Steps:

1. AT = READ Acked_Transitions
2. CHECK (AT <> (T, T, T))
3. VERIFY Event_Algorthm_Inhibit = TRUE
4. TRANSMIT AcknowledgeAlarm
   - 'Acknowledging Process Identifier' = (any valid value),
   - 'Event Object Identifier' = O1,
   - 'Event State Acknowledged' = ES_TO_ACK,
   - 'Time Stamp' = TS_TO_ACK,
   - 'Time of Acknowledgment' = (the current timestamp)
5. RECEIVE BACnet-SimpleACK-PDU
6. BEFORE **Notification Fail Time**
   RECEIVE ConfirmedEventNotification-Request,
   - 'Process Identifier' = (the configured process ID),
   - 'Initiating Device Identifier' = IUT,
   - 'Event Object Identifier' = O1,
   - 'Time Stamp' = (any valid time stamp),
   - 'Notification Class' = (the class configured for O1),
   - 'Priority' = (the value configured for the transition),
   - 'Event Type' = (any valid event type),
   - 'Message Text' = (optional, any valid message text),
   - 'Notify Type' = ACK_NOTIFICATION,
   - 'To State' = ES_TO_ACK
7. TRANSMIT BACnet-SimpleACK-PDU
8. AT2 = READ Acked_Transitions
9. CHECK (AT2 is equal to AT, except the bit associated with ES_TO_ACK is TRUE)

[Add new **Clause 7.3.1.X7**, p. 59]

7.3.1.X7 Event_Algorithm_Inhibit_Ref Tests

7.3.1.X7.1 Event_Algorithm_Inhibit_Ref Test

Purpose: To verify that the object referenced by Event_Algorithm_Inhibit_Ref controls Event_Algorithm_Inhibit and, thus, whether or not the event state detection algorithm is executed.

Test Concept: Execute test 7.3.1.X2.1 against an object, O2, which supports both Event_Algorithm_Inhibit_Ref and Event_Algorithm_Inhibit, and instead of writing Event_Algorithm_Inhibit, write the property referenced by Event_Algorithm_Inhibit_Ref to change the value in the Event_Algorithm_Inhibit property.

Configuration Requirements: If the IUT has no object in which the Event_Algorithm_Inhibit_Ref property is absent or can be made uninitialized, or has no object in which Event_Detection_Enable can be made TRUE, this test shall be skipped.

7.3.1.X7.2 Event_Algorithm_Inhibit Writable Test

Purpose: To verify that if the Event_Algorithm_Inhibit_Ref property is absent or is uninitialized, then the Event_Algorithm_Inhibit property shall be writable.

Configuration Requirements: Select an event-initiating object, O1, in which Event_Algorithm_Inhibit_Ref property is absent or is uninitialized. If the IUT has no such object, this test shall be skipped.

Test Steps:

1. WRITE Event_Algorithm_Inhibit = TRUE
2. WRITE Event_Algorithm_Inhibit = FALSE

[Add new **Clause 7.3.1.X8**, p. 59]

7.3.1.X8 Reliability_Evaluation_Inhibit Tests

7.3.1.X8.1 Reliability_Evaluation_Inhibit Test

Purpose: To verify that Reliability_Evaluation_Inhibit controls whether or not fault conditions are detected.

Test Concept: Select an event generating object, O1, which supports the Reliability_Evaluation_Inhibit property. With Reliability_Evaluation_Inhibit FALSE, make a fault condition exist. Verify that Reliability changes and that a notification is generated. Set Reliability_Evaluation_Inhibit to TRUE. Verify that the Reliability changes to NO_FAULT_DETECTED and that a TO_NORMAL notification is generated. Remove the fault condition and ensure that no notification is generated. Make a fault condition exist and verify that Reliability remains NO_FAULT_DETECTED and that no notification is generated.

Configuration Requirements: O1 is configured to detect and report unconfirmed events, is in the NORMAL state, and Reliability_Evaluation_Inhibit equals FALSE, so that reliability evaluation for that object is configured to detect fault conditions. If no object exists in the IUT for which fault conditions can be generated, then this test shall be skipped.

Test Steps:

1. VERIFY pCurrentState = NORMAL
2. VERIFY Reliability = NO_FAULT_DETECTED
3. MAKE (a condition exist that would cause O1 to generate a TO_FAULT transition)
4. BEFORE **Notification Fail Time**
   RECEIVE UnconfirmedEventNotification-Request
   'Process Identifier' = (the value configured for the transition),
   'Initiating Device Identifier' = IUT,
   'Event Object Identifier' = O1,
   'Time Stamp' = (any valid timestamp),
   'Priority' = (any valid priority),
   'Event Type' = CHANGE_OF_RELIABILITY,
   'Message Text' = (optional, any valid message text),
   'Notify Type' = ALARM | EVENT,
   'AckRequired' = TRUE | FALSE,
   'From State' = NORMAL,
   'To State' = FAULT,
   'Event Values' = (any values appropriate to CHANGE_OF_RELIABILITY)
5. WRITE Reliability_Evaluation_Inhibit = TRUE
6. BEFORE **Internal Processing Fail Time** + **Notification Fail Time**
   RECEIVE UnconfirmedEventNotification-Request
   'Process Identifier' = (the value configured for the transition),
   'Initiating Device Identifier' = IUT,
   'Event Object Identifier' = O1,
   'Time Stamp' = (any valid timestamp),
   'Priority' = (any valid priority),
   'Event Type' = CHANGE_OF_RELIABILITY,
   'Message Text' = (optional, any valid message text),

'Notify Type' = ALARM | EVENT,
'AckRequired' = TRUE | FALSE,
'From State' = FAULT,
'To State' = NORMAL,
'Event Values' = (any values appropriate to CHANGE_OF_RELIABILITY)

7. VERIFY Reliability = NO_FAULT_DETECTED
8. VERIFY pCurrentState = NORMAL
9. MAKE (remove the fault condition)
10. WAIT (pTimeDelayNormal)
11. WAIT **Notification Fail Time**
12. CHECK (that the IUT did not send any event notifications for O1)
13. MAKE (a condition exist that would cause O1 to generate a TO_NORMAL transition)
14. WAIT **Notification Fail Time**
15. VERIFY Reliability = NO_FAULT_DETECTED
16. VERIFY pCurrentState = NORMAL
17. CHECK (that the IUT did not send any event notifications for O1)

Notes to Tester: This behavior can alternately be tested using the ConfirmedEventNotification service, but it is not necessary to test both.

7.3.1.X8.2 Reliability_Evaluation_Inhibit Summarization Test

Purpose: To verify that event generating objects are reported by summarization routines as needed, even when the reliability evaluation has been inhibited.

Test_Concept: Select an event generating object, O1, which is configured for event reporting, is configured to require acknowledgement for TO_FAULT transition, and its Acked_Transitions property is (T, F, T). Verify that the event is reported when the device responds to a GetEventInformation request.

Configuration Requirements: O1 is configured such that it requires acknowledgement of the TO_FAULT transition, and the Acked_Transitions is (T, F, T). O1's Reliability_Evaluation_Inhibit equals TRUE, so that reliability evaluation for that object is inhibited. The number of event generating objects in the IUT that have an Event_State other than NORMAL, or which have an Acked_Transitions other than (T, T, T), is such that they can all be reported in a single GetEventInformation-ACK response.

Test Steps:

1. VERIFY Acked_Transitions = (T, F, T)
2. VERIFY Event_Algorithm_Inhibit = TRUE
3. TRANSMIT GetEventInformation
4. RECEIVE GetEventInformation-Ack,
   'List of Event Summaries' = (list of object identifiers which includes O1)
   'More Events' = FALSE

[Add new **Clause 7.3.1.X9**, p. 59]

7.3.1.X9 Event_Detection_Enable Tests

7.3.1.X9.1 Event_Detection_Enable Inhibits Event Generation

Purpose: To verify that Event_Detection_Enable enables and disables event detection in objects which are configured for event reporting.

Test Concept: Select an event generating object, O1, that is configured for event reporting. If possible, make the object generate an event, to an offnormal, so that if the object can have a non-normal state, it enters that state early in the test. This

will help detect incorrect implementations that initiate a TO_NORMAL transition when the algorithm is disabled. Set the Event_Detection_Enable property to FALSE. Verify the Event_State is NORMAL and the Acked_Transitions, Event_Time_Stamps, and Event_Message_Texts are equal to their respective initial conditions, as mandated in the standard. Repeat the process that made the object generate an event and observe that no notification messages are transmitted.

Configuration Requirements: O1 is configured to detect and report unconfirmed events and requires acknowledgments for all transitions. Event_Detection_Enable is equal to TRUE. D1 is either the pTimeDelay, or pTimeDelayNormal parameter, or 0 (for transitions to and from FAULT state) depending on the event transition. For this test, NO_TS equals a BACnetDateTime with all unspecified values, a BACnet Time with all unspecified values, or a sequence number of 0.

Test Steps:

1. VERIFY Event_Detection_Enable = TRUE
2. MAKE (a condition exist which will cause O1 to transition, to an offnormal state if possible)
3. WAIT D1
4. BEFORE **Notification Fail Time**
   RECEIVE UnconfirmedEventNotification-Request
   'Process Identifier' = (any valid process identifier),
   'Initiating Device Identifier' = IUT,
   'Event Object Identifier' = O1,
   'Time Stamp' = (any valid time stamp),
   'Notification Class' = (the notification class configured for O1),
   'Priority' = (the value configured for the transition),
   'Event Type' = (any valid event type),
   'Message Text' = (optional, any valid message text),
   'Notify Type' = (value from the Notify_Type property configured for O1),
   'AckRequired' = TRUE,
   'From State' = (any valid event state),
   'To State' = (any event state appropriate to the event type),
   'Event Values' = (any values appropriate to the event type)
5. IF Event_Detection_Enable is writable THEN
   WRITE Event_Detection_Enable = FALSE
   ELSE
   MAKE (Event_Detection_Enable to FALSE. This property is expected to be set during system configuration and is not expected to change dynamically.)
6. WAIT (D1 + **Notification Fail Time + Internal Processing Fail Time)**
7. CHECK (that the IUT did not send any further event notifications for O1)
8. VERIFY pCurrentState = NORMAL
9. VERIFY Acked_Transitions = (T,T,T)
10. IF (Protocol_Revision is present AND Protocol_Revision ≥ 1) THEN
    VERIFY Event_Time_Stamps = [NO_TS, NO_TS, NO_TS]
11. IF the Event_Message_Texts property exists THEN
    VERIFY Event_Message_Texts = ['', '', '']
12. MAKE (a condition exist which would cause O1 to transition, if Event_Detection_Enable were TRUE)
13. WAIT (D1 + **Notification Fail Time)**
14. CHECK (that the IUT did not send any event notifications for O1)
15. VERIFY pCurrentState = NORMAL
16. VERIFY Acked_Transitions = (T,T,T)
17. IF (Protocol_Revision is present AND Protocol_Revision ≥ 1) THEN
    VERIFY Event_Time_Stamps = [NO_TS, NO_TS, NO_TS]
18. IF the Event_Message_Texts property exists THEN
    VERIFY Event_Message_Texts = ['', '', '']

Notes to Tester: This behavior can alternately be tested using the ConfirmedEventNotification service, but it is not necessary to test both.

7.3.1.X9.2 Event_Detection_Enable Inhibits FAULT

Purpose: To verify that Event_Detection_Enable disables fault reporting.

Test Concept: 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. Select an event generating object, O1, that is configured for event reporting, and which can be made to go into FAULT. Set the Event_Detection_Enable property to FALSE. Create a condition which will cause O1 to transition to FAULT, if Event_Detection_Enable is TRUE. Verify the Event_State is NORMAL and the Acked_Transitions, Event_Time_Stamps, and Event_Message_Texts are equal to their respective initial conditions, as mandated in the standard, and no notification messages are transmitted.

Configuration Requirements: O1 is an object capable of detecting and reporting an event for a FAULT condition, and the Event_Detection_Enable can be set to FALSE. Reliability_Evaluation_Inhibit is equal to TRUE. For this test, NO_TS equals a BACnetDateTime with all unspecified values, a BACnet Time with all unspecified values, or a sequence number of 0.

Test Steps:

1. VERIFY Event_Detection_Enable = FALSE
2. IF Reliability is writable THEN
3. WRITE Reliability = (any value other than NO_FAULT_DETECTED)
   ELSE
4. MAKE (a condition exist which would cause O1 to transition to FAULT, if Event_Detection_Enable were TRUE)
5. WAIT **Notification Fail Time**
6. CHECK (that the IUT did not send any event notifications due to this condition)
7. VERIFY pCurrentState = NORMAL
8. VERIFY Acked_Transitions = (T,T,T)
9. IF (Protocol_Revision is present AND Protocol_Revision ≥ 1) THEN
   VERIFY Event_Time_Stamps = [NO_TS, NO_TS, NO_TS]
10. IF Event_Message_Texts property exists THEN
    VERIFY Event_Message_Texts = ['', '', '']

[Change **Clause 7.3.2.11.1**, p. 77]

7.3.2.11.1 Event_Type Test

~~Dependencies: ReadProperty Service Execution Tests, 9.18; WriteProperty Service Execution Tests, 9.22.~~

~~BACnet Reference Clauses: 12.12.5.~~

Purpose: This test case verifies that Event_Type property of an Event Enrollment object properly tracks changes to the Event_Parameters property.

Test Concept: Write to the Event_Parameters property and verify that the Event_Type tracks.

Configuration Requirements: The IUT shall be configured with an Event Enrollment object, E1, having a writable Event_Parameters property that will accept a value with a different algorithm choice. The tester shall choose a valid Event_Parameters value, EP1, with an event type value, ET1, that the IUT will accept where the ET1 differs from the Event_Type already existing in E1.

Test Steps:

1. VERIFY Event_Type <> ET1
2. TRANSMIT WriteProperty-Request,
   'Object Identifier' = E1,
   'Property Identifier' = Event_Parameters,
   'Property Value' = EP1
3. RECEIVE BACnet-SimpleACK-PDU
4. VERIFY Event_Type = ET1

[Change **Clause 7.3.2.21.1**, p. 88]

7.3.2.21.1 Priority Tests

~~Dependencies: ConfirmedEventNotification Service Initiation Tests, 8.4; UnconfirmedEventNotification Service Initiation Tests, 8.5; ReadProperty Service Execution Tests, 9.18; WriteProperty Service Execution Tests, 9.22.~~

~~BACnet Reference Clause: 12.21.6.~~

Purpose: To verify that the IUT implements the functionality of the Priority property of the Notification Class object when initiating ~~even~~*event* notifications.

Test Concept: The TD will select one instance of the Notification Class object and one instance of an event-generating object that is linked to ~~it~~*the Notification Class object*. The properties of the event-generating object will be manipulated to cause the Event_State to change from NORMAL to OFFNORMAL, from OFFNORMAL back to NORMAL, from NORMAL to FAULT, and from FAULT back to NORMAL. For each state transition, the appropriate use of priority in the resulting event notification will be verified. It must be possible to trigger the events of this test, or the test result is considered to be a failure.

Configuration Requirements: The IUT shall be configured with one or more instance of the Notification Class object and at least one event-generating object that is linked to the Notification Class object. The event-generating object may be any object that supports intrinsic reporting or it may be an Event Enrollment object. The Notification Class object shall be configured with separate, distinct Priority values for TO-OFFNORMAL, TO-NORMAL, and TO-FAULT transitions. All Event_Enable bits shall be set to TRUE. The referenced event-triggering property shall be set to a value that results in a NORMAL condition.

~~In the test description below, "X" is used to designate the event-triggering property.~~

Test Steps:

~~1. WAIT (Time_Delay + **Notification Fail Time**)~~

~~2~~*1*. VERIFY ~~Event_State~~*pCurrentState* = NORMAL
~~3~~*2*. IF (~~X~~*pMonitoredValue* is writable) THEN
WRITE ~~X~~*pMonitoredValue* = (a value that is OFFNORMAL)
ELSE
MAKE (~~X~~*pMonitoredValue* have a value that is OFFNORMAL)
~~4~~*3*. WAIT (~~Time_Delay~~*pTimeDelay*)
~~5~~*4*. BEFORE **Notification Fail Time**
RECEIVE ConfirmedEventNotification-Request,

'Process Identifier' =	(any valid process ID),
'Initiating Device Identifier' =	IUT,
'Event Object Identifier' =	(the event-generating object configured for this test),
'Time Stamp' =	(any valid time stamp),
'Notification Class' =	(the class corresponding to the object being tested),
'Priority' =	(the value configured to correspond to a TO-OFFNORMAL transition),
'Event Type' =	(any valid event type),
'Message Text' =	*(optional, any valid message text),*
'Notify Type' =	EVENT | ALARM,
'AckRequired' =	TRUE | FALSE,
'From State' =	NORMAL,
'To State' =	OFFNORMAL,
'Event Values' =	(values appropriate to the event type)

5. TRANSMIT BACnet-SimpleACK-PDU
6. VERIFY ~~Event_State~~*pCurrentState* = OFFNORMAL
7. IF (~~X~~*pMonitoredValue* is writable) THEN
WRITE ~~X~~*pMonitoredValue* = (a value that is NORMAL)
ELSE
MAKE (~~X~~*pMonitoredValue* have a value that is NORMAL)
8. WAIT (~~Time_Delay~~*pTimeDelayNormal*)
9. BEFORE -**Notification Fail Time**
RECEIVE ConfirmedEventNotification-Request,

'Process Identifier' =	(any valid process ID),
'Initiating Device Identifier' =	IUT,
'Event Object Identifier' =	(the event-generating object configured for this test),
'Time Stamp' =	(any valid time stamp),
'Notification Class' =	(the class corresponding to the object being tested),
'Priority' =	(the value configured to correspond to a TO-NORMAL transition),
'Event Type' =	(any valid event type),
'Message Text' =	*(optional, any valid message text),*
'Notify Type' =	EVENT | ALARM,
'AckRequired' =	TRUE | FALSE,
'From State' =	OF*F*NORMAL,
'To State' =	NORMAL,
'Event Values' =	(values appropriate to the event type)

10. TRANSMIT BACnet-SimpleACK-PDU
~~10~~*11*. VERIFY ~~Event_State~~*pCurrentState* = NORMAL
~~11~~*12*. IF (the event-triggering object can be placed into a fault condition) THEN {
~~12~~*13*. MAKE (~~the event-triggering~~ *a condition exist that will cause the* object ~~change to a fault condition~~*to generate a TO_NORMAL transition*)
~~13~~*14*. BEFORE **Notification Fail Time**
RECEIVE ConfirmedEventNotification-Request,

'Process Identifier' =	(any valid process ID),
'Initiating Device Identifier' =	IUT,
'Event Object Identifier' =	(the event-generating object configured for this test),
'Time Stamp' =	(any valid time stamp),
'Notification Class' =	(the class corresponding to the object being tested),
'Priority' =	(the value configured to correspond to a TO-FAULT transition),

'Event Type' = *(IF (Protocol_Revision < 13) THEN*
(any valid event type)
ELSE
CHANGE_OF_RELIABILITY),
'Message Text' = *(optional, any valid message text),*
'Notify Type' = EVENT | ALARM,
'AckRequired' = TRUE | FALSE,
'From State' = NORMAL,
'To State' = FAULT,
'Event Values' = (values appropriate to the event type)

15. TRANSMIT BACnet-SimpleACK-PDU

~~14~~*16*. VERIFY ~~Event_State~~*pCurrentState* = FAULT

~~15~~*17*. MAKE (~~the event triggering~~*a condition exist that will cause the* object ~~change to a normal condition~~ *to generate a TO_NORMAL transition*)

18. WAIT (pTimeDelayNormal)

~~16~~*19*. BEFORE **Notification Fail Time**
RECEIVE ConfirmedEventNotification-Request,
'Process Identifier' = (any valid process ID),
'Initiating Device Identifier' = IUT,
'Event Object Identifier' = (the event-generating object configured for this test),
'Time Stamp' = (any valid time stamp),
'Notification Class' = (the class corresponding to the object being tested),
'Priority' = (the value configured to correspond to a TO-NORMAL transition),
'Event Type' = *(IF (Protocol_Revision < 13) THEN*
(any valid event type)
ELSE
CHANGE_OF_RELIABILITY),
'Message Text' = *(optional, any valid message text),*
'Notify Type' = EVENT | ALARM,
'AckRequired' = TRUE | FALSE,
'From State' = FAULT,
'To State' = NORMAL,
'Event Values' = (values appropriate to the event type)

20. TRANSMIT BACnet-SimpleACK-PDU

~~17~~*21*. VERIFY ~~Event_State~~*pCurrentState* = NORMAL

}

Notes to Tester: The UnconfirmedEventNotification service may be substituted for the ConfirmedEventNotification service*, in which case the TD shall skip all of the steps in which a BACnet-SimpleACK-PDU is sent*. ~~The 'Message Text' parameter is omitted in the test description because it is optional. The IUT may include this parameter in the notification messages.~~

[Change **Clause 7.3.2.21.2.1**, p. 90]

7.3.2.21.2.1 Ack_Required False Test

~~Dependencies: ConfirmedEventNotification Service Initiation Tests, 8.4; UnconfirmedEventNotification Service Initiation Tests, 8.5; ReadProperty Service Execution Tests, 9.18; WriteProperty Service Execution Tests, 9.22.~~

~~BACnet Reference Clause: 12.21.7.~~

Purpose: To verify that if the Ack_Required property indicates that event notifications do not require acknowledgment, then the AckRequired parameter of the notification message conveys that fact. If the IUT does not support unacknowledged event notifications, this test shall be omitted.

Test Concept: The TD will select one instance of the Notification Class object and one instance of an event-generating object that is linked to ~~it~~*the Notification Class object*. The properties of the event-generating object will be manipulated to cause the

Event_State to change from NORMAL to OFFNORMAL, from OFFNORMAL back to NORMAL, from NORMAL to FAULT, and from FAULT back to NORMAL. For each state transition the appropriate value for the 'AckRequired' parameter is verified.

Configuration Requirements: The configuration requirements are identical to those in ~~7.3.2.20.1~~*7.3.2.21.1;* except for an additional requirement that the value of the Ack_Required property shall be B'000' indicating that no acknowledgments are required.

Test Steps: The test steps are identical to those in ~~7.3.2.20.1~~*7.3.2.21.1,* with the additional constraint that the 'AckRequired' parameter shall be FALSE in all event notification messages.

Notes to Tester: The UnconfirmedEventNotification service may be substituted for the ConfirmedEventNotification service*, in which case the TD shall skip all of the steps in which a BACnet-SimpleACK-PDU is sent*. ~~The 'Message Text' parameter is omitted in the test description because it is optional. The IUT may include this parameter in the notification messages.~~

[Remove **Clause 7.3.2.21.2.2**, p. 90]
[Reviewer Note: The purpose of test 7.3.2.21.2.2 is sufficiently covered by 7.3.2.21.1 and other tests.]

~~**7.3.2.21.2.2 Ack_Required True Test**~~

~~Dependencies: ConfirmedEventNotification Service Initiation Tests, 8.4; UnconfirmedEventNotification Service Initiation Tests, 8.5; ReadProperty Service Execution Tests, 9.18; WriteProperty Service Execution Tests, 9.22.~~

~~BACnet Reference Clause: 12.21.7.~~

~~Purpose: To verify that if the Ack_Required property indicates that event notifications require acknowledgment, then the AckRequired parameter of the notification message conveys that fact. If the IUT does not support acknowledged event notifications, this test shall be omitted.~~

~~Test Concept: The TD will select one instance of the Notification Class object and one instance of an event-generating object that is linked to it. The properties of the event-generating object will be manipulated to cause the Event_State to change from NORMAL to OFFNORMAL, from OFFNORMAL back to NORMAL, from NORMAL to FAULT, and from FAULT back to NORMAL. For each state transition the appropriate value for the 'AckRequired' parameter is verified.~~

~~Configuration Requirements: The configuration requirements are identical to those in 7.3.2.20.1 except for an additional requirement that the value of the Ack_Required property shall be B'111' indicating that acknowledgments are required.~~

~~Test Steps: The test steps are identical to those in 7.3.2.20.1 with the additional constraint that the 'AckRequired' parameter shall be TRUE in all event notification messages.~~

~~Notes to Tester: The UnconfirmedEventNotification service may be substituted for the ConfirmedEventNotification service. The 'Message Text' parameter is omitted in the test description because it is optional. The IUT may include this parameter in the notification messages.~~

[Change **Clause 7.3.2.21.3.1**, p. 91]
[Reviewer Note: Updated Test Concept to include changes from 135-2010af.]

7.3.2.21.3.1 ValidDays Test

~~Dependencies: ConfirmedEventNotification Service Initiation Tests, 8.4; UnconfirmedEventNotification Service Initiation Tests, 8.5; ReadProperty Service Execution Tests, 9.18; WriteProperty Service Execution Tests, 9.22; TimeSynchronization Service Execution Tests, 9.30; UTCTimeSynchronization Service Execution Tests, 9.31.~~

~~BACnet Reference Clause: 12.21.8.~~

Purpose: To verify the operation of the Valid Days parameter of a BACnetDestination as used in the Recipient_List property of the Notification Class object.

Test Concept: The TD will select one instance of the Notification Class object and one instance of an event-generating object that is linked to ~~it~~*the Notification Class object*. The Recipient_List of the Notification Class object shall contain a single recipient with the Valid Days parameter configured so that at least one day is TRUE and at least one day is FALSE. The properties of the event-generating object will be manipulated to cause the Event_State to change from NORMAL to OFFNORMAL. The tester verifies that if the local date is one of the valid days a notification message is transmitted and ~~the~~ ~~if~~ *if the* local date is not a valid day then no notification message is transmitted. *For devices that implement a read-only Recipient_List property for all instances of Notification Class objects and are exclusively configured for all days (Valid Days set to all Days), this test shall be omitted. For devices that implement a writeable Recipient_List property for all instances of Notification Class objects, and exclusively accept all days as the only permitted configuration, this test shall be omitted.*

Configuration Requirements: The IUT shall be configured with one or more instance of the Notification Class object and at least one event-generating object that is linked to the Notification Class object. The event-generating object may be any object that supports intrinsic reporting or it may be an Event Enrollment object. The event-generating object shall have the Event_Enable property configured to transmit notification messages for all event transitions. The event-generating object shall be configured to be in a NORMAL event state at the start of the test. The Notification Class object shall be configured with a single recipient in the Recipient_List. The Valid Days parameter shall be configured so that at least one day of the week has a value of TRUE and at least one day of the week has a value of FALSE. The Transitions parameter shall be configured for the recipient to receive notifications for all event transitions.

~~In the test description below, "X" is used to designate the event-triggering property.~~

Test Steps:

1. (TRANSMIT TimeSynchronization-Request,
   'Time' = (any time within the window defined by From Time and To Time in the BACnet Destination that corresponds to one of the valid days)) |
   (TRANSMIT UTCTimeSynchronization-Request,
   'Time' = (any time within the window defined by From Time and To Time in the BACnet Destination that corresponds to one of the valid days*, converted to UTC*)) |
   *(*MAKE (the local date and time = (any time within the window defined by From Time and To Time in the BACnetDestination that corresponds to one of the valid days))
2. WAIT (~~Time_Delay~~*pTimeDelay* + **Notification Fail Time**)
3. VERIFY ~~Event_State~~*pCurrentState* = NORMAL
4. IF (~~X~~*pMonitoredValue* is writable) THEN
   WRITE ~~X~~*pMonitoredValue* = (a value that is OFFNORMAL)
   ELSE
   MAKE (~~X~~*pMonitoredValue* have a value that is OFFNORMAL)
5. WAIT (~~Time_Delay~~*pTimeDelay*)
6. BEFORE **Notification Fail Time**
   RECEIVE ConfirmedEventNotification-Request,
   'Process Identifier' = (any valid process ID),
   'Initiating Device Identifier' = IUT,
   'Event Object Identifier' = (the event-generating object configured for this test),
   'Time Stamp' = (~~the current local time~~*any valid time stamp*),
   'Notification Class' = (the class corresponding to the object being tested),
   'Priority' = (the value configured to correspond to a TO-OFFNORMAL transition),
   'Event Type' = (any valid event type),
   'Message Text' = *(optional, any valid message text),*
   'Notify Type' = EVENT | ALARM,
   'AckRequired' = TRUE | FALSE,
   'From State' = NORMAL,
   'To State' = OFFNORMAL,
   'Event Values' = (values appropriate to the event type)

7. TRANSMIT BACnet-SimpleACK-PDU
8. VERIFY ~~Event_State~~*pCurrentState* = OFFNORMAL
9. (TRANSMIT TimeSynchronization-Request,
   'Time' = (any time within the window defined by From Time and To time in the BACnet Destination that corresponds to one of the invalid days)) |
   (TRANSMIT UTCTimeSynchronization-Request,
   'Time' = (any time within the window defined by From Time and To Time in the BACnet Destination that corresponds to one of the invalid days, converted to UTC)) |
   *(*MAKE (the local date and time = ~~(~~any time within the window defined by From Time and To Time in the BACnetDestination that corresponds to one of the invalid days))
10. IF (~~X~~*pMonitoredValue* is writable) THEN
    WRITE ~~X~~*pMonitoredValue* = (a value that is NORMAL)
    ELSE
    MAKE (~~X~~*pMonitoredValue* have a value that is NORMAL)
11. WAIT (~~Time_Delay~~*pTimeDelay* + **Notification Fail Time**)
12. CHECK (verify that no notification message was transmitted)

Notes to Tester: The UnconfirmedEventNotification service may be substituted for the ConfirmedEventNotification service, in which case the TD shall skip all of the steps in which a BACnet-SimpleACK-PDU is sent. ~~The 'Message Text' parameter is omitted in the test description because it is optional. The IUT may include this parameter in the notification messages.~~

[Change **Clause 7.3.2.21.3.2**, p. 92]

7.3.2.21.3.2 FromTime and ToTime Test

~~Dependencies: ValidDays Test, 7.3.2.21.3.1; ConfirmedEventNotification Service Initiation Tests, 8.4; UnconfirmedEventNotification Service Initiation Tests, 8.5; ReadProperty Service Execution Tests, 9.18; TimeSynchronization Service Execution Tests, 9.30;~~

~~BACnet Reference Clause: 12.21.8.~~

Purpose: To verify the operation of the From Time and To Time parameters of a BACnetDestination as used in the Recipient_List property of the Notification Class object.

Test Concept: The case where the local date and time fall within the window defined by the From Time and To Time parameters is covered by the ValidDays test in 7.3.2.21.3.1. This test uses the same IUT configuration and sets the local time to a value that is one of the ValidDays but outside of the window defined by the From Time and To Time parameters. The objective is to verify that an event notification message is not transmitted when the event is triggered. *For devices that implement a read-only Recipient_List property for all instances of Notification Class objects and are exclusively configured for all times (From Time set to 00:00:00.0, To_Time set to 23:59:59.99), this test shall be omitted. For devices that implement a writeable Notification Class Recipient_List property for all instances of Notification Class objects, and exclusively accept all times as the only permitted configuration, this test shall be omitted.*

Configuration Requirements: The configuration requirements are identical to the requirements in 7.3.2.21.3.1.

Test Steps:

1. (TRANSMIT TimeSynchronization-Request,
   'Time' = (any time outside the window defined by From Time and To Time in the BACnet Destination that corresponds to one of the valid days)) |
   (TRANSMIT UTCTimeSynchronization-Request,
   'Time' = (any time within the window defined by From Time and To Time in the BACnet Destination that corresponds to one of the valid days*, converted to UTC*)) |
   MAKE (the local date and time = (any time outside the window defined by From Time and To Time in the BACnetDestination that corresponds to one of the valid days))

2. WAIT (~~Time_Delay~~*pTimeDelay* + **Notification Fail Time**)
3. VERIFY ~~Event_State~~*pCurrentState* = NORMAL
4. IF (~~X~~*pMonitoredValue* is writable) THEN
   WRITE ~~X~~*pMonitoredValue* = (a value that is OFFNORMAL)
   ELSE
   MAKE (~~X~~*pMonitoredValue* have a value that is OFFNORMAL)
5. WAIT (~~Time_Delay~~*pTimeDelay* + **Notification Fail Time**)
6. CHECK (verify that no notification message was transmitted)

[Change **Clause 7.3.2.21.3.3**, p.93]

7.3.2.21.3.3 ~~IssueConfirmedNotifications~~*IssueConfirmedNotifications* Test

~~Dependencies: ConfirmedEventNotification Service Initiation Tests, 8.4; UnconfirmedEventNotification Service Initiation Tests, 8.5; WriteProperty Service Execution Tests, 9.22.~~

~~BACnet Reference Clause: 12.21.8.~~

Purpose: To verify that ConfirmedEventNotification messages are used if the Issue Confirmed Notifications parameter has the value TRUE and UnconfirmedEventNotification messages are used if the value is FALSE. If the IUT does not support both confirmed and unconfirmed event notifications this test may be omitted. *For devices that implement a read-only Recipient_List property for all instances of Notification Class objects, and there is a value of FALSE for the Issue Confirmed Notifications parameter in all instances, this test shall be omitted.*

Configuration Requirements: The IUT shall be configured with two or more instances of the Notification Class object and event-generating objects that are linked to the Notification Class objects. The event-generating objects may be objects that support intrinsic reporting or they may be Event Enrollment objects. The event-generating objects shall have the Event_Enable property configured to transmit notification messages for all event transitions. The event-generating objects shall be configured to be in a NORMAL event state at the start of the test. One Notification Class object, *N1*~~N1~~, shall be configured with Issue Confirmed Notifications equal to TRUE. The other Notification Class object, *N2*~~N2~~, shall be configured with Issue Confirmed Notifications equal to FALSE. The Valid Days parameter shall be configured so that at least one day of the week has a value of TRUE. The Transitions parameter shall be configured for the recipient to receive notifications for all event transitions. The local date and time shall be configured to be within the window defined by From Time and To Time on one of the ValidDays.

In the test description below "*X1*~~X1~~" and "*X2*~~X2~~" are used to designate the ~~event-triggering property~~*pMonitoredValue algorithm parameter* linked to Notification objects " *N1*~~N1~~" and " *N2*~~N2~~", respectively.

Test Steps:

1. VERIFY (the event-generating object linked to *N1*~~N1~~), ~~Event_State~~*pCurrentState* = NORMAL
2. VERIFY (the event-generating object linked to *N2*~~N2~~), ~~Event_State~~*pCurrentState* = NORMAL
3. WAIT (~~Time_Delay~~*pTimeDelay* + **Notification Fail Time**)
4. IF (X_1 is writable) THEN
   WRITE *X1*~~X1~~ = (a value that is OFFNORMAL)
   ELSE
   MAKE (*X1*~~X1~~ a value that is OFFNORMAL)
5. WAIT (~~Time_Delay~~*pTimeDelay*)
6. BEFORE **Notification Fail Time**
   RECEIVE ConfirmedEventNotification-Request,
   'Process Identifier' = (any valid process ID),
   'Initiating Device Identifier' = IUT,
   'Event Object Identifier' = (the event-generating object linked to *N1*~~N1~~),
   'Time Stamp' = (~~the current local time~~*any valid time stamp*),
   'Notification Class' = (the class corresponding to the object being tested),
   'Priority' = (the value configured to correspond to a TO-OFFNORMAL transition),

	'Event Type' =	(any valid event type),
	'Message Text' =	*(optional, any valid message text),*
	'Notify Type' =	EVENT | ALARM,
	'AckRequired' =	TRUE | FALSE,
	'From State' =	NORMAL,
	'To State' =	OFFNORMAL,
	'Event Values' =	(values appropriate to the event type)

7. IF (X_2 is writable) THEN
   WRITE *X2*~~X2~~ = (a value that is OFFNORMAL)
   ELSE
   MAKE (*X2*~~X2~~ a value that is OFFNORMAL)
8. WAIT (~~Time_Delay~~*pTimeDelay*)
9. BEFORE **Notification Fail Time**
   RECEIVE UnconfirmedEventNotification-Request,

	'Process Identifier' =	(any valid process ID),
	'Initiating Device Identifier' =	IUT,
	'Event Object Identifier' =	(the event-generating object linked to *N2*~~N2~~),
	'Time Stamp' =	(~~the current local time~~*any valid time stamp*),
	'Notification Class' =	(the class corresponding to the object being tested),
	'Priority' =	(the value configured to correspond to a TO-OFFNORMAL transition),
	'Event Type' =	(any valid event type),
	'Message Text' =	*(optional, any valid message text),*
	'Notify Type' =	EVENT | ALARM,
	'AckRequired' =	TRUE | FALSE,
	'From State' =	NORMAL,
	'To State' =	OFFNORMAL,
	'Event Values' =	(values appropriate to the event type)

Notes to Tester: If the Recipient_List is writable, and the Issue Confirmed Notifications can be changed, then this test can be performed using only one Notification Class object by writing to the Recipient_List in order to change between confirmed and unconfirmed notifications. ~~The 'Message Text' parameter is omitted in the test description because it is optional. The IUT may include this parameter in the notification messages.~~

[Change **Clause 7.3.2.21.3.4**, p.94]

7.3.2.21.3.4 Transitions Test

~~Dependencies: ConfirmedEventNotification-Service-Initiation-Tests, 8.4; UnconfirmedEventNotification-Service-Initiation Tests, 8.5; ReadProperty-Service-Execution-Tests, 9.18; WriteProperty-Service-Execution-Tests, 9.22.~~

~~BACnet-Reference-Clause: 12.21.8.~~

Purpose: To verify that notification messages are transmitted only if the bit in the Transitions parameter corresponding to the event transition is set.

Test Concept: The IUT is configured such that the Transitions parameter indicates that some event transitions are to trigger an event notification and some are not. Each event transition is triggered and the IUT is monitored to verify that notification messages are transmitted only for those transitions for which the Transitions parameter has a value of TRUE. *For devices that implement a read-only Recipient_List property for all instances of Notification Class objects, and are exclusively configured for all transitions (all bits in Transitions set to TRUE), this test shall be omitted. For devices that implement a writeable Notification Class Recipient_List property for all instances of Notification Class objects, and exclusively accept all transitions as the only permitted configuration, this test shall be omitted.*

Configuration Requirements: The IUT shall be configured with one or more instance of the Notification Class object and at least one event-generating object that is linked to the Notification Class object. ~~The event-generating object may be any object that supports intrinsic reporting or it may be an Event Enrollment object.~~ The event-generating object shall have the

Event_Enable property configured to transmit notification messages for all event transitions. The event-generating object shall be configured to be in a NORMAL event state at the start of the test. The Notification Class object shall be configured with a single recipient in the Recipient_List. The Transitions parameter shall be configured with a value of TRUE for either the TO-OFFNORMAL transition or the TO-NORMAL transition, and the other event transition shall have a value of FALSE. The local time shall be configured such that it represents one of the valid days in the window specified by From Time and To Time.

~~In the test description below, "X" is used to designate the event-triggering property.~~
Test Steps:

1. VERIFY ~~Event_State~~*pCurrentState* = NORMAL
2. WAIT (~~Time_Delay~~*pTimeDelay* + **Notification Fail Time**)
3. IF (~~X~~*pMonitoredValue* is writable) THEN
   WRITE ~~X~~*pMonitoredValue* = (a value that is OFFNORMAL)
   ELSE
   MAKE (~~X~~*pMonitoredValue* have a value that is OFFNORMAL)
4. WAIT (~~Time_Delay~~*pTimeDelay*)
5. BEFORE **Notification Fail Time**
   IF (the Transitions bit corresponding to the TO-OFFNORMAL transition is TRUE) THEN
   RECEIVE ConfirmedEventNotification-Request,
   'Process Identifier' = (any valid process ID),
   'Initiating Device Identifier' = IUT,
   'Event Object Identifier' = (the event-generating object configured for this test),
   'Time Stamp' = (~~the current local time~~*any valid time stamp*),
   'Notification Class' = (the class corresponding to the object being tested),
   'Priority' = (the value configured to correspond to a TO-OFFNORMAL transition),
   'Event Type' = (any valid event type),
   'Message Text' = *(optional, any valid message text),*
   'Notify Type' = EVENT | ALARM,
   'AckRequired' = TRUE | FALSE,
   'From State' = NORMAL,
   'To State' = OFFNORMAL,
   'Event Values' = (values appropriate to the event type)
   TRANSMIT BACnet-SimpleACK-PDU
   ELSE
   CHECK (verify that the IUT did not transmit an event notification message)
6. VERIFY ~~Event_State~~*pCurrentState* = OFFNORMAL
7. IF (~~X~~*pMonitoredValue* is writable) THEN
   WRITE ~~X~~*pMonitoredValue* = (a value that is NORMAL)
   ELSE
   MAKE (~~X~~*pMonitoredValue* have a value that is NORMAL)
8. WAIT (~~Time_Delay~~*pTimeDelayNormal*)
9. BEFORE **Notification Fail Time**
   IF (the Transitions bit corresponding to the TO-NORMAL transition is TRUE) THEN
   RECEIVE ConfirmedEventNotification-Request,
   'Process Identifier' = (any valid process ID),
   'Initiating Device Identifier' = IUT,
   'Event Object Identifier' = (the event-generating object configured for this test),
   'Time Stamp' = (~~the current local time~~*any valid time stamp*),
   'Notification Class' = (the class corresponding to the object being tested),
   'Priority' = (the value configured to correspond to a TO-NORMAL transition),
   'Event Type' = (any valid event type),
   'Message Text' = *(optional, any valid message text),*
   'Notify Type' = EVENT | ALARM,
   'AckRequired' = TRUE | FALSE,
   'From State' = OFFNORMAL,

'To State' = NORMAL,
'Event Values' = (values appropriate to the event type)

ELSE
CHECK (verify that the IUT did not transmit an event notification message)

10. VERIFY ~~Event_State~~*pCurrentState* = NORMAL
11. IF (the event-triggering object can be placed into a fault condition) THEN {
MAKE (~~the event triggering~~ *a condition exist that will cause the* object ~~change to a fault condition~~ *to generate a TO_FAULT transition*)
BEFORE **Notification Fail Time**
IF (the Transitions bit corresponding to the TO-FAULT transition is TRUE) THEN
RECEIVE ConfirmedEventNotification-Request,
'Process Identifier' = (any valid process ID),
'Initiating Device Identifier' = IUT,
'Event Object Identifier' = (the event-generating object configured for this test),
'Time Stamp' = (~~the current local time~~*any valid time stamp*),
'Notification Class' = (the class corresponding to the object being tested),
'Priority' = (the value configured to correspond to a TO-FAULT transition),
'Event Type' = *(IF (Protocol_Revision < 13) THEN*
(any valid event type)
ELSE
CHANGE_OF_RELIABILITY),
'Message Text' = *(optional, any valid message text),*
'Notify Type' = EVENT | ALARM,
'AckRequired' = TRUE | FALSE,
'From State' = NORMAL,
'To State' = FAULT,
'Event Values' = (values appropriate to the event type)
ELSE
CHECK (verify that the IUT did not transmit an event notification message)
VERIFY ~~Event_State~~*pCurrentState* = FAULT
}

Notes to Tester: The UnconfirmedEventNotification service may be substituted for the ConfirmedEventNotification service*, in which case the TD shall skip all of the steps in which a BACnet-SimpleACK-PDU is sent*. ~~The 'Message Text' parameter is omitted in the test description because it is optional. The IUT may include this parameter in the notification messages.~~

[Change **Clause 7.3.2.21.3.5**, p.96]

7.3.2.21.3.5 Recipient_List Property Supports Device Identifier Recipients Test

Purpose: To verify that the Recipient_List property of the Notification Class object supports the device form of the Recipient component*,* and that the IUT is able to associate a MAC address with the Device Identifier. The intent is to ensure that the IUT is able to locate the specified alarm recipient and send notification to the specified recipient. This test shall be run if the IUT's Notification Class object's Recipient_List property supports the BACnet object identifier form of BACnetRecipient.

Test Concept: The tester shall select a single ~~event generating~~*event generating* object*,* E*,* in the IUT that references Notification Class object*,* N. The tester shall add an entry into the Recipient_List of the associated Notification Class object that specifies a Device Identifier, D, for a device that the IUT is not already aware of. -The TD, acting as device D, shall be located on a different network than the IUT to ensure that the IUT is capable of binding to recipients located on any network. *For devices that implement a read-only Recipient_List property for all instances of Notification Class objects, and there is an address form of the Recipient component in all instances, this test shall be omitted. D1 is either the pTimeDelay, or pTimeDelayNormal parameter, or 0 (for transitions to and from FAULT state) depending on the event transition.*

Configuration Requirements: The TD shall be configured so that it does not execute WhoHas.

Test Steps:

1. WRITE N.RecipientList = ({all days, all times, D, any process ID *(PID)*, FALSE, all transitions})
2. MAKE (~~the event-generating object, E, transition~~*a condition exist that will cause E to generate an event transition*)
3. *WAIT D1*
~~3~~*4*. BEFORE ~~Notification Fail Time~~*(**Notification Fail Time** plus the amount of time the IUT takes to perform device discovery)*

RECEIVE UnconfirmedEventNotification-Request,

'Process Identifier' =	*PID*~~(the valid process ID from step 1)~~,
'Initiating Device Identifier' =	IUT,
'Event Object Identifier' =	E,
'Time Stamp' =	(any valid time stamp),
'Notification Class' =	(N's instance),
'Priority' =	(any valid priority),
'Event Type' =	(any valid event type),
'Message Text' =	*(optional, any valid message text),*
'Notify Type' =	ALARM | EVENT,
'AckRequired' =	TRUE | FALSE,
'From State' =	(any valid event state),
'To State' =	(any valid event state),
'Event Values' =	(values appropriate to the event type)

Notes to Tester: The IUT is expected to initiate one or more range-restricted WhoIs requests after the modification of the Recipient_List but before the sending of the notification. The IUT might also need to perform other network discovery operations. Given that there are multiple approaches to the use of WhoIs for device discovery, the test only focuses on the IUT's ability to find device D and not on the specifics or timing of the WhoIs requests.

[Change **Clause 7.3.2.21.3.6**, p.96]

7.3.2.21.3.6 Recipient_List Property Supports Network Address Recipients

Purpose: To verify that the Recipient_List property of the Notification Class object supports the address form of the Recipient component. The intent is to ensure that the IUT is able to send notifications to the specified recipient.

Test Concept: The tester shall select a single event-generating object, E, in the IUT that references Notification Class object, N. The tester shall add an entry into the Recipient_List of the associated Notification Class object that specifies a BACnetAddress, A, where A is a unicast or is a local, remote, or global broadcast address. *For devices that implement a read-only Recipient_List property for all instances of Notification Class objects, and there is a Device Identifier form of the Recipient component in all instances, this test shall be skipped. D1 is either the pTimeDelay, or pTimeDelayNormal parameter, or 0 (for transitions to and from FAULT state) depending on the event transition.*

Test Steps:

1. WRITE N.RecipientList = (-{all days, all times, A, any process ID *(PID)*, FALSE, all transitions}-)
2. MAKE (~~the event-generating object, E, transition~~*a condition exist that will cause E to generate an event transition*)
3. *WAIT D1*
~~3~~*4*. BEFORE ~~Notification Fail Time~~***Notification Fail Time***

RECEIVE
DESTINATION = A,
UnconfirmedEventNotification-Request,

'Process Identifier' =	~~(the valid process ID from step 1)~~*PID*,
'Initiating Device Identifier' =	IUT,
'Event Object Identifier' =	E,
'Time Stamp' =	(~~the current local time~~*any valid time stamp*),

'Notification Class' =	(N's instance),
'Priority' =	(any valid priority),
'Event Type' =	(any valid event type),
'Message Text' =	*(optional, any valid message text),*
'Notify Type' =	ALARM | EVENT,
'AckRequired' =	TRUE | FALSE,
'From State' =	(any valid event state),
'To State' =	(any valid event state),
'Event Values' =	(values appropriate to the event type)

[Add new **Clause 7.3.2.21.3.7**, p. 97]

7.3.2.21.3.7 Recipient_List non-volatility test

Purpose: This test case verifies that a Notification Class object Recipient_List is maintained through a power failure and device restart.

Test Concept: Write the Recipient_List of a Notification Class object and restart the IUT device by issuing a ReinitializeDevice – WARMSTART service request and by temporarily removing power. When the device has resumed operation after each restart, verify that the Recipient_List contains the values that were written. This test is only applied to IUT devices that have writable Notification Class object Recipient_List properties. If the device only accepts Recipient_List values that include Valid Days = (1, 1, 1, 1, 1, 1, 1), From Time = 00:00:00.00, To Time = 23:59:59.99, and Transitions = (True, True, True), then those values shall be used in this test. If the IUT accepts Recipient_List sizes greater than one, then at least two different BACnetDestination values shall be written in the list. If the device does not support the ReinitializeDevice – WARMSTART service, then only the removal of power will be tested.

Test Steps:

1. WRITE Recipient_List = (RL: any valid value)
2. IF (ReinitializeDevice – WARMSTART execution is supported, i.e. BIBB DM-RD-B) THEN {
   TRANSMIT ReinitializeDevice-Request
   Reinitialized State of Device = WARMSTART
   Password = (any valid password)
   RECEIVE BACnet-SimpleACK-PDU
   CHECK (Did the IUT perform a WARMSTART reboot?)
   VERIFY Recipient_List = RL
   }
   ELSE {
   MAKE (power cycle the IUT to make it reinitialize)
   VERIFY Recipient_List = RL
   }

[Add new Clause **7.3.2.21.3.8**, p. 97]

7.3.2.21.3.8 Read-only Recipient_List with internal Notification Forwarder objects

Purpose: This test case verifies that a read-only Notification Class object Recipient_List is configured with only content designed for internal Notification Forwarder objects.

Test Concept: This test is only applied to IUT devices that have read-only Notification Class object Recipient_List properties and are capable of containing a Notification Forwarder object. The Notification Class Recipient_List is read and checked to ensure all entries in the Recipient_List refer to the local device.

Test Steps:

1. READ RL = Recipient_List
2. CHECK (All Recipients in RL are equal to IUT)

[Add new **Clause 7.3.2.21.3.9**, p. 97]

7.3.2.21.3.9 Read-only Recipient_List for external Notification Forwarder Objects

Purpose: This test case verifies that a read-only Notification Class object Recipient_List is configured with content designed for external Notification Forwarder objects.

Test Concept: Read the Recipient_List of the Notification Class objects and check that the length is 1, the Recipient is local broadcast, Valid Days are all days, From Time and To Time are the entire day, Process Identifier is 0, Issue Confirmed Notifications parameter is False and Transitions is set to all transitions. This test is only applied to IUT devices that have read-only Notification Class object Recipient_List properties, and which do not contain internal Notification Forwarder objects.

Test Steps:

1. VERIFY Recipient_List = {(1, 1, 1, 1, 1, 1, 1) -- Valid Days
   00:00:00.0 -- From Time
   23:59:59.99 -- To Time
   (BACnetAddress: network-number = 0, zero length mac-address)
   0 -- Process Identifier
   False -- Issue Confirmed Notifications
   (True, True, True) -- Transitions
   }

[Add new **Clause 7.3.2.21.3.10**, p. 97]

7.3.2.21.3.10 Read-only Recipient_List Without Notification Forwarder Test

Purpose: This test case verifies that each read-only Recipient_List for all instances of Notification Class objects, in a device that cannot contain a Notification Forwarder object, are correctly configured. This test is only applied to devices that have a read-only Recipient_List property for all instances of Notification Class objects in a device that cannot contain a Notification Forwarder object.

Test Concept: Read the Recipient_List of the Notification Class objects and check that the length of each object is 1, the Recipient is local broadcast, Valid Days are all days, From Time and To Time are the entire day, Process Identifier is 0, Issue Confirmed Notification is False and Transitions is set to all transitions. This test is only applied to IUT devices that have read-only Notification Class object Recipient_List properties and are not capable of containing a Notification Forwarder object.

Test Steps:

1. REPEAT X = (each Notification Class object instance in the IUT) DO
   {VERIFY (X), Recipient_List = {--list size of one
   {(1, 1, 1, 1, 1, 1, 1) -- Valid Days
   00:00:00.0 -- From Time
   23:59:59.99 -- To Time
   (BACnetAddress: network-number = 0, zero length mac-address)--Recipient
   0 -- Process Identifier
   False -- Issue Confirmed Notifications

```
                                        (True, True, True)      -- Transitions
                                        }                       -- end of list entry
                                    }                           -- end of list
    }
```

[Change **Clause 7.3.2.24.10**, p. 137]

7.3.2.24.10 Notification_Threshold Test

~~Dependencies: ReadProperty Service Execution Tests, 9.18; WriteProperty Service Execution Tests, 9.22.~~

Purpose: To verify that the Notification_Threshold property reflects the number of records collected since a previous notification, or since logging started, that causes a Buffer_Ready notification to be sent.

Test Concept: The logging object*, LO1,* is configured to acquire data by whatever means. ~~-~~Record_Count is set to zero. Collection of data proceeds until a notification is seen, and the value of Record_Count is checked. Collection continues until the second notification, when Record_Count verified again.

Configuration Requirements: Start_Time, if present, shall be configured with a date and time preceding the beginning of the test. Stop_Time, if present shall be configured with the latest possible date and time, in order that it occur after the end of the test. ~~-~~Enable shall be set to FALSE. Notification_Threshold shall be set to a non-zero value.

Test Steps:

1. WRITE Record_Count = 0
2. WAIT ~~-~~**Internal Processing Fail Time**
3. READ X = Total_Record_Count
4. WRITE Enable = TRUE
5. MAKE (~~the logging object~~ *LO1* collect number of records specified by ~~Notification_Threshold~~*pThreshold*)
6. RECEIVE ConfirmedEventNotification-Request,

'Process Identifier' =	(any valid process ID),
'Initiating Device Identifier' =	IUT,
'Event Object Identifier' =	~~(the object being tested)~~*LO1*,
'Time Stamp' =	(~~any appropriate BACnetTimeStamp value~~*any valid time stamp*),
'Notification Class' =	(the configured notification class),
'Priority' =	(the value configured to correspond to a TO-NORMAL transition),
'Event Type' =	BUFFER_READY,
'Message Text' =	*(optional, any valid message text),*
'Notify Type' =	EVENT | ALARM,
'AckRequired' =	TRUE | FALSE,
'From State' =	NORMAL,
'To State' =	NORMAL,
'Event Values' =	(~~'Buffer Property': the logging object~~*pLogBuffer*~~)~~,
	(~~'Previous Notification'~~*pPreviousCount*: valid value < X),
	(~~'Current Notification'~~*pMonitoredValue*: any value Y_1 where
	Y_1 >= X and Y_1 <= X + ~~Notification_Threshold~~ *pThreshold*)

7. TRANSMIT BACnet-SimpleACK-PDU
8. VERIFY Total_Record_Count >= Y_1
9. MAKE (~~the logging object~~ *LO1* collect number of records specified by ~~Notification_Threshold~~ *pThreshold*)
10. RECEIVE ConfirmedEventNotification-Request,

'Process Identifier' =	(any valid process ID),
'Initiating Device Identifier' =	IUT,
'Event Object Identifier' =	(the object being tested),
'Time Stamp' =	(~~any appropriate BACnetTimeStamp value~~*any valid time stamp*),
'Notification Class' =	(the configured notification class),

'Priority' =	(the value configured to correspond to a TO-NORMAL transition),
'Event Type' =	BUFFER_READY,
'Message Text' =	*(optional, any valid message text),*
'Notify Type' =	EVENT | ALARM,
'AckRequired' =	TRUE | FALSE,
'From State' =	NORMAL,
'To State' =	NORMAL,
'Event Values' =	~~('Buffer Property': the logging object)~~*pLogBuffer*,
	(~~'Previous Notification'~~*pPreviousCount*: Y_1),
	(~~'Current Notification'~~*pMonitoredValue*: Y_1 + ~~Notification_Threshold~~*pThreshold*)

11. TRANSMIT BACnet-SimpleACK-PDU
12. VERIFY Total_Record_Count >= Y_1 + ~~Notification_Threshold~~*pThreshold*
13. WRITE Enable = FALSE

[Change **Clause 7.3.2.24.17**, p. 143]

7.3.2.24.17 Last_Notify_Record Test

~~Dependencies: ReadProperty Service Execution Tests, 9.15; WriteProperty Service Execution Tests, 9.18.~~

~~BACnet Reference Clauses: 12.25.19 and 12.25.26~~

Purpose: To verify that the Last_Notify_Record property reflects the values sent in the most recent notification.

Test Concept: The log buffer is cleared. The Log is allowed to collect records until it issues a BUFFER_READY notification. The value of the Last_Notify_Record property is checked.

~~Test~~ Configuration *Requirements*: The log is configured to send BUFFER_READY notifications to the TD.

Test Steps:

1. WRITE Record_Count = 0
2. READ prev = Last_Notify_Record
3. WRITE Enable = TRUE
4. MAKE (Log object collect number of records specified by ~~Notification_Threshold~~*pThreshold*)
5. RECEIVE ConfirmedEventNotification-Request,

~~'~~'Process Identifier~~'~~' =	(the configured process identifier)
~~'~~'Initiating Device Identifier~~'~~' =	IUT,
~~'~~'Event Object Identifier~~'~~' =	(Log object being tested),
~~'~~'Time Stamp~~'~~' =	(any appropriate BACnetTimeStamp value),
~~'~~'Notification Class~~'~~' =	(configured notification class),
~~'~~'Priority~~'~~' =	(value configured to correspond to a TO-NORMAL),
~~'~~'Event Type~~'~~' =	BUFFER_READY,
'Message Text' =	*(optional, any valid message text),*
~~'~~'Notify Type~~'~~' =	EVENT | ALARM,
~~'~~'AckRequired~~'~~' =	TRUE | FALSE,
~~'~~'FromState~~'~~' =	NORMAL,
~~'~~'To State~~'~~' =	NORMAL,
~~'~~'Event Values~~'~~' =	~~((the log object), Log_Buffer~~*pLogBuffer*,
	~~previous notification = prev~~*(pPreviousCount*: *prev)*,
	~~current notification, C1~~*(pMonitoredValue: C1*)

6. TRANSMIT BACnet-SimpleACK-PDU
7. WRITE Enable = FALSE
8. VERIFY Last_Notify_Record = C1

[Change **Clause 7.3.2.24.18**, p. 144]

7.3.2.24.18 Records_Since_Notification Test

~~Dependencies: ReadProperty Service Execution Tests, 9.18; WriteProperty Service Execution Tests, 9.22.~~

~~BACnet Reference Clauses: 12.25.18, 12.25.19, 12.25.22~~

Purpose: To verify that the Records_Since_Notification property reflects the number of records recorded by the log, ~~that~~ *which* have not yet been reported via a BUFFER_READY notification.

Test Concept: The log buffer is cleared. The log is allowed to collect records until it issues a BUFFER_READY notification and is halted before a second notification is generated. -The value of the Records_Since_Notification property is checked.

~~Test~~ Configuration *Requirements*: The ~~Trend Log~~*logging object, LO1,* is configured to send BUFFER_READY notifications to the TD.

Test Steps:

1. WRITE Enable = TRUE
2. WRITE Record_Count = 0
3. MAKE (~~Log object~~*LO1* collect a sufficient number of records in order to trigger a notification)
4. RECEIVE ConfirmedEventNotification-Request,
   - ~~'~~Process Identifier~~'~~ = (the configured process identifier)
   - ~~'~~Initiating Device Identifier~~'~~ = IUT,
   - ~~'~~Event Object Identifier~~'~~ = ~~(Log object being tested)~~*LO1*,
   - ~~'~~Time Stamp~~'~~ = (Any appropriate BACnetTimeStamp value),
   - ~~'~~Notification Class~~'~~ = (configured notification class),
   - ~~'~~Priority~~'~~ = (value configured to correspond to a TO-NORMAL),
   - ~~'~~Event Type~~'~~ = BUFFER_READY,
   - *'Message Text'* = *(optional, any valid message text),*
   - ~~'~~Notify Type~~'~~ = EVENT | ALARM,
   - ~~'~~AckRequired~~'~~ = TRUE | FALSE,
   - ~~'~~FromState~~'~~ = NORMAL,
   - ~~'~~To State~~'~~ = NORMAL,
   - ~~'~~Event Values~~'~~ = ~~((the log object), Log_Buffer)~~*pLogBuffer*, ~~previous notification~~*pPreviousCount*, ~~current notification, C1~~*(pMonitoredValue: C1)*
5. TRANSMIT BACnet-SimpleACK-PDU
6. MAKE (~~Log object~~*LO1* collect N records, such that N < ~~Notification_Threshold~~*pThreshold*-1)
7. WRITE Enable = FALSE
8. READ T1 = Total_Record_Count
9. VERIFY Records_Since_Notification = T1 - C1

7.3.2.25 Event Log Tests

...

[Change **Clause 7.3.2.25.1**, p. 145]

7.3.2.25.1 Internal Logging of Notifications

Purpose: To verify the IUT correctly collects and represents the Notifications, which it initiates.

Test Concept: Make the IUT generate two event notification messages, which the IUT logs. Use ReadRange to retrieve them from an Event Log and compare the two representations.

Configuration Requirements: The tester shall choose two events which are configured to be sent to the TD and to be placed into one of the IUT's Event Log~~s~~, LO1.

Test Steps:

1. WRITE Enable = TRUE
2. MAKE (~~IUT generate an EventNotification~~*a condition exist that will cause the device to generate an event transition*)
3. *WAIT D1*
~~3~~4. RECEIVE ConfirmedEventNotification-Request,
'Process Identifier' = (any valid process identifier),
'Initiating Device Identifier' = IUT,
'Event Object Identifier' = (any valid object),
'Time Stamp' = (T1, any valid timestamp),
'Notification Class' = (any valid notification class),
'Priority' = (any valid priority),
'Event Type' = (any standard event type),
'Message Text' = ~~(any character string)~~*(optional, any valid message text)*,
'Notify Type' = ALARM | EVENT,
'AckRequired' = TRUE | FALSE,
'From State' = (state S1, any valid state for this event type),
'To State' = (state S2, any valid state for this event type that can follow S1),
'Event Values' = (any values appropriate to the event type)
4~~5~~. TRANSMIT BACnet-SimpleACK-PDU
~~5~~6. MAKE (~~IUT generate an EventNotification~~*a condition exist that will cause the device to generate an event transition*)
7. *WAIT D2*
~~6~~8. RECEIVE ConfirmedEventNotification-Request,
'Process Identifier' = (any valid process identifier),
'Initiating Device Identifier' = IUT,
'Event Object Identifier' = (any valid object),
'Time Stamp' = (T2, any valid timestamp),
'Notification Class' = (any valid notification class),
'Priority' = (any valid priority),
'Event Type' = (any standard event type),
'Message Text' = ~~(any character string)~~*(optional, any valid message text)*,
'Notify Type' = ALARM | EVENT,
'AckRequired' = TRUE | FALSE,
'From State' = (state S3, any valid state for this event type),
'To State' = (state S4, any valid state for this event type that can follow S3),
'Event Values' = (any values appropriate to the event type)
~~7~~9. TRANSMIT BACnet-SimpleACK-PDU
~~8~~10.READ RC = LO1, Record_Count
~~9~~11.TRANSMIT ReadRange-Request,
'Object Identifier' = LO1,
'Property Identifier' = Log_Buffer,
'~~'~~Reference Index'~~'~~ = RC,
'~~'~~Count'~~'~~ = -2
~~10~~12. RECEIVE ReadRange-ACK,
'Object Identifier' = LO1,
'Property Identifier' = Log_Buffer,
'Result Flags' = {FALSE, ?, TRUE},
'Item Count' ~~=~~ 2,

'Item Data' = (logged data that matches the information received in steps 3 and 6, except that Process_Identifier may be any value and is not required to match)

~~11~~*13*. CHECK (T2 > T1, and that the notifications were logged in order)

Notes to Tester: When the UnconfirmedEventNotification service is used instead of the ConfirmedEventNotification service, the TD shall skip the steps in which a *BACnet*-SimpleACK-PDU is sent.

[Change **Clause 7.3.2.26**, p. 146]

7.3.2.~~26.~~*25.2* Remote Logging of Notifications

Purpose: To verify that the IUT correctly collects and represents the Notifications which it receives.

Test Concept: Make TD send multiple event notification messages. Use ReadRange to retrieve the events from an Event Log, or ~~perhaps~~ from multiple Event Logs in the IUT, and compare the two representations.

Configuration Requirements: LO1 is an Event Log object in IUT, which logs the event types *that*~~which~~ are sent. Stop_When_Full in LO1 shall be FALSE or absent.

Test Steps:

1. WRITE Enable = TRUE
2. TRANSMIT ConfirmedEventNotification-Request,
   'Process Identifier' = (any valid process identifier),
   'Initiating Device Identifier' = TD,
   'Event Object Identifier' = (any valid object identifier),
   'Time Stamp' = (T1, any valid timestamp),
   'Notification Class' = (any valid notification class),
   'Priority' = (any valid priority),
   'Event Type' = (any standard event type),
   'Message Text' = (any character string),
   'Notify Type' = ALARM | EVENT,
   'AckRequired' = TRUE | FALSE,
   'From State' = (state S1, any valid state for this event type),
   'To State' = (state S2, any valid state for this event type that can follow S1),
   'Event Values' = (any values appropriate to the event type)
3. RECEIVE BACnet-SimpleACK-PDU
4. TRANSMIT ConfirmedEventNotification-Request,
   'Process Identifier' = (any valid process identifier),
   'Initiating Device Identifier' = IUT,
   'Event Object Identifier' = (any valid object identifier),
   'Time Stamp' = (T2, any valid timestamp),
   'Notification Class' = (any valid notification class),
   'Priority' = (any valid priority),
   'Event Type' = (any standard event type),
   'Message Text' = (any character string),
   'Notify Type' = ALARM | EVENT,
   'AckRequired' = TRUE | FALSE,
   'From State' = (state S3, any valid state for this event type),
   'To State' = (state S4, any valid state for this event type that can follow S3),
   'Event Values' = (any values appropriate to the event type)
5. RECEIVE BACnet-SimpleACK-PDU
6. READ RC = LO1, Record_Count
7. TRANSMIT ReadRange-Request,
   'Object Identifier' = LO1,

'Property Identifier' = Log_Buffer,
'~~"~~Reference Index'~~"~~ = RC,
'~~"~~Count'~~"~~ = -2

8. RECEIVE ReadRange-ACK,
'Object Identifier' = LO1,
'Property Identifier' = Log_Buffer,
'Result Flags' = {FALSE, ?, TRUE},
'Item Count' = 2,
'Item Data' = (logged data that matches the information received in steps 2 and 4, except that Process_Identifier can be any value and is not required to match)
9. CHECK (that the events were logged in the order in which they were received)

Notes to Tester: When the UnconfirmedEventNotification service is used instead of the ConfirmedEventNotification service, the test shall skip the steps in which a *BACnet*-SimpleACK-PDU is expected.

[Change **Clause 7.3.2.27**, p. 148]

7.3.2.~~27.~~*25.3* Internal Logging of ACK_NOTIFICATIONs

Purpose: To verify the IUT correctly collects and represents an ACK_NOTIFICATION which it initiates.

Test Concept: Make the IUT generate an ACK_NOTIFICATION message. Use ReadRange to retrieve that same event from an Event Log and compare the two representations. If the IUT does not support logging of the ACK_NOTIFICATIONs which it initiates, this test shall be skipped.

Configuration Requirements: ~~-~~O1 is an event initiating object in the IUT, which is configured to send event notifications to *the* TD. LO1 is an Event Log object in *the* IUT*,* which logs ACK_NOTIFICATIONs.

Test Steps:

1. WRITE Enable = TRUE
2. READ RC = LO1, Record_Count
3. MAKE (~~the IUT generate a notification~~ *a condition exist that will cause the object to generate an event transition*)
4. WAIT D1
~~4~~*5*. RECEIVE ConfirmedEventNotification-Request,
'Process Identifier' = (PI1, any valid process identifier),
'Initiating Device Identifier' = IUT,
'Event Object Identifier' = O1,
'Time Stamp' = (T1, any valid timestamp),
'Notification Class' = (N1, any valid notification class),
'Priority' = (P1, any valid priority),
'Event Type' = (ET1, any standard event type),
'Message Text' = ~~(any character string)~~*(optional, any valid message text)*,
'Notify Type' = ALARM | EVENT,
'AckRequired' = TRUE | FALSE,
'From State' = (S1, any valid state for this event type),
'To State' = (S2, any valid state for this event type),
'Event Values' = (any values appropriate to the event type)
~~5~~*6*. TRANSMIT BACnet-SimpleACK-PDU
~~6~~*7*. TRANSMIT AcknowledgeAlarm-Request,
'Acknowledging Process Identifier' = (any valid value),
'Event Object Identifier' = O1,
'Event State Acknowledged' = S2,
'Time Stamp' = T1,
'*Acknowledgement Source*' = *(any valid value),*

'Time of Acknowledgment' = (the current time)

~~7~~*8*. RECEIVE BACnet-SimpleACK-PDU

~~8~~*9*. BEFORE **Notification Fail Time**

RECEIVE ConfirmedEventNotification-Request,
'Process Identifier' = PI1,
'Initiating Device Identifier' = IUT,
'Event Object Identifier' = O1,
'Time Stamp' = T2, any valid timestamp > T1),
'Notification Class' = N1,
'Priority' = P1,
'Event Type' = ET1,
'Message Text' = ~~any character string)~~*(optional, any valid message text)*,
'Notify Type' = ACK_NOTIFICATION,
'From State' = S1

~~9~~*10*.TRANSMIT BACnet-SimpleACK-PDU

~~10~~*11*. TRANSMIT ReadRange-Request,
'Object Identifier' = LO1,
'Property Identifier' = Log_Buffer,
~~'~~'Reference Index'~~'~~ = RC,
~~'~~'Count'~~'~~ = -1

~~11~~12. RECEIVE ReadRange-ACK,
'Object Identifier' = LO1,
'Property Identifier' = Log_Buffer,
'Result Flags' = {FALSE, ?, TRUE},
'Item Count' = 1,
'Item Data' = (logged data that matches the information received in step 4, except that Process_Identifier can be any value and is not required to match)

Notes to Tester: When the UnconfirmedEventNotification service is used instead of the ConfirmedEventNotification service, the TD shall skip the steps in which *BACnet*-SimpleACK-PDUs are sent in response to ConfirmedEventNotifications.

[Change **Clause 7.3.2.28**, p. 149]

7.3.2.~~28.~~*25.4* Remote Logging of ACK_NOTIFICATIONs

Purpose: To verify that the IUT correctly collects and represents ACK_NOTIFICATIONs which it receives.

Test Concept: Send an ACK_NOTIFICATION to the IUT. Use ReadRange to retrieve that same event from an Event Log, and compare the two representations.

Configuration Requirements: LO1 is an Event Log object in *the* IUT, which logs ACK_NOTIFICATIONs. Stop_When_Full in LO1 shall be FALSE or absent.

Test Steps:

1. WRITE Enable = TRUE
2. TRANSMIT ConfirmedEventNotification-Request,
'Process Identifier' = (any valid process identifier),
'Initiating Device Identifier' = IUT,
'Event Object Identifier' = (any valid object identifier),
'Time Stamp' = (T1, any valid timestamp),
'Notification Class' = (any valid notification class),
'Priority' = (any valid priority),
'Event Type' = (any standard event type),

	'Message Text' =	(any character string),
	'Notify Type' =	ACK_NOTIFICATION,
	'From State' =	(state S1, any valid state for this event type)
3.	RECEIVE BACnet-SimpleACK-PDU	
4.	READ RC = LO1, Record_Count	
5.	TRANSMIT ReadRange-Request,	
	'Object Identifier' =	LO1,
	'Property Identifier' =	Log_Buffer,
	"Reference Index" =	RC,
	"Count" =	-1
6.	RECEIVE ReadRange-ACK,	
	'Object Identifier' =	LO1,
	'Property Identifier' =	Log_Buffer,
	'Result Flags' =	{FALSE, ?, TRUE},
	'Item Count' =	1,
	'Item Data' =	(logged data that matches the information received in step 2,
	—	except that Process_Identifier can be any value and is not required to match)

Notes to Tester: When the UnconfirmedEventNotification service is used instead of the ConfirmedEventNotification service, the test shall skip the step in which a *BACnet*-SimpleACK-PDU is expected.

[Add new **Clause 7.3.2.30**, p. 152]
[Reviewer Note: Addendum 135-2010*af*-31 at Protocol_Revision 13 added language and a new standard object type: Alert Enrollment. This proposes tests for the specified behavior.]

7.3.2.30 Alert Enrollment Tests

7.3.2.30.1 Alert Enrollment Reports The Source Object

Purpose: To verify that the Alert Enrollment object's notifications provide an object identifier as the first parameter.

Test Concept: Select an Alert Enrollment object, O1, and cause it to generate a notification. Verify that the notification uses the extended notification parameters and that the first parameter is an object identifier.

Test Configuration: O1 is configured to issue unconfirmed event notifications. O2, referenced in this test, is another configured object. D1 is a vendor specific delay (may be zero).

Test Steps:

1. VERIFY pCurrentState = NORMAL
2. MAKE (a condition exist which will cause O1 to transition)
3. WAIT D1
4. BEFORE **Notification Fail Time**
   RECEIVE UnconfirmedEventNotification-Request

'Process Identifier' =	(any valid process identifier),
'Initiating Device Identifier' =	IUT,
'Event Object Identifier' =	O1,
'Time Stamp' =	(any valid timestamp),
'Priority' =	(any valid priority),
'Event Type' =	EXTENDED,
'Message Text' =	(optional, any valid message text),
'Notify Type' =	ALARM | EVENT,
'AckRequired' =	FALSE,
'From State' =	NORMAL,
'To State' =	NORMAL,
'Event Values' =	(any valid vendor id), (any valid extended event type), (O2: an object identifier), (any valid list of parameters)

5. CHECK (that O2 exists in the IUT)

7.3.2.30.2 Alert Enrollment Does Not Generate Acknowledgeable Transitions

Purpose: To verify that the Alert Enrollment object's notifications are not acknowledgable.

Test Concept: Select an Alert Enrollment object, O1, and cause it to generate a notification. Verify that the notification does not require an acknowledgment.

Test Configuration: O1 is configured to issue unconfirmed event notifications. O2, referenced in this test, is another configured object. D1 is a vendor specific delay (may be zero).

Test Steps:

1. VERIFY ~~Event_State~~pCurrentState = NORMAL
2. MAKE (a condition exist which will cause O1 to transition)
3. WAIT D1

4. BEFORE **Notification Fail Time**
   RECEIVE UnconfirmedEventNotification-Request

'Process Identifier' =	(any valid process identifier),
'Initiating Device Identifier' =	IUT,
'Event Object Identifier' =	O1,
'Time Stamp' =	(any valid timestamp),
'Priority' =	(any valid priority),
'Event Type' =	EXTENDED,
'Message Text' =	(optional, any valid message text),
'Notify Type' =	ALARM | EVENT,
'AckRequired' =	FALSE,
'From State' =	NORMAL,
'To State' =	NORMAL,
'Event Values' =	(any valid vendor id), (any valid extended event type), (O2: an object identifier), (any valid list of parameters)

5. VERIFY Acked_Transitions = (T, T, T)

[Change **Clause 8.1**, p. 152]

8.1 AcknowledgeAlarm Service Initiation Tests

~~Dependencies: None.~~

~~BACnet Reference Clause: 13.5.~~

Purpose: To verify that the IUT is capable of acknowledging alarms and events that are reported to the IUT via the ConfirmedEventNotification and UnconfirmedEventNotfication services.

Configuration *Requirements*: For this test, the tester shall choose ~~1~~*one* object, O1, in the TD, which is configured to send event notifications to the IUT. The tester places O1 into an alarm state such that the transition requires an acknowledgment.

Test Steps:

1. TRANSMIT ConfirmedEventNotification-Request | UnconfirmedEventNotification-Request,
   'Subscriber Process Identifier' = (a value acceptable to the IUT configured in the Notification Class object for the IUT),
   'Initiating Device Identifier' = TD,
   'Event Object Identifier' = O1,
   'Time Stamp' = (any valid value, T1),
   'Notification Class' = (the value configured in O1),
   'Priority' = (any value selected by the TD),
   'Event Type' = (any value selected by the TD),
   'Notify Type' = ALARM | EVENT,
   'AckRequired' = TRUE,
   'From State' = (any valid value),
   'To State' = (any valid value, S1),
   'Event Values' = (any event values appropriate to the event type)
2. IF (the ConfirmedEventNotification choice was selected) THEN
   RECEIVE BACnet-SimpleACK-PDU
3. MAKE (the IUT acknowledge O1)
4. RECEIVE AcknowledgeAlarm-Request,
   'Acknowledging Process Identifier' = (any process identifier),
   'Event Object Identifier' = O1,
   'Event State Acknowledged' = S1,
   'Time Stamp' = T1,
   'Acknowledgement Source' = (any valid value),
   'Time of Acknowledgement' = (any valid value)
5. TRANSMIT BACnet-SimpleACK-PDU

[Change **Clause 8.4.1**, p. 165]

8.4 ConfirmedEventNotification Service Initiation Tests

This clause defines the tests necessary to demonstrate support for initiating ConfirmedEventNotification service requests. The ConfirmedEventNotification tests are specific to the event detection algorithm used. For each object type that supports intrinsic event reporting, the IUT shall pass the tests for the event detection algorithm that applies to that object type. If the IUT supports the Event Enrollment object, it shall pass the tests for the event detection algorithm that corresponds to each event type supported.

For devices that implement a read-only Recipient_List property for all instances of Notification Class objects, and are exclusively configured for UnconfirmedEventNotification service initiation, and the device cannot contain a Notification Forwarder object, this clause and all subclauses shall be omitted from the testing protocol.

For devices that implement a read-only Recipient_List property for all instances of Notification Class objects, and are exclusively configured for UnconfirmedEventNotification service initiation, and the device can contain a Notification Forwarder object, this clause and all subclauses shall be applied to the Notification Forwarder objects and not to the Notification Class objects.

[Change **Clause 8.4.1**, p. 165]

8.4.1 CHANGE_OF_BITSTRING Test~~s~~ *(ConfirmedEventNotification)*

~~Dependencies: ReadProperty Service Execution Tests, 9.18; WriteProperty Service Execution Tests, 9.22.~~

~~BACnet Reference Clauses: 12.12.5, 12.12.7, 13.3.1, and 13.8.~~

Purpose: To verify the correct operation of the Change of Bitstring event algorithm.

Test Concept: The object begins the test in a NORMAL state. ~~The referenced property~~*pMonitoredValue* is changed to a value that is one of the values designated in ~~List_Of_Bitstring_Values~~*pAlarmValues*. After the time delay expires, the object should enter the OFFNORMAL state and transmit an event notification message. ~~The referenced property~~*pMonitoredValue* is then changed to a value corresponding to a NORMAL state. After the time delay, the object should enter the NORMAL state and transmit an event notification message.

Configuration Requirements: The IUT shall be configured such that the Event_Enable property has a value of TRUE for the TO-OFFNORMAL and TO-NORMAL transitions. The ~~Issue_Confirmed_Notifications property~~*'Issue Confirmed Notifications' parameter* shall have a value of TRUE. The event-generating object~~s~~ shall be in a NORMAL state, at the start of the test.

Test Steps:

1. VERIFY ~~Event_State~~*pCurrentState* = NORMAL
2. IF (~~the referenced property~~*pMonitoredValue* is writable) THEN
   WRITE ~~(referenced property~~*pMonitoredValue*~~)~~ = (a value ~~x: x = one of the List_Of_Bitstring_Values~~*from the pAlarmValues list* after ~~the bitmask~~*pBitmask* is applied)
   ELSE
   MAKE (~~the referenced property~~*pMonitoredValue* have a value ~~x: x = one of the List_Of_Bitstring_Values~~*from the pAlarmValues list* after the ~~bitmask~~*pBitmask* is applied)
3. WAIT (~~Time_Delay~~*pTimeDelay*)
4. BEFORE **Notification Fail Time**
   RECEIVE ConfirmedEventNotification-Request,
   'Process Identifier' = (any valid process ID),
   'Initiating Device Identifier' = IUT,
   'Event Object Identifier' = (the object being tested),
   'Time Stamp' = (~~the current local time~~*any valid time stamp*),

'Notification Class' = (the configured notification class),
'Priority' = (the value configured to correspond to a TO-OFFNORMAL transition),
'Event Type' = CHANGE_OF_BITSTRING,
'Message Text' = *(optional, any valid message text),*
'Notify Type' = EVENT | ALARM,
'AckRequired' = TRUE | FALSE,
'From State' = NORMAL,
'To State' = OFFNORMAL,
'Event Values' = ~~referenced bitstring~~*pMonitoredValue*, ~~Status_Flags~~*pStatusFlags*

5. TRANSMIT BACnet-SimpleACK-PDU
6. ~~IF (the object being tested is not an Event Enrollment object) THEN~~*IF (Protocol_Revision is present AND Protocol_Revision e 13) THEN*
   VERIFY ~~Status_Flags~~*pStatusFlags* = (TRUE, FALSE, ?, ?)
7. VERIFY ~~Event_State~~*pCurrentState* = OFFNORMAL
8. IF (Protocol_Revision is present and Protocol_Revision ≥ 1) THEN
   VERIFY Event_Time_Stamps = (the timestamp in step 4, *, *)
9. IF (~~Present_Value~~*pMonitoredValue* is writable) THEN
   WRITE (~~referenced property~~*pMonitoredValue*) = (a value ~~x: x~~*that* corresponds to a NORMAL state)
   ELSE
   MAKE (~~the referenced property~~*pMonitoredValue* have a value ~~x: x~~*that* corresponds to a NORMAL state)
10. WAIT (~~Time_Delay~~*pTimeDelayNormal*)
11. BEFORE **Notification Fail Time**
    RECEIVE ConfirmedEventNotification-Request,
    'Process Identifier' = (any valid process ID),
    'Initiating Device Identifier' = IUT,
    'Event Object Identifier' = (the object being tested),
    'Time Stamp' = (~~the current local time~~*any valid time stamp*),
    'Notification Class' = (the configured notification class),
    'Priority' = (the value configured to correspond to a TO-NORMAL transition),
    'Event Type' = CHANGE_OF_BITSTRING,
    'Message Text' = *(optional, any valid message text),*
    'Notify Type' = EVENT | ALARM,
    'AckRequired' = TRUE | FALSE,
    'From State' = OFFNORMAL,
    'To State' = NORMAL,
    'Event Values' = ~~referenced bitstring~~*pMonitoredValue*, ~~Status_Flags~~*pStatusFlags*
12. TRANSMIT BACnet-SimpleACK-PDU
13. ~~IF (the object being tested is not an Event Enrollment object) THEN~~*IF (Protocol_Revision is present AND Protocol_Revision e 13) THEN*
    VERIFY ~~Status_Flags~~*pStatusFlags* = (FALSE, FALSE, ?, ?)
14. VERIFY ~~Event_State~~*pCurrentState* = NORMAL
15. IF (Protocol_Revision is present and Protocol_Revision ≥ 1) THEN
    VERIFY Event_Time_Stamps = (the timestamp in step 4, *, the timestamp in step 11)

Notes to Tester: ~~The 'Message Text' parameter is omitted in the test description because it is optional. The IUT may include this parameter in the notification messages.~~ The time stamps indicated by "*" in steps 8 and 15 can have a value that indicates an unspecified time or a time that precedes the timestamp in step 4.

[Change **Clause 8.4.2**, p. 167]

8.4.2 CHANGE_OF_STATE Tests *(ConfirmedEventNotification)*

~~Dependencies: ReadProperty Service Execution Tests, 9.18; WriteProperty Service Execution Tests, 9.22.~~

~~BACnet Reference Clauses: 12.6, 12.8, 12.18, 12.20, 13.2,13.3.2, and 13.8.~~

Purpose: To verify the correct operation of the CHANGE_OF_STATE event algorithm. ~~This test applies to Event Enrollment objects with an Event_Type of CHANGE_OF_STATE, and to intrinsic event reporting for Binary Input, Binary Value, Multi-state Input and Multi-state Value objects.~~

Test Concept: The object begins the test in a NORMAL state. The Present_Value (referenced property) is changed to a value that is one of the values designated in List_Of_Values. After the time delay expires, the object should enter the OFFNORMAL state and transmit an event notification message. The Present_Value (referenced property) is then changed to a value corresponding to a NORMAL state. After the time delay, the object should enter the NORMAL state and transmit an event notification message. ~~For Multi-state Input and Multi-state Value objects there is a special case of the CHANGE_OF_STATE algorithm that applies to transitions to the FAULT state. The test procedure includes a test for this special case.~~ *If the IUT claims conformance to Protocol_Revision 12 or lower, and a Multi-state Input or Multi-state Value object is being tested, the transition to and from the FAULT state is also tested.*

Configuration Requirements: The IUT shall be configured such that the Event_Enable property has a value of TRUE for the TO-OFFNORMAL, TO-FAULT and TO-NORMAL transitions. The ~~Issue_Confirmed_Notifications property~~ *'Issue Confirmed Notifications' parameter* shall have a value of TRUE. The event-generating object~~s~~ shall be in a NORMAL state at the start of the test.

If the IUT claims conformance to Protocol_Revision 12 or lower, and supports intrinsic reporting for Multi-state Input or Multi-state Value objects, the intrinsic reporting object shall be configured with at least one of the two properties, Alarm_Values (referred to as pAlarmValues in the test steps) and Fault_Values (referred to as pFaultValues in the test steps), containing at least one value.

If the IUT claims conformance to Protocol_Revision 12 or lower, and supports intrinsic reporting for Binary Input or Binary Value objects, the intrinsic reporting object shall be configured with the Alarm_Value property (referred to as pAlarmValues in the test steps) containing at least one value.

If the IUT claims conformance to Protocol_Revision 12 or lower, and supports algorithmic change reporting with an Event_Type of CHANGE_OF_STATE, the List_Of_Values parameter of the Event_Parameters property (referred to as pAlarmValues in the test steps) shall contain at least one value.

If the IUT claims conformance to Protocol_Revision 13 or greater, and supports the CHANGE_OF_STATE algorithm, the IUT shall be configured with at least one value for pAlarmValues.

~~In the test description below Present_Value is used as the referenced property. If an Event Enrollment object is being tested Present_Value should be replaced by the appropriate property reference.~~

Test Steps:

1. VERIFY ~~Event_State~~*pCurrentState* = NORMAL

~~2. IF (the object, or referenced object, if using Event Enrollment, is a binary object or it is a multi-state object with a non-empty Alarm_Values property) THEN~~

2. IF ((Protocol_Revision is present AND Protocol_Revision ≥ 13) OR ((Protocol_Revision is present AND Protocol_Revision d 12) AND (pAlarmValues contains at least one value))) THEN {

~~3. IF (Present_Value is writable) THEN~~
~~WRITE Present_Value = (a value x: x = Alarm_Value for binary objects or one of the Alarm_Values for multi-state objects)~~
~~ELSE~~
~~MAKE (Present_Value have a value x: x = Alarm_Value for binary objects or one of the Alarm_Values for multi-state objects)~~

3. IF (pMonitoredValue is writable) THEN
WRITE pMonitoredValue = (a value from pAlarmValues)
ELSE
MAKE (pMonitoredValue have a value from pAlarmValues)

4. WAIT (~~Time_Delay~~*pTimeDelay*)
5. BEFORE **Notification Fail Time**
   RECEIVE ConfirmedEventNotification-Request,
   'Process Identifier' = (any valid process ID),
   'Initiating Device Identifier' = IUT,
   'Event Object Identifier' = (the intrinsic reporting object being tested or the Event Enrollment object being tested),
   'Time Stamp' = (T1, the current local time),
   'Notification Class' = (the configured notification class),
   'Priority' = (the value configured to correspond to a ~~TO-OFFNORMAL~~ *TO_OFFNORMAL* transition),
   'Event Type' = CHANGE_OF_STATE,
   'Message Text' = *(optional, any valid message text),*
   'Notify Type' = EVENT | ALARM,
   'AckRequired' = TRUE | FALSE,
   'From State' = NORMAL,
   'To State' = OFFNORMAL,
   'Event Values' = ~~Present_Value~~*pMonitoredValue*, ~~Status_Flags~~*pStatusFlags*
6. TRANSMIT BACnet-SimpleACK-PDU
7. ~~IF (the object being tested is NOT an Event Enrollment object) THEN~~*IF (Protocol_Revision is present AND Protocol_Revision e 13) THEN*
   VERIFY ~~Status_Flags~~*pStatusFlags* = (TRUE, FALSE, ?, ?)
8. VERIFY ~~Event_State~~*pCurrentState* = OFFNORMAL
9. IF (Protocol_Revision is present and Protocol_Revision ≥ 1) THEN
   VERIFY Event_Time_Stamps = (T1, *, *)
10. IF (~~Present_Value~~*pMonitoredValue* is writable) THEN
    WRITE ~~Present_Value~~*pMonitoredValue* = (a value ~~x: x~~*that* corresponds to a NORMAL state)
    ELSE
    MAKE (~~Present_Value~~*pMonitoredValue* have a value ~~x: x~~*that* corresponds to a NORMAL state)
11. WAIT (~~Time_Delay~~*pTimeDelayNormal*)
12. BEFORE **Notification Fail Time**
    RECEIVE ConfirmedEventNotification-Request,
    'Process Identifier' = (any valid process ID),
    'Initiating Device Identifier' = IUT,
    'Event Object Identifier' = (the intrinsic reporting object being tested or the Event Enrollment object being tested),
    'Time Stamp' = (T2, the current local time),
    'Notification Class' = (the configured notification class),
    'Priority' = (the value configured to correspond to a ~~TO-OFFNORMAL~~*TO_OFFNORMAL* transition),
    'Event Type' = CHANGE_OF_STATE,
    'Message Text' = *(optional, any valid message text),*
    'Notify Type' = EVENT | ALARM,
    'AckRequired' = TRUE | FALSE,
    'From State' = OFFNORMAL,
    'To State' = NORMAL,
    'Event Values' = ~~Present_Value~~*pMonitoredValue*, ~~Status_Flags~~*pStatusFlags*
13. TRANSMIT BACnet-SimpleACK-PDU
14. ~~IF (the object being tested is NOT an Event Enrollment object) THEN~~*IF (Protocol_Revision is present AND Protocol_Revision e 13) THEN*
    VERIFY ~~Status_Flags~~*pStatusFlags* = (FALSE, FALSE, ?, ?)
15. VERIFY ~~Event_State~~*pCurrentState* = NORMAL
16. IF (Protocol_Revision is present and Protocol_Revision ≥ 1) THEN
    VERIFY Event_Time_Stamps = (T1, *, T2)

}

~~17. IF (the object being tested is a multi-state object that supports intrinsic reporting) THEN~~

~~18. IF (Present_Value is writable) THEN~~
~~WRITE Present_Value = (a value x: x = one of the Fault_Values)~~
~~ELSE~~
~~MAKE (Present_Value have a value x: x = one of the Fault_Values)~~
~~19. BEFORE **Notification Fail Time**~~
~~RECEIVE ConfirmedEventNotification-Request,~~
~~'Process Identifier' = (any valid process ID),~~
~~'Initiating Device Identifier' = IUT,~~
~~'Event Object Identifier' = (the intrinsic reporting object being tested),~~
~~'Time Stamp' = (T3, the current local time),~~
~~'Notification Class' = (the configured notification class),~~
~~'Priority' = (the value configured to correspond to a TO-FAULT transition),~~
~~'Event Type' = CHANGE_OF_STATE,~~
~~'Notify Type' = EVENT | ALARM,~~
~~'AckRequired' = TRUE | FALSE,~~
~~'From State' = NORMAL,~~
~~'To State' = FAULT,~~
~~'Event Values' = Present_Value, Status_Flags~~
~~20. TRANSMIT BACnet-SimpleACK-PDU~~
~~21. IF (the object being tested is NOT an Event Enrollment object) THEN~~
~~VERIFY Status_Flags = (FALSE, TRUE, ?, ?)~~
~~22. VERIFY Event_State = FAULT~~
~~23. IF (Protocol_Revision is present and Protocol_Revision ≥ 1) THEN~~
~~VERIFY Event_Time_Stamps = (T1, T3, T2)~~
~~24. IF (the object being tested is a multi-state object that supports intrinsic reporting and Protocol_Revision is present and Protocol_Revision ≥ 1) THEN~~
~~VERIFY Reliability = MULTI_STATE_FAULT~~
~~25. IF (Present_Value is writable) THEN~~
~~WRITE Present_Value = (a value x: x corresponds to a NORMAL state)~~
~~ELSE~~
~~MAKE (Present_Value have a value x: x corresponds to a NORMAL state)~~
~~26. BEFORE **Notification Fail Time**~~
~~RECEIVE ConfirmedEventNotification-Request,~~
~~'Process Identifier' = (any valid process ID),~~
~~'Initiating Device Identifier' = IUT,~~
~~'Event Object Identifier' = (the intrinsic reporting object being tested),~~
~~'Time Stamp' = (T4, the current local time),~~
~~'Notification Class' = (the configured notification class),~~
~~'Priority' = (the value configured to correspond to a TO-NORMAL transition),~~
~~'Event Type' = CHANGE_OF_STATE,~~
~~'Notify Type' = EVENT | ALARM,~~
~~'AckRequired' = TRUE | FALSE,~~
~~'From State' = FAULT,~~
~~'To State' = NORMAL,~~
~~'Event Values' = Present_Value, Status_Flags~~
~~27. TRANSMIT BACnet-SimpleACK-PDU~~
~~28. IF (the object being tested is NOT an Event Enrollment object) THEN~~
~~VERIFY Status_Flags = (FALSE, FALSE, ?, ?)~~
~~29. VERIFY Event_State = NORMAL~~
~~30. IF (Protocol_Revision is present and Protocol_Revision ≥ 1) THEN~~
~~VERIFY Event_Time_Stamps = (T1, T3, T4)~~

17. IF ((Protocol_Revision is present AND Protocol_Revision d 12) AND (intrinsic reporting is being tested) AND (the intrinsic reporting object is configured with pFaultValues containing at least one values)) THEN {
18. IF (pMonitoredValue is writable) THEN
WRITE pMonitoredValue = (a value from pFaultValues)
ELSE

MAKE (pMonitoredValue have a value from pFaultValues)

19. *BEFORE **Notification Fail Time***
RECEIVE ConfirmedEventNotification-Request,
'Process Identifier' = (any valid process ID),
'Initiating Device Identifier' = IUT,
'Event Object Identifier' = (the intrinsic reporting object being tested),
'Time Stamp' = (Tfault: any valid timestamp),
'Notification Class' = (the configured notification class),
'Priority' = (the value configured to correspond to a TO_FAULT transition),
'Event Type' = CHANGE_OF_STATE,
'Message Text' = (optional, any valid message text),
'Notify Type' = EVENT | ALARM,
'AckRequired' = TRUE | FALSE,
'From State' = NORMAL,
'To State' = FAULT,
'Event Values' = pMonitoredValue, pStatusFlags

20. *TRANSMIT BACnet-SimpleACK-PDU*

21 *VERIFY pCurrentState = FAULT*

22. *IF (Protocol_Revision is present AND Protocol_Revision ≥ 1) THEN*
VERIFY Event_Time_Stamps = (Toffnormal, Tfault, Tnormal)

23. *VERIFY pCurrentReliability = MULTI_STATE_FAULT*

24. *IF (pMonitoredValue is writable) THEN*
WRITE pMonitoredValue = (a value that corresponds to a NORMAL state)
ELSE
MAKE (pMonitoredValue have a value that corresponds to a NORMAL state)

25. *BEFORE **Notification Fail Time***
RECEIVE ConfirmedEventNotification-Request,
'Process Identifier' = (any valid process ID),
'Initiating Device Identifier' = IUT,
'Event Object Identifier' = (the intrinsic reporting object being tested),
'Time Stamp' = (Tnormal: any valid timestamp),
'Notification Class' = (the configured notification class),
'Priority' = (the value configured to correspond to a TO_NORMAL transition),
'Event Type' = CHANGE_OF_STATE,
'Message Text' = (optional, any valid message text),
'Notify Type' = EVENT | ALARM,
'AckRequired' = TRUE | FALSE,
'From State' = FAULT,
'To State' = NORMAL,
'Event Values' = pMonitoredValue, pStatusFlags

26. *TRANSMIT BACnet-SimpleACK-PDU*

27. *VERIFY pCurrentState = NORMAL*

28. *IF (Protocol_Revision is present AND Protocol_Revision ≥ 1) THEN*
VERIFY Event_Time_Stamps = (Toffnormal, Tfault, Tnormal)

}

Notes to Tester: ~~The 'Message Text' parameter is omitted in the test description because it is optional. The IUT may include this parameter in the notification messages~~. The time stamps indicated by "*" in *steps 9 and 16* can have a value that indicates an unspecified time or a time that precedes the timestamp T1. *pCurrentReliability refers to the Reliability property of the event-generating object for this test.*

[Change **Clause 8.4.3**, p. 169]

8.4.3 CHANGE_OF_VALUE Tests *(ConfirmedEventNotification)*

This clause defines the tests necessary to demonstrate support for the CHANGE_OF_VALUE event algorithm. The CHANGE_OF_VALUE algorithm can be applied to both numerical and bitstring datatypes. The IUT shall pass the tests for both applications.

8.4.3.1 Numerical Algorithm *(ConfirmedEventNotification)*

~~The test in this clause applies to use of the CHANGE_OF_VALUE algorithm applied to Integer or Real datatypes.~~

~~Dependencies: ReadProperty Service Execution Tests, 9.18; WriteProperty Service Execution Tests, 9.22.~~

~~BACnet Reference Clauses: 12.12, 13.3.2, and 13.8.~~

Purpose: To verify the correct operation of the CHANGE_OF_VALUE event algorithm as applied to numerical datatypes.

Test Concept: The object begins the test in a NORMAL state. *pMonitoredValue*~~The referenced property~~ is changed by a value that is less than ~~the Referenced_Property_Increment~~*pIncrement*. The tester verifies that no event notification is transmitted. ~~The referenced property~~*pMonitoredValue* is changed again to a value that differs from the original value by an amount greater than ~~the Referenced_Property_Increment~~*pIncrement*. The tester verifies that an event notification message is transmitted and that the proper ~~Event_State~~*event state* transitions occur.

Configuration Requirements: The IUT shall be configured such that the Event_Enable property has a value of TRUE for the TO-NORMAL transition. The ~~Issue_Confirmed_Notifications property~~*'Issue Confirmed Notifications' parameter* shall have a value of TRUE. The event-generating object shall be in a NORMAL state at the start of the test.

Test Steps:

1. VERIFY ~~Event_State~~*pCurrentState* = NORMAL
2. IF (~~the referenced property~~*pMonitoredValue* is writable) THEN
   WRITE (~~referenced property~~*pMonitoredValue*) = (a value ~~x: x~~*that* differs from the initial value by less than ~~Referenced_Property_Increment~~*pIncrement*)
   ELSE
   MAKE (~~the referenced property~~*pMonitoredValue* have a value ~~x: x~~*that* differs from the initial value by less than ~~Referenced_Property_Increment~~*pIncrement*)
3. WAIT (~~Time_Delay~~*pTimeDelayNormal* + **Notification Fail Time**)
4. CHECK (verify that no event notification message is transmitted)
5. IF (~~the referenced property~~*pMonitoredValue* is writable) THEN
   WRITE (~~referenced property~~*pMonitoredValue*) = (a value ~~x: x~~*that* differs from the initial value in step 1 by more than ~~Referenced_Property_Increment~~*pIncrement*)
   ELSE
   MAKE (~~the referenced property~~*pMonitoredValue* have a value ~~x: x~~*that* differs from the initial value in step 1 by more than ~~Referenced_Property_Increment~~*pIncrement*)
6. WAIT (~~Time_Delay~~*pTimeDelayNormal*)
7. BEFORE **Notification Fail Time**
   RECEIVE ConfirmedEventNotification-Request,

'Process Identifier' =	(any valid process ID),
'Initiating Device Identifier' =	IUT,
'Event Object Identifier' =	(the object being tested),
'Time Stamp' =	(~~the current local time~~*any valid time stamp*),
'Notification Class' =	(the configured notification class),
'Priority' =	(the value configured to correspond to a TO-NORMAL transition),
'Event Type' =	CHANGE_OF_VALUE,
'Message Text' =	*(optional, any valid message text),*
'Notify Type' =	EVENT | ALARM,

'AckRequired' = TRUE | FALSE,
'From State' = NORMAL,
'To State' = NORMAL,
'Event Values' = ~~changed value~~*pMonitoredValue*, ~~Status_Flags~~*pStatusFlags*

8. TRANSMIT BACnet-SimpleACK-PDU
9. ~~IF (the object being tested is not an Event Enrollment object) THEN~~*IF (Protocol_Revision is present AND Protocol_Revision e 13) THEN*
   VERIFY ~~Status_Flags~~*pStatusFlags* = (FALSE, FALSE, ?, ?)
10. VERIFY ~~Event_State~~*pCurrentState* = NORMAL
11. IF (Protocol_Revision is present and Protocol_Revision ≥ 1) THEN
    VERIFY Event_Time_Stamps = (*, *, the timestamp in step 7)

Notes to Tester: ~~The 'Message Text' parameter is omitted in the test description because it is optional. The IUT may include this parameter in the notification messages~~. The time stamps indicated by "*" in step 11 can have a value that indicates an unspecified time or a time that precedes the timestamp in step 7.

8.4.3.2 Bitstring Algorithm *(ConfirmedEventNotification)*

~~The test in this clause applies to use of the CHANGE_OF_VALUE algorithm applied to Bitstring datatypes.~~

~~Dependencies: ReadProperty Service Execution Tests, 9.18; WriteProperty Service Execution Tests, 9.22.~~

~~BACnet Reference Clauses: 12.12, 13.3.2, and 13.8.~~

Purpose: To verify the correct operation of the CHANGE_OF_VALUE event algorithm as applied to Bitstring datatypes.

Test Concept: The object begins the test in a NORMAL state. ~~The referenced property~~*pMonitoredValue* is changed to a new value such that none of the bits in the *p*Bitmask are changed. The tester verifies that no event notification is transmitted. ~~The referenced property~~*pMonitoredValue* is changed again to a value that differs in one or more bits that are included in the *p*Bitmask. The tester verifies that an event notification message is transmitted and that the proper ~~Event_State~~*event state* transitions occur.

Configuration Requirements: The IUT shall be configured such that the Event_Enable property has a value of TRUE for the TO-NORMAL transition. The ~~Issue_Confirmed_Notifications property~~*'Issue Confirmed Notifications' parameter* shall have a value of TRUE. ~~The~~ *p*Bitmask shall be configured so that at least one but not all bits of ~~the referenced property~~*pMonitoredValue* are included in the mask. The event-generating object shall be in a NORMAL state at the start of the test.

Test Steps:

1. VERIFY ~~Event_State~~*pCurrentState* = NORMAL
2. IF (~~the referenced property~~*pMonitoredValue* is writable) THEN
   WRITE (~~referenced property~~*pMonitoredValue*) = (a value ~~x: x~~*that* differs from the initial value but only in bits that are not included in ~~Bitmask~~*pBitmask*)
   ELSE
   MAKE (~~the referenced property~~*pMonitoredValue* have a value ~~x: x~~*that* differs from the initial value but only in bits that are not included in ~~Bitmask~~*pBitmask*)
3. WAIT (~~Time_Delay~~*pTimeDelayNormal* + **Notification Fail Time**)
4. CHECK (verify that no event notification message is transmitted)
5. IF (~~the referenced property~~*pMonitoredValue* is writable) THEN
   WRITE (~~referenced property~~*pMonitoredValue*) = (a value ~~x: x~~*that* differs from the initial value in one or more bits included in ~~Bitmask~~*pBitmask*)
   ELSE
   MAKE (~~the referenced property~~*pMonitoredValue* have a value ~~x: x~~*that* differs from the initial value one or more bits included in ~~Bitmask~~*pBitmask*)

6. WAIT (~~Time_Delay~~*pTimeDelayNormal*)
7. BEFORE **Notification Fail Time**
   RECEIVE ConfirmedEventNotification-Request,
   'Process Identifier' = (any valid process ID),
   'Initiating Device Identifier' = IUT,
   'Event Object Identifier' = (the object being tested),
   'Time Stamp' = (~~the current local time~~*any valid time stamp*),
   'Notification Class' = (the configured notification class),
   'Priority' = (the value configured to correspond to a TO-NORMAL transition),
   'Event Type' = CHANGE_OF_VALUE,
   'Notify Type' = EVENT | ALARM,
   'AckRequired' = TRUE | FALSE,
   'From State' = NORMAL,
   'To State' = NORMAL,
   'Event Values' = ~~changed-value~~*pMonitoredValue*, ~~Status_Flags~~*pStatusFlags*
8. TRANSMIT BACnet-SimpleACK-PDU
9. ~~IF (the object being tested is not an Event Enrollment object) THEN~~*IF (Protocol_Revision is present AND Protocol_Revision e 13) THEN*
   VERIFY ~~Status_Flags~~*pStatusFlags* = (FALSE, FALSE, ?, ?)
10. VERIFY ~~Event_State~~*pCurrentState* = NORMAL
11. IF (Protocol_Revision is present and Protocol_Revision ≥ 1) THEN
    VERIFY Event_Time_Stamps = (*, *, the timestamp in step 7)

Notes to Tester: ~~The 'Message Text' parameter is omitted in the test description because it is optional. The IUT may include this parameter in the notification messages~~. The time stamps indicated by "*" in step 11 can have a value that indicates an unspecified time or a time that precedes the timestamp in step 7.

[Change **Clause 8.4.4**, p. 172]

8.4.4 COMMAND_FAILURE Tests *(ConfirmedEventNotification)*

~~Dependencies: ReadProperty Service Execution Tests, 9.18; WriteProperty Service Execution Tests, 9.22.~~

~~BACnet Reference Clauses: 12.7, 12.12, 12.19, 13.2, 13.3.4, and 13.8.~~

Purpose: To verify the correct operation of the COMMAND_FAILURE algorithm.

Test Concept: ~~The Feedback_Value (Feedback_Property_Reference)~~*pFeedbackValue* shall be decoupled from the input signal that is normally used to verify the output. Initially *pMonitoredValue* and ~~Feedback_Value (Feedback_Property_Reference)~~*pFeedbackValue* are in agreement. *pMonitoredValue* is changed and an event notification should be transmitted indicating a transition to an OFFNORMAL state. ~~The Feedback_Value (Feedback_Property_Reference)~~*pFeedbackValue* is changed to again agree with the *pMonitoredValue*. A second event notification is transmitted, indicating a return to a NORMAL state.

Configuration Requirements: The IUT shall be configured such that the Event_Enable property has a value of TRUE for the TO-OFFNORMAL and TO-NORMAL transitions. The ~~Issue_Confirmed_Notifications property~~*'Issue Confirmed Notifications' parameter* shall have a value of TRUE. The event-generating object shall be in a NORMAL state at the start of the test. The Feedback_Value property shall be decoupled from the input signal that is normally used to verify the output so that it can be independently manipulated.

~~In the test description below Present_Value is used as the referenced property and Feedback_Value is used to verify the output. If an Event Enrollment object is being tested these properties shall be replaced by the appropriate property reference.~~

Test Steps:

1. VERIFY ~~Event_State~~*pCurrentState* = NORMAL
2. ~~IF (the object being tested is not an Event Enrollment object) THEN~~*IF (Protocol_Revision is present AND Protocol_Revision e 13) THEN*
   VERIFY ~~Status_Flags~~*pStatusFlags* = (FALSE, FALSE, FALSE, FALSE)
3. IF (~~Present_Value~~*pMonitoredValue* is writable) THEN
   WRITE ~~Present_Value~~*pMonitoredValue* = (a different value)
   ELSE
   MAKE (~~Present_Value~~*pMonitoredValue* take on a different value)
4. WAIT (~~Time_Delay~~*pTimeDelay*)
5. BEFORE **Notification Fail Time**
   RECEIVE ConfirmedEventNotification-Request,
   'Process Identifier' = (any valid process ID),
   'Initiating Device Identifier' = IUT,
   'Event Object Identifier' = (the object being tested),
   'Time Stamp' = (~~the current local time~~*any valid time stamp*),
   'Notification Class' = (the configured notification class),
   'Priority' = (the value configured to correspond to a TO-OFFNORMAL transition),
   'Event Type' = COMMAND_FAILURE,
   'Message Text' = *(optional, any valid message text),*
   'Notify Type' = EVENT | ALARM,
   'AckRequired' = TRUE | FALSE,
   'From State' = NORMAL,
   'To State' = OFFNORMAL,
   'Event Values' = ~~Present_Value~~*pMonitoredValue*, ~~Status_Flags~~*pStatusFlags*, ~~Feedback_Value~~*pFeedbackValue*
6. TRANSMIT BACnet-SimpleACK-PDU
7. ~~IF (the object being tested is not an Event Enrollment object) THEN~~*IF (Protocol_Revision is present AND Protocol_Revision e 13) THEN*
   VERIFY ~~Status_Flags~~*pStatusFlags* = (TRUE, FALSE, ?, ?)
8. VERIFY ~~Event_State~~*pCurrentState* = OFFNORMAL
9. IF (Protocol_Revision is present and Protocol_Revision ≥ 1) THEN
   VERIFY Event_Time_Stamps = (the timestamp in step 5, *, *)
10. IF (~~Feedback_Value~~*pFeedbackValue* is writable) THEN
    WRITE ~~Feedback_Value~~*pFeedbackValue* = (a value consistent with ~~Present_Value~~*pMonitoredValue*)
    ELSE
    MAKE (~~Feedback_Value~~*pFeedbackValue* take on a value consistent with ~~Present_Value~~*pMonitoredValue*)
11. WAIT (~~Time_Delay~~*pTimeDelayNormal*)
12. BEFORE **Notification Fail Time**
    RECEIVE ConfirmedEventNotification-Request,
    'Process Identifier' = (any valid process ID),
    'Initiating Device Identifier' = IUT,
    'Event Object Identifier' = (the object being tested),
    'Time Stamp' = (~~the current local time~~*any valid time stamp*),
    'Notification Class' = (the configured notification class),
    'Priority' = (the value configured to correspond to a TO-NORMAL transition),
    'Event Type' = COMMAND_FAILURE,
    'Message Text' = *(optional, any valid message text),*
    'Notify Type' = EVENT | ALARM,
    'AckRequired' = TRUE | FALSE,
    'From State' = OFFNORMAL,
    'To State' = NORMAL,
    'Event Values' = ~~Present_Value~~*pMonitoredValue*, ~~Status_Flags~~*pStatusFlags*, ~~Feedback_Value~~*pFeedbackValue*
13. TRANSMIT BACnet-SimpleACK-PDU
14. ~~IF (the object being tested is not an Event Enrollment object) THEN~~*IF (Protocol_Revision is present AND Protocol_Revision e 13) THEN*

VERIFY ~~Status_Flags~~*pStatusFlags* = (FALSE, FALSE, ?, ?)

15. VERIFY ~~Event_State~~*pCurrentState* = NORMAL
16. IF (Protocol_Revision is present and Protocol_Revision ≥ 1) THEN
    VERIFY Event_Time_Stamps = (the timestamp in step 5, *, the timestamp in step 12)

Notes to Tester: ~~The 'Message Text' parameter is omitted in the test description because it is optional. The IUT may include this parameter in the notification messages~~. The time stamps indicated by "*" in steps 9 and 16 can have a value that indicates an unspecified time or a time that precedes the timestamp in step 5.

[Change **Clause 8.4.5**, p. 173]

8.4.5 FLOATING_LIMIT Test~~s~~ ***(ConfirmedEventNotification)***

~~Dependencies: ReadProperty Service Execution Tests, 9.18; WriteProperty Service Execution Tests, 9.22.~~

~~BACnet Reference Clauses: 12.12, 12.17, 12.21, 13.2, 13.3.5, and 13.8.~~

Purpose: To verify the correct operation of the Floating Limit event algorithm. ~~When testing Loop objects both High_Diff_Limit and Low_Diff_Limit shall be replaced by Error_Limit in the test description below.~~

Test Concept: The object begins the test in a NORMAL state. The referenced property is raised to a value that is below but within *p*Deadband of ~~the high limit~~*pHighDiffLimit*. At this point the object should still be in a NORMAL state. ~~The referenced property~~*pMonitoredValue* is raised to a value that is above ~~the high limit~~*pHighDiffLimit*. After ~~the time delay~~*pTimeDelay* expires, the object should enter the HIGH_LIMIT state and transmit an event notification message. The referenced property is lowered to a value that is below ~~the high limit~~*pHighDiffLimit* but still within *p*Deadband of ~~the limit~~*pHighDiffLimit*. The object should remain in the HIGH_LIMIT state. ~~The referenced property~~*pMonitoredValue* is lowered further to a normal value that is not within *p*Deadband of a limit. After ~~the time delay~~*pTimeDelayNormal* expires, the object should enter the NORMAL state and issue an event notification. The same process is repeated to test ~~the low limit~~*pLowDiffLimit*.

Configuration Requirements: The IUT shall be configured such that the Event_Enable property has a value of TRUE for the TO-OFFNORMAL and TO-NORMAL transitions. The ~~Issue_Confirmed_Notifications property~~*'Issue Confirmed Notifications' parameter* shall have a value of TRUE. The event-generating ~~objects~~*object* shall be in a NORMAL state at the start of the test.

Test Steps:

1. VERIFY ~~Event_State~~*pCurrentState* = NORMAL
2. IF (~~the referenced property~~*pMonitoredValue* is writable) THEN
   WRITE (~~referenced property~~*pMonitoredValue*) = (a value x:
   (~~Setpoint_Reference~~*pSetpoint* + ~~High_Diff_Limit~~*pHighDiffLimit* – ~~Deadband~~*pDeadband*)< x < (~~Setpoint_Reference~~*pSetpoint* + ~~High_Diff_Limit~~*pHighDiffLimit*))
   ELSE
   MAKE (~~the referenced property~~*pMonitoredValue* have a value x:
   (~~Setpoint_Reference~~*pSetpoint* + ~~High_Diff_Limit~~*pHighDiffLimit* – ~~Deadband~~*pDeadband*)< x < (~~Setpoint_Reference~~*pSetpoint* + ~~High_Diff_Limit~~*pHighDiffLimit*))
3. WAIT (~~Time_Delay~~*pTimeDelay* + **Notification Fail Time**)
4. CHECK (verify that no notification message has been transmitted)
5. VERIFY ~~Event_State~~*pCurrentState* = NORMAL
6. IF (~~the referenced property~~*pMonitoredValue* is writable) THEN
   WRITE (~~referenced property~~*pMonitoredValue*) = (a value x: x > (~~Setpoint_Reference~~*pSetpoint* + ~~High_Diff_Limit~~*pHighDiffLimit*))
   ELSE
   MAKE (~~the referenced property~~*pMonitoredValue* have a value x: x > (~~Setpoint_Reference~~*pSetpoint* + ~~High_Diff_Limit~~*pHighDiffLimit*))
7. WAIT (~~Time_Delay~~*pTimeDelay*)

8. BEFORE **Notification Fail Time**
   RECEIVE ConfirmedEventNotification-Request,
   'Process Identifier' = (any valid process ID),
   'Initiating Device Identifier' = IUT,
   'Event Object Identifier' = (the object being tested),
   'Time Stamp' = (~~the current local time~~*any valid time stamp*),
   'Notification Class' = (the configured notification class),
   'Priority' = (the value configured to correspond to a TO-OFFNORMAL transition),
   'Event Type' = FLOATING_LIMIT,
   'Notify Type' = EVENT | ALARM,
   'AckRequired' = TRUE | FALSE,
   'From State' = NORMAL,
   'To State' = HIGH_LIMIT,
   'Event Values' = ~~reference value~~*pMonitoredValue*, ~~Status_Flags~~*pStatusFlags*, ~~setpoint value~~*pSetpoint*, ~~error limit~~*pHighDiffLimit*~~,~~
9. TRANSMIT BACnet-SimpleACK-PDU
10. ~~IF (the object being tested is not an Event Enrollment object) THEN~~*IF (Protocol_Revision is present AND Protocol_Revision e 13) THEN*
    VERIFY ~~Status_Flags~~*pStatusFlags* = (TRUE, FALSE, ?, ?)
11. VERIFY ~~Event_State~~*pCurrentState* = ~~HIGH_LIMIT~~*pHighDiffLimit*
12. IF (Protocol_Revision is present and Protocol_Revision ≥ 1) THEN
    VERIFY Event_Time_Stamps = (the timestamp in step 8, *, *)
13. IF (~~the referenced property~~*pMonitoredValue* is writable) THEN
    WRITE (~~referenced property~~*pMonitoredValue*) = (a value x:
    (~~Setpoint_Reference~~*pSetpoint* + ~~High_Diff_Limit~~*pHighDiffLimit* – ~~Deadband~~*pDeadband*)< x < ~~Setpoint_Reference~~*pSetpoint* + ~~High_Diff_Limit~~*pHighDiffLimit*))
    ELSE
    MAKE (~~the referenced property~~*pMonitoredValue* have a value x:
    (~~Setpoint_Reference~~*pSetpoint* + ~~High_Diff_Limit~~*pHighDiffLimit* – ~~Deadband~~*pDeadband*)< x < ~~Setpoint_Reference~~*pSetpoint* + ~~High_Diff_Limit~~*pHighDiffLimit*))
14. WAIT (~~Time_Delay~~*pTimeDelayNormal* + **Notification Fail Time**)
15. CHECK (verify that no notification message has been transmitted)
16. VERIFY ~~Event_State~~*pCurrentState* = ~~HIGH_LIMIT~~*pHighDiffLimit*
17. IF (~~the referenced property~~*pMonitoredValue* is writable) THEN
    WRITE (~~referenced property~~*pMonitoredValue*) = (a value x:
    (~~Setpoint_Reference~~*pSetpoint* - ~~Low_Diff_Limit~~*pLowDiffLimit* + ~~Deadband~~*pDeadband*) < x < (~~Setpoint_Reference~~*pSetpoint* + ~~High_Diff_Limit~~*pHighDiffLimit* – ~~Deadband~~*pDeadband*))
    ELSE
    MAKE (~~the referenced property~~*pMonitoredValue* have a value x:
    (~~Setpoint_Reference~~*pSetpoint* - ~~Low_Diff_Limit~~*pLowDiffLimit* + ~~Deadband~~*pDeadband*) < x < (~~Setpoint_Reference~~*pSetpoint* + ~~High_Diff_Limit~~*pHighDiffLimit* – ~~Deadband~~*pDeadband*))
18. WAIT (~~Time_Delay~~*pTimeDelayNormal*)
19. BEFORE **Notification Fail Time**
    RECEIVE ConfirmedEventNotification-Request,
    'Process Identifier' = (any valid process ID),
    'Initiating Device Identifier' = IUT,
    'Event Object Identifier' = (the object being tested),
    'Time Stamp' = (~~the current local time~~*any valid time stamp*),
    'Notification Class' = (the configured notification class),
    'Priority' = (the value configured to correspond to a TO-NORMAL transition),
    'Event Type' = FLOATING_LIMIT,
    'Notify Type' = EVENT | ALARM,
    'AckRequired' = TRUE | FALSE,
    'From State' = HIGH_LIMIT,
    'To State' = NORMAL,

'Event Values' = ~~reference-value~~*pMonitoredValue*, ~~Status_Flags~~*pStatusFlags*, ~~setpoint-value~~*pSetpoint*, ~~error-limit~~*pHighDiffLimit*~~,~~

20. TRANSMIT BACnet-SimpleACK-PDU
21. ~~IF (the object being tested is not an Event Enrollment object) THEN~~*IF (Protocol_Revision is present AND Protocol_Revision e 13) THEN*
    VERIFY ~~Status_Flags~~*pStatusFlags* = (FALSE, FALSE, ?, ?)
22. VERIFY ~~Event_State~~*pCurrentState* = NORMAL
23. IF (Protocol_Revision is present and Protocol_Revision ≥ 1) THEN
    VERIFY Event_Time_Stamps = (the timestamp in step 8, *, the timestamp in step 19)
24. IF (~~the referenced property~~*pMonitoredValue* is writable) THEN
    WRITE (~~referenced property~~*pMonitoredValue*) = (a value x:
    (~~Setpoint_Reference~~*pSetpoint* - ~~Low_Diff_Limit~~*pLowDiffLimit* < x < (~~Setpoint_Reference~~*pSetpoint* - ~~Low_Diff_Limit~~*pLowDiffLimit* + ~~Deadband~~*pDeadband*))
    ELSE
    MAKE (~~the referenced property~~*pMonitoredValue* have a value x:
    (~~Setpoint_Reference~~*pSetpoint* - ~~Low_Diff_Limit~~*pLowDiffLimit* < x < (~~Setpoint_Reference~~*pSetpoint* - ~~Low_Diff_Limit~~*pLowDiffLimit* + ~~Deadband~~*pDeadband*))
25. WAIT (~~Time_Delay~~*pTimeDelay* + **Notification Fail Time**)
26. CHECK (verify that no notification message has been transmitted)
27. VERIFY ~~Event_State~~*pCurrentState* = NORMAL
28. IF (~~the referenced property~~*pMonitoredValue* is writable) THEN
    WRITE (~~referenced property~~*pMonitoredValue*) = (a value x such x < (~~Setpoint_Reference~~*pSetpoint* - ~~Low_Diff_Limit~~*pLowDiffLimit*))
    ELSE
    MAKE (~~referenced property~~*pMonitoredValue* have a value x: x < (~~Setpoint_Reference~~*pSetpoint* - ~~Low_Diff_Limit~~*pLowDiffLimit*))
29. WAIT (~~Time_Delay~~*pTimeDelay*)
30. BEFORE **Notification Fail Time**
    RECEIVE ConfirmedEventNotification-Request,
    'Process Identifier' = (any valid process ID),
    'Initiating Device Identifier' = IUT,
    'Event Object Identifier' = (the object being tested),
    'Time Stamp' = (~~the current local time~~*any valid time stamp*),
    'Notification Class' = (the configured notification class),
    'Priority' = (the value configured to correspond to a TO-OFFNORMAL transition),
    'Event Type' = FLOATING_LIMIT,
    'Notify Type' = EVENT | ALARM,
    'AckRequired' = TRUE | FALSE,
    'From State' = NORMAL,
    'To State' = LOW_LIMIT,
    'Event Values' = ~~reference-value~~*pMonitoredValue*, ~~Status_Flags~~*pStatusFlags*, ~~setpoint-value~~*pSetpoint*, ~~error-limit~~*pLowDiffLimit*~~,~~
31. TRANSMIT BACnet-SimpleACK-PDU
32. ~~IF (the object being tested is not an Event Enrollment object) THEN~~*IF (Protocol_Revision is present AND Protocol_Revision e 13) THEN*
    VERIFY ~~Status_Flags~~*pStatusFlags* = (TRUE, FALSE, ?, ?)
33. VERIFY ~~Event_State~~*pCurrentState* = LOW_LIMIT
34. IF (Protocol_Revision is present and Protocol_Revision ≥ 1) THEN
    VERIFY Event_Time_Stamps = (the timestamp in step 30, *, the timestamp in step 19)
35. IF (~~the referenced property~~*pMonitoredValue* is writable) THEN
    WRITE (~~referenced property~~*pMonitoredValue*) = (a value x:
    (~~Setpoint_Reference~~*pSetpoint* - ~~Low_Limit~~*pLowDiffLimit*) < x < (~~Setpoint_Reference~~*pSetpoint* - ~~Low_Limit~~*pLowDiffLimit* + ~~Deadband~~*pDeadband*))
    ELSE
    MAKE (~~the referenced property~~*pMonitoredValue* have a value x:

(~~Setpoint_Reference~~*pSetpoint* - ~~Low_Limit~~*pLowDiffLimit*) < x < (~~Setpoint_Reference~~*pSetpoint* - ~~Low_Limit~~*pLowDiffLimit* + ~~Deadband~~*pDeadband*))

36. WAIT (~~Time_Delay~~*pTimeDelayNormal* + **Notification Fail Time**)
37. CHECK (verify that no notification message has been transmitted)
38. VERIFY ~~Event_State~~*pCurrentState* = ~~Low_Limit~~*pLowDiffLimit*
39. IF (~~the referenced property~~*pMonitoredValue* is writable) THEN
    WRITE (~~referenced property~~*pMonitoredValue*) = (a value x:
    (~~Setpoint_Reference~~*pSetpoint* - ~~Low_Limit~~*pLowDiffLimit* + ~~Deadband~~*pDeadband*) < x < (~~Setpoint_Reference~~*pSetpoint* + ~~High_Diff_Limit~~*pHighDiffLimit* – ~~Deadband~~*pDeadband*))
    ELSE
    MAKE (~~the eferenced property~~*pMonitoredValue* have a value x:
    (~~Setpoint_Reference~~*pSetpoint* - ~~Low_Limit~~*pLowDiffLimit* + ~~Deadband~~*pDeadband*) < x < (~~Setpoint_Reference~~*pSetpoint* + ~~High_Diff_Limit~~*pHighDiffLimit* – ~~Deadband~~*pDeadband*))
40. WAIT (~~Time_Delay~~*pTimeDelayNormal*)
41. BEFORE **Notification Fail Time**
    RECEIVE ConfirmedEventNotification-Request,
    'Process Identifier' = (any valid process ID),
    'Initiating Device Identifier' = IUT,
    'Event Object Identifier' = (the object being tested),
    'Time Stamp' = (~~the current local time~~*any valid time stamp*),
    'Notification Class' = (the configured notification class),
    'Priority' = (the value configured to correspond to a TO-NORMAL transition),
    'Event Type' = FLOATING_LIMIT,
    'Message Text' = *(optional, any valid message text),* 'Notify Type' = EVENT | ALARM,
    'AckRequired' = TRUE | FALSE,
    'From State' = LOW_LIMIT,
    'To State' = NORMAL,
    'Event Values' = ~~reference-value~~*pMonitoredValue*, ~~Status_Flags~~*pStatusFlags*, ~~setpoint-value~~*pSetpoint*, ~~error-limit~~*pLowDiffLimit*~~,~~
42. TRANSMIT BACnet-SimpleACK-PDU
43. ~~IF (the object being tested is not an Event Enrollment object) THEN~~*IF (Protocol_Revision is present AND Protocol_Revision e 13) THEN*
    VERIFY ~~Status_Flags~~*pStatusFlags* = (FALSE, FALSE, ?, ?)
44. VERIFY ~~Event_State~~*pCurrentState* = NORMAL
45. IF (Protocol_Revision is present and Protocol_Revision ≥ 1) THEN
    VERIFY Event_Time_Stamps = (the timestamp in step 30, *, the timestamp in step 41)

Notes to Tester: ~~The 'Message Text' parameter is omitted in the test description because it is optional. The IUT may include this parameter in the notification messages.~~ The time stamps indicated by "*" in steps 12, 23, 34 and 45 can have a value that indicates an unspecified time or a time that precedes the timestamp in step 8.

[Change **Clause 8.4.6**, p. 176]

8.4.6 OUT_OF_RANGE Test~~s~~ *(ConfirmedEventNotification)*

~~Dependencies: ReadProperty Service Execution Tests, 9.18; WriteProperty Service Execution Tests, 9.22.~~

~~BACnet Reference Clauses: 12.2, 12.3, 12.4, 12.12, 12.23, 13.2, 13.3.6, and 13.8.~~

Purpose: To verify the correct operation of the OUT_OF_RANGE event algorithm.

Test Concept: The object begins the test in a NORMAL state. *pMonitoredValue*~~The Present_Value (referenced property)~~ is raised to a value that is below but within *p*Deadband of the *pHighLimit*~~high limit~~. At this point the object should still be in a NORMAL state. *pMonitoredValue*~~The Present_Value (referenced property)~~ is raised to a value that is above *pHighLimit*~~the~~

~~high limit~~. After the time delay expires, the object should enter the HIGH_LIMIT state and transmit an event notification message. *pMonitoredValue*~~The Present_Value (referenced property)~~ is lowered to a value that is below the ~~high limit~~*pHighLimit* but still within *p*Deadband of the limit. The object should remain in the HIGH_LIMIT state. *pMonitoredValue*~~The Present_Value (referenced property)~~ is lowered further to a normal value that is not within *p*Deadband of a limit. After the time delay expires, the object should enter the NORMAL state and issue an event notification. The same process is repeated to test the low limit.

Configuration Requirements: The IUT shall be configured such that the Event_Enable property has a value of TRUE for the TO-OFFNORMAL and TO-NORMAL transitions. For objects using intrinsic reporting the Limit_Enable property shall have a value of TRUE for both ~~HighLimit and LowLimit~~*HIGH_LIMIT and LOW_LIMIT* events. The ~~Issue_Confirmed_Notifications property~~*'Issue Confirmed Notifications' parameter* shall have a value of TRUE. The event-generating object~~s~~ shall be in a NORMAL state at the start of the test.

~~In the test description below Present_Value is used as the referenced property. If an Event Enrollment object is being tested Present_Value should be replaced by the appropriate property reference.~~

Test Steps:

1. VERIFY ~~Event_State~~*pCurrentState* = NORMAL
2. IF (~~Present_Value~~*pMonitoredValue* is writable) THEN
   WRITE ~~Present_Value~~*pMonitoredValue* = (a value x: (~~High_Limit~~ *pHighLimit* – ~~Deadband~~*pDeadband*) < x < ~~High_Limit~~*pHighLimit*)
   ELSE
   MAKE (~~Present_Value~~*pMonitoredValue* have a value x: (~~High_Limit~~ *pHighLimit* – ~~Deadband~~*pDeadband*) < x < ~~High_Limit~~*pHighLimit*)
3. WAIT (~~Time_Delay~~*pTimeDelay* + **Notification Fail Time**)
4. CHECK (verify that no notification message has been transmitted)
5. VERIFY ~~Event_State~~*pCurrentState* = NORMAL
6. IF (~~Present_Value~~*pMonitoredValue* is writable) THEN
   WRITE ~~Present_Value~~*pMonitoredValue* = (a value x such x > ~~High_Limit~~ *p HighLimit*)
   ELSE
   MAKE (~~Present_Value~~*pMonitoredValue* have a value x: x > ~~High_Limit~~*pHighLimit*)
7. WAIT (~~Time_Delay~~*pTimeDelay*)
8. BEFORE **Notification Fail Time**
   RECEIVE ConfirmedEventNotification-Request,

'Process Identifier' =	(any valid process ID),
'Initiating Device Identifier' =	IUT,
'Event Object Identifier' =	(the object being tested),
'Time Stamp' =	(~~the current local time~~*any valid time stamp*),
'Notification Class' =	(the configured notification class),
'Priority' =	(the value configured to correspond to a TO-OFFNORMAL transition),
'Event Type' =	OUT_OF_RANGE,
'Message Text' =	*(optional, any valid message text),*
'Notify Type' =	EVENT | ALARM,
'AckRequired' =	TRUE | FALSE,
'From State' =	NORMAL,
'To State' =	HIGH_LIMIT,
'Event Values' =	~~Present_Value~~*pMonitoredValue*, ~~Status_Flags~~*pStatusFlags*, ~~Deadband~~*pDeadband*, ~~High_Limit~~*pHighLimit*

9. TRANSMIT BACnet-SimpleACK-PDU
10. ~~IF (the object being tested is not an Event Enrollment object) THEN~~*IF (Protocol_Revision is present AND Protocol_Revision e 13) THEN*
    VERIFY ~~Status_Flags~~*pStatusFlags* = (TRUE, FALSE, ?, ?)
11. VERIFY ~~Event_State~~*pCurrentState* = HIGH_LIMIT
12. IF (Protocol_Revision is present and Protocol_Revision ≥ 1) THEN
    VERIFY Event_Time_Stamps = (the timestamp in step 8, *, *)

13. IF (~~Present_Value~~*pMonitoredValue* is writable) THEN
    WRITE ~~Present_Value~~*pMonitoredValue* = (a value x: (~~High_Limit~~*pHighLimit* – ~~Deadband~~*pDeadband*)< x < ~~High_Limit~~*pHighLimit*)
    ELSE
    MAKE (~~Present_Value~~*pMonitoredValue* have a value x: (~~High_Limit~~*pHighLimit* – ~~Deadband~~*pDeadband*)< x < ~~High_Limit~~*pHighLimit*)
14. WAIT (~~Time_Delay~~*pTimeDelayNormal* + **Notification Fail Time**)
15. CHECK (verify that no notification message has been transmitted)
16. VERIFY ~~Event_State~~*pCurrentState* = HIGH_LIMIT
17. IF (~~Present_Value~~*pMonitoredValue* is writable) THEN
    WRITE ~~Present_Value~~*pMonitoredValue* = (a value x: (~~Low_Limit~~*pLowDiffLimit* + ~~Deadband~~*pDeadband*) < x < (~~High_Limit~~p*HighLimit* – ~~Deadband~~*pDeadband*))
    ELSE
    MAKE (~~Present_Value~~*pMonitoredValue* have a value x: (~~Low_Limit~~*pLowDiffLimit* + ~~Deadband~~*pDeadband*) < x < (~~High_Limit~~*pHighLimit* – ~~Deadband~~*pDeadband*))
18. WAIT (~~Time_Delay~~*pTimeDelayNormal*)
19. BEFORE **Notification Fail Time**
    RECEIVE ConfirmedEventNotification-Request,
    'Process Identifier' = (any valid process ID),
    'Initiating Device Identifier' = IUT,
    'Event Object Identifier' = (the object being tested),
    'Time Stamp' = (~~the current local time~~*any valid time stamp*),
    'Notification Class' = (the configured notification class),
    'Priority' = (the value configured to correspond to a TO-NORMAL transition),
    'Event Type' = OUT_OF_RANGE,
    'Message Text' = *(optional, any valid message text),*
    'Notify Type' = EVENT | ALARM,
    'AckRequired' = TRUE | FALSE,
    'From State' = HIGH_LIMIT,
    'To State' = NORMAL,
    'Event Values' = ~~Present_Value~~*pMonitoredValue*, ~~Status_Flags~~*pStatusFlags*, ~~Deadband~~*pDeadband*, ~~High_Limit~~*pHighLimit*
20. TRANSMIT BACnet-SimpleACK-PDU
21. ~~IF (the object being tested is not an Event Enrollment object) THEN~~*IF (Protocol_Revision is present AND Protocol_Revision e 13) THEN*
    VERIFY ~~Status_Flags~~*pStatusFlags* = (FALSE, FALSE, ?, ?)
22. VERIFY ~~Event_State~~*pCurrentState* = NORMAL
23. IF (Protocol_Revision is present and Protocol_Revision ≥ 1) THEN
    VERIFY Event_Time_Stamps = (the timestamp in step 8, *, the timestamp in step 19)
24. IF (~~Present_Value~~*pMonitoredValue* is writable) THEN
    WRITE ~~Present_Value~~*pMonitoredValue* = (a value x: ~~Low_Limit~~*pLowDiffLimit* < x < (~~Low_Limit~~*pLowDiffLimit* + ~~Deadband~~*pDeadband*))
    ELSE
    MAKE (~~Present_Value~~*pMonitoredValue* have a value x: ~~Low_Limit~~*pLowDiffLimit* < x < (~~Low_Limit~~*pLowDiffLimit* + ~~Deadband~~*pDeadband*))
25. WAIT (~~Time_Delay~~*pTimeDelay* + **Notification Fail Time**)
26. CHECK (verify that no notification message has been transmitted)
27. VERIFY ~~Event_State~~*pCurrentState* = NORMAL
28. IF (~~Present_Value~~*pMonitoredValue* is writable) THEN
    WRITE ~~Present_Value~~*pMonitoredValue* = (a value x such x < ~~Low_Limit~~*pLowDiffLimit*)
    ELSE
    MAKE (~~Present_Value~~*pMonitoredValue* have a value x: x < ~~Low_Limit~~*pLowDiffLimit*)
29. WAIT (~~Time_Delay~~*pTimeDelay*)
30. BEFORE **Notification Fail Time**
    RECEIVE ConfirmedEventNotification-Request,
    'Process Identifier' = (any valid process ID),

'Initiating Device Identifier' = IUT,
'Event Object Identifier' = (the object being tested),
'Time Stamp' = ~~(the current local time)~~*(any valid time stamp)*,
'Notification Class' = (the configured notification class),
'Priority' = (the value configured to correspond to a TO-OFFNORMAL transition),
'Event Type' = OUT_OF_RANGE,
'Message Text' = *(optional, any valid message text),*
'Notify Type' = EVENT | ALARM,
'AckRequired' = TRUE | FALSE,
'From State' = NORMAL,
'To State' = LOW_LIMIT,
'Event Values' = ~~Present_Value~~*pMonitoredValue*, ~~Status_Flags~~*pStatusFlags*, ~~Deadband~~*pDeadband*, ~~Low_Limit~~*pLowDiffLimit*

31. TRANSMIT BACnet-SimpleACK-PDU
32. ~~IF (the object being tested is not an Event Enrollment object) THEN~~*IF (Protocol_Revision is present AND Protocol_Revision e 13) THEN*
    VERIFY ~~Status_Flags~~*pStatusFlags* = (TRUE, FALSE, ?, ?)
33. VERIFY ~~Event_State~~*pCurrentState* = LOW_LIMIT
34. IF (Protocol_Revision is present and Protocol_Revision ≥ 1) THEN
    VERIFY Event_Time_Stamps = (the timestamp in step 30, *, the timestamp in step 19)
35. IF (~~Present_Value~~*pMonitoredValue* is writable) THEN
    WRITE ~~Present_Value~~*pMonitoredValue* = (a value x:~~ Low_Limit~~*pLowDiffLimit* < x < (~~Low_Limit~~*pLowDiffLimit* + ~~Deadband~~*pDeadband*))
    ELSE
    MAKE (~~Present_Value~~*pMonitoredValue* have a value x: ~~Low_Limit~~*pLowDiffLimit* < x < (~~Low_Limit~~*pLowDiffLimit* + ~~Deadband~~*pDeadband*))
36. WAIT (~~Time_Delay~~*pTimeDelayNormal* + **Notification Fail Time**)
37. CHECK (verify that no notification message has been transmitted)
38. VERIFY ~~Event_State~~*pCurrentState* = LOW_LIMIT
39. IF (~~Present_Value~~*pMonitoredValue* is writable) THEN
    WRITE ~~Present_Value~~*pMonitoredValue* = (a value x: (~~Low_Limit~~*pLowDiffLimit* + ~~Deadband~~*pDeadband*) < x < (~~High_Limit~~p*HighLimit* – ~~Deadband~~*pDeadband*))
    ELSE
    MAKE (~~Present_Value~~*pMonitoredValue* have a value x: (~~Low_Limit~~*pLowDiffLimit* + ~~Deadband~~*pDeadband*) < x < (~~High_Limit~~ *pHighLimit* – ~~Deadband~~*pDeadband*))
40. WAIT (~~Time_Delay~~*pTimeDelayNormal*)
41. BEFORE **Notification Fail Time**
    RECEIVE ConfirmedEventNotification-Request,
    'Process Identifier' = (any valid process ID),
    'Initiating Device Identifier' = IUT,
    'Event Object Identifier' = (the object being tested),
    'Time Stamp' = (~~the current local time~~*any valid time stamp*),
    'Notification Class' = (the configured notification class),
    'Priority' = (the value configured to correspond to a TO-NORMAL transition),
    'Event Type' = OUT_OF_RANGE,
    'Message Text' = *(optional, any valid message text),*
    'Notify Type' = EVENT | ALARM,
    'AckRequired' = TRUE | FALSE,
    'From State' = LOW_LIMIT,
    'To State' = NORMAL,
    'Event Values' = ~~Present_Value~~*pMonitoredValue*, ~~Status_Flags~~*pStatusFlags*, ~~Deadband~~*pDeadband*, ~~Low_Limit~~*pLowDiffLimit*
42. TRANSMIT BACnet-SimpleACK-PDU
43. ~~IF (the object being tested is not an Event Enrollment object) THEN~~*IF (Protocol_Revision is present AND Protocol_Revision e 13) THEN*
    VERIFY ~~Status_Flags~~*pStatusFlags* = (FALSE, FALSE, ?, ?)

44. VERIFY ~~Event_State~~*pCurrentState* = NORMAL
45. IF (Protocol_Revision is present and Protocol_Revision ≥ 1) THEN
    VERIFY Event_Time_Stamps = (the timestamp in step 30, *, the timestamp in step 41)

Notes to Tester: ~~The 'Message Text' parameter is omitted in the test description because it is optional. The IUT may include this parameter in the notification messages.~~ The time stamps indicated by "*" in steps 12, 23, 34 and 45 can have a value that indicates an unspecified time or a time that precedes the timestamp in step 8.

[Change **Clause 8.4.7**, p. 179]

8.4.7 BUFFER_READY Test~~s~~ *(ConfirmedEventNotification)*

~~Dependencies: ReadProperty Service Execution Tests, 9.18.~~

~~BACnet Reference Clauses: 12.12, 12.25, 13.2, 13.3.7, and 13.8.~~

Purpose: To verify the correct operation of the BUFFER_READY event algorithm. ~~This test applies to logging objects that support intrinsic reporting and to Event Enrollment objects with an Event_Type of BUFFER_READY.~~

Test Concept: The object that performs the notification ("the event initiating object") begins the test in a NORMAL state. The object containing the buffer ("the buffer object") acquires enough records to generate a notification, at which time, the notifying object performs a TO-NORMAL transition and sends a BUFFER_READY notification.

Configuration Requirements: The IUT shall be configured such that the Event_Enable property has a value of TRUE for the TO-NORMAL transition. The event initiating object shall be in a NORMAL state at the start of the test. The 'Issue Confirmed Notifications' parameter of the element of the Notification Class' Recipient_List property referring to the TD shall be set to TRUE.

Test Steps:

1.– VERIFY ~~Event_State~~*pCurrentState* = NORMAL
2.– MAKE (buffer object collect enough records to generate a notification.)
3.– RECEIVE ConfirmedEventNotification-Request,

'Process Identifier' =	(any valid process ID),
'Initiating Device Identifier' =	IUT,
'Event Object Identifier' =	(the intrinsic reporting object being tested or the Event Enrollment object being tested),
'Time Stamp' =	(~~any appropriate BACnetTimeStamp value~~*any valid time stamp*),
'Notification Class' =	(the configured notification class),
'Priority' =	(the value configured to correspond to a TO-NORMAL transition),
'Event Type' =	BUFFER_READY,
'Message Text' =	*(optional, any valid message text),*
'Notify Type' =	EVENT | ALARM,
'AckRequired' =	TRUE | FALSE,
'From State' =	NORMAL,
'To State' =	NORMAL,
'Event Values' =	~~'Buffer Property': reference to the Log_Buffer property)~~*pLogBuffer*, (~~'Previous Notification'~~*pPreviousCount*: any valid value), (~~'Current Notification'~~*pMonitoredValue*: any valid value CN1)

4.– TRANSMIT BACnet-SimpleACK-PDU
5.– MAKE (logging object collect the number of records specified by Notification_Threshold)
6.– RECEIVE ConfirmedEventNotification-Request,

'Process Identifier' =	(any valid process ID),
'Initiating Device Identifier' =	IUT,
'Event Object Identifier' =	(the intrinsic reporting object being tested or the Event Enrollment object being

tested),
'Time Stamp' = (~~any appropriate BACnetTimeStamp value~~*any valid time stamp*),
'Notification Class' = (the configured notification class),
'Priority' = (the value configured to correspond to a TO-NORMAL transition),
'Event Type' = BUFFER_READY,
'Message Text' = *(optional, any valid message text),*
'Notify Type' = EVENT | ALARM,
'AckRequired' = TRUE | FALSE,
'From State' = NORMAL,
'To State' = NORMAL,
'Event Values' = (~~'Buffer Property': reference to the Log_Buffer property)~~*pLogBuffer*,
(~~'Previous Notification'~~*pPreviousCount*: CN1),
(~~'Current Notification'~~*pMonitoredValue*: CN1 + Notification_Threshold)

7. TRANSMIT BACnet-SimpleACK-PDU

[Change **Clause 8.4.8.1**, p. 180]

8.4.8.1 NORMAL to OFFNORMAL Transition Test

~~Dependencies: ReadProperty Service Execution Tests, 9.18; WriteProperty Service Execution Tests, 9.22.~~

~~BACnet Reference Clauses: 12.12.7, 12.15, 12.16, 13.3.8, 13.8 and Figure 13-9.~~

Purpose: This test case verifies the correct operation of the CHANGE_OF_LIFE_SAFETY event algorithm for objects transitioning ~~between~~ *from* NORMAL ~~and~~*to* OFFNORMAL event states. ~~It applies to Event Enrollment objects with an Event_Type of CHANGE_OF_LIFE_SAFETY and to intrinsic event reporting for Life Safety Point and Life Safety Zone objects~~.

Test Concept: The object begins the test in a NORMAL state. The *pMonitoredValue parameter*~~Present_Value (referenced property)~~ is changed to one of the values designated in ~~List_Of_Alarm_Values~~ *pAlarmValues*. After the time delay expires, the object should enter the OFFNORMAL state and transmit an event notification message.

Configuration Requirements: The IUT shall be configured such that the Event_Enable property has a value of TRUE for the TO-OFFNORMAL~~, TO-FAULT~~ and TO-NORMAL transitions. The ~~Issue_Confirmed_Notifications property~~*'Issue Confirmed Notifications' parameter* shall have a value of TRUE. The event-generating objects shall be in a NORMAL state at the start of the test.
~~In the test description below Present_Value is used as the referenced property. If an Event Enrollment object is being tested, Present_Value should be replaced by the appropriate property reference.~~

Test Steps:

1. VERIFY Event_Detection_Enable = TRUE
~~1~~ *2.* VERIFY ~~Event_State~~ *pCurrentState* = NORMAL
~~2~~ *3.* MAKE (~~Present_Value~~ *pMonitoredValue* have a value ~~x, x :one of the Alarm_Values~~*from the pAlarmValues list*)
~~3.~~ *4.* WAIT *pTimeDelay*~~TimeDelay~~
~~4.~~ *5.* BEFORE **Notification Fail Time**
~~5.~~ RECEIVE ConfirmedEventNotification-Request,
'Process Identifier' = (any valid process ID),
'Initiating Device Identifier' = IUT,
'Event Object Identifier' = (the intrinsic reporting object being tested or the Event Enrollment Object being tested),
'Time Stamp' = (*T1: any valid time stamp* ~~the current local time~~),
'Notification Class' = (the configured notification class),
'Priority' = (the value configured to correspond to a TO-OFFNORMAL transition),
'Event Type' = CHANGE_OF_LIFE_SAFETY,

'Message Text' = *(S1, optional: any valid message text),*
'Notify Type' = EVENT | ALARM,
~~-~~'AckRequired' = TRUE | FALSE,
'From State' = NORMAL,
'To State' = ~~—~~ OFFNORMAL,
'Event Values' = ~~Present_Value, Mode, Status_Flags, Operation_Expected~~ *pMonitoredValue,pMode,pStatusFlags, pOperationExpected*

6. TRANSMIT BACnet-SimpleACK-PDU
7. *IF (Protocol_Revision is present AND Protocol_Revision e 13) THEN*
   VERIFY ~~Status_Flags~~*pStatusFlags* = (TRUE, FALSE, ?, ?)
8. VERIFY ~~Event_State~~*pCurrentState* = OFFNORMAL
9. *IF (Protocol_Revision is present AND Protocol_Revision ≥ 1) THEN*
   VERIFY Event_Time_Stamps = (*T1* ~~the timestamp in step 5~~, *, *)
10. *VERIFY Event_Message_Texts = (S1, *, *)*

[Change **Clause 8.4.8.2**, p. 181]

8.4.8.2 OFFNORMAL to NORMAL Transition Test

~~Dependencies: ReadProperty Service Execution Tests, 9.18; WriteProperty Service Execution Tests, 9.22.~~

~~BACnet Reference Clauses: 12.12.7, 12.15, 12.16, 13.3.8, 13.8 and Figure 13-9.~~

Purpose: This test case verifies the correct operation of the CHANGE_OF_LIFE_SAFETY event algorithm for objects transitioning ~~between~~ *from* OFFNORMAL ~~and~~*to* NORMAL event states. ~~It applies to Event Enrollment objects with an Event_Type of CHANGE_OF_LIFE_SAFETY and to intrinsic event reporting for Life Safety Point and Life Safety Zone objects.~~

Test Concept: The object begins the test in an OFFNORMAL state. The *pMonitoredValue* ~~Present_Value (referenced property)~~ is made to be in the NORMAL state. If latching is supported, the object should remain in the OFFNORMAL state until object is reset. This test needs to be repeated for all reset operations supported. If latching is not supported, then after the time delay expires, the object should enter the NORMAL state and transmit an event notification message. It is expected that the TD will often be able to reset the object by using the LifeSafetyOperation service with a reset or reset-alarm request.

Configuration Requirements: The IUT shall be configured such that the Event_Enable property has a value of TRUE for the TO-OFFNORMAL~~, TO-FAULT~~ and TO-NORMAL transitions. The ~~Issue_Confirmed_Notifications property~~*'Issue Confirmed Notifications' parameter* shall have a value of TRUE. The event-generating objects shall be in a OFFNORMAL state at the start of the test. ~~. In the test description below Present_Value is used as the referenced property. If an Event Enrollment object is being tested, Present_Value should be replaced by the appropriate property reference.~~

Test Steps:

1. VERIFY Event_Detection_Enable = TRUE
~~1.~~ *2.* MAKE ~~Present_Value~~ *pMonitoredValue* = NORMAL state
~~2.~~ *3.* WAIT *pTimeDelay* ~~TimeDelay~~
~~3.~~ *4.* IF (latching is supported) THEN *{*
~~4.~~ *5.* CHECK (~~Event_State~~ *pCurrentState* = OFFNORMAL)
~~5.~~ *6.* MAKE (the object reset)
~~6.~~ *7.* BEFORE **Notification Fail Time**
~~7.~~ RECEIVE ConfirmedEventNotification-Request,
'Process Identifier' = (any valid process ID),
'Initiating Device Identifier' = IUT,

	'Event Object Identifier' =	(the intrinsic reporting object being tested or the Event Enrollment object being tested),
	'Time Stamp' =	(*T1: any valid time stamp* ~~the current local time~~),
	'Notification Class' =	(the configured notification class),
	'Priority' =	(the value configured to correspond to a TO-OFFNORMAL transition),
	'Event Type' =	CHANGE_OF_LIFE_SAFETY,
	'Message Text' =	*(S1, optional: any valid message text),*
	'Notify Type' =	EVENT | ALARM,
	'AckRequired' =	TRUE | FALSE,
	'From State' =	OFFNORMAL,
	'To State' =	NORMAL,
	~~'~~'Event Values' =	~~Present_Value, Mode, Status_Flags, Operation_Expected~~ *pMonitoredValue, pMode, pStatusFlags, pOperationExpected*

8. TRANSMIT BACnet-SimpleACK-PDU,
9. *IF (Protocol_Revision is present AND Protocol_Revision e 13) THEN*
   VERIFY ~~Status_Flags~~*pStatusFlags* = (FALSE, FALSE, ?, ?),
10. VERIFY ~~Event_State~~*pCurrentState* = NORMAL,
11. *IF (Protocol_Revision is present AND Protocol_Revision ≥ 1) THEN*
    VERIFY Event_Time_Stamps = (~~the timestamp in step 7, *, *~~)*(*,*,T1*),
12. *VERIFY Event_Message_Texts = (*, *, S1)*

}
ELSE *{*

~~12.~~ *13.* BEFORE **Notification Fail Time**

~~13.~~ RECEIVE ConfirmedEventNotification-Request,

	'Process Identifier' =	(any valid process ID),
	'Initiating Device Identifier' =	IUT,
	'Event Object Identifier' =	(the intrinsic reporting object being tested or the Event Enrollment object being tested),
	'Time Stamp' =	(*T2: any valid time stamp* ~~the current local time~~),
	'Notification Class' =	(the configured notification class),
	*'*Priority' =	(the value configured to correspond to a TO-OFFNORMAL transition),
	'Event Type' =	CHANGE_OF_LIFE_SAFETY,
	'Message Text' =	*(S2, optional: any valid message text),*
	'Notify Type' =	EVENT | ALARM,
	'AckRequired' =	TRUE | FALSE,
	~~'~~'From State' =	OFFNORMAL,
	'To State' =	NORMAL,
	'Event Values' =	~~Present_Value, Mode, Status_Flags, Operation_Expected~~*pMonitoredValue, pMode, pStatusFlags, pOperationExpected*

14. TRANSMIT BACnet-SimpleACK-PDU
15. *IF (Protocol_Revision is present AND Protocol_Revision e 13) THEN*
    VERIFY ~~Status_Flags~~*pStatusFlags* = (FALSE, FALSE, ?, ?)
16. VERIFY ~~Event_State~~*pCurrentState* = NORMAL
17. *IF (Protocol_Revision is present AND Protocol_Revision ≥ 1) THEN*
    VERIFY Event_Time_Stamps = (~~the timestamp in step 13, *, *~~)*(*, *, T2)*
18. *VERIFY Event_Message_Texts = (*, *, S2)*

}

[Change **Clause 8.4.8.3**, p. 182]

8.4.8.3 NORMAL to LIFE_SAFETY_ALARM Transition Test

~~Dependencies: ReadProperty Service Execution Tests, 9.18; WriteProperty Service Execution Tests, 9.22.~~

~~BACnet Reference Clauses: 12.12.7, 12.15, 12.16, 13.3.8, 13.13 and Figure 13.9.~~

Purpose: This test case verifies the correct operation of the CHANGE_OF_LIFE_SAFETY event algorithm for objects transitioning ~~between~~ *from* NORMAL ~~and~~*to* LIFE_SAFETY_ALARM event states. ~~It applies to Event Enrollment objects with an Event_Type of CHANGE_OF_LIFE_SAFETY and to intrinsic event reporting for Life Safety Point and Life Safety Zone objects.~~

Test Concept: The object begins the test in a NORMAL state. The The *pMonitoredValue* ~~Present_Value (referenced property)~~is changed to one of the values designated in ~~Life_Safety_Alarm_Values~~*pLifeSafetyAlarmValues*. After the time delay expires, the object should enter the LIFE_SAFETY_ALARM state and transmit an event notification message.

Configuration Requirements: The IUT shall be configured such that the Event_Enable property has a value of TRUE for the TO-OFFNORMAL~~, TO-FAULT~~ and TO-NORMAL transitions. The ~~Issue_Confirmed_Notifications property~~*'Issue Confirmed Notifications' parameter* shall have a value of TRUE. The event-generating objects shall be in a NORMAL state at the start of the test. ~~. In the test description below Present_Value is used as the referenced property. If an Event Enrollment object is being tested, Present_Value should be replaced by the appropriate property reference.~~

Test Steps:

1. VERIFY Event_Detection_Enable = TRUE

~~1.~~ *2.* VERIFY ~~Event_State~~*pCurrentState* = NORMAL

~~2.~~ *3.* MAKE (~~Present_Value~~ *pMonitoredValue* have a value ~~x such that x corresponds to a LIFE_SAFETY_ALARM state~~ *from the pLifeSafetyAlarmValues list*)

~~3.~~ *4.* WAIT *pTimeDelay* ~~TimeDelay~~

~~4.~~ *5.* BEFORE **Notification Fail Time**

~~5.~~ RECEIVE ConfirmedEventNotification-Request,

'Process Identifier' =	(any valid process ID),
'Initiating Device Identifier' =	IUT,
~~'~~'Event Object Identifier' =	(the intrinsic reporting object being tested or the Event Enrollment object being tested),
'Time Stamp' =	(*T1: any valid time stamp* ~~the current local time~~),
'Notification Class' =	(the configured notification class),
'Priority' =	(the value configured to correspond to a TO-OFFNORMAL transition),
'Event Type' =	CHANGE_OF_LIFE_SAFETY,
'Message Text' =	*(S1, optional: any valid message text),*
'Notify Type' =	EVENT | ALARM,
'AckRequired' =	TRUE | FALSE,
'From State' =	NORMAL,
'To State' =	LIFE_SAFETY_ALARM,
'Event Values' =	~~Present_Value, Mode, Status_Flags, Operation_Expected~~*pMonitoredValue, pMode, pStatusFlags, pOperationExpected*

6. TRANSMIT BACnet-SimpleACK-PDU

7. *IF (Protocol_Revision is present AND Protocol_Revision e 13) THEN*
   VERIFY ~~Status_Flags~~*pStatusFlags* = (TRUE, FALSE, ?, ?)

8. VERIFY ~~Event_State~~*pCurrentState* = LIFE_SAFETY_ALARM

9. *IF (Protocol_Revision is present AND Protocol_Revision ≥ 1) THEN*
   VERIFY Event_Time_Stamps = (~~the timestamp in step 5~~*T1*, *, *)

10. *VERIFY Event_Message_Texts = (S1, *, *)*

[Change **Clause 8.4.8.4**, p. 183]

8.4.8.4 LIFE_SAFETY_ALARM to NORMAL Transition Test

~~Dependencies: ReadProperty Service Execution Tests, 9.18; WriteProperty Service Execution Tests, 9.22.~~

~~BACnet Reference Clauses: 12.12.7, 12.15, 12.16, 13.3.8, 13.13 and Figure 13.9.~~

Purpose: This test case verifies the correct operation of the CHANGE_OF_LIFE_SAFETY event algorithm for objects transitioning ~~between~~ *from* LIFE_SAFETY_ALARM ~~and~~*to* NORMAL event states. ~~It applies to Event Enrollment objects with an Event_Type of CHANGE_OF_LIFE_SAFETY and to intrinsic event reporting for Life Safety Point and Life Safety Zone objects.~~

Test Concept: The object begins the test in a~~n~~ LIFE_SAFETY_ALARM state. The *pMonitoredValue* ~~Present_Value (referenced property)~~ is made to be in the NORMAL state. If latching is supported, the object should remain in the LIFE_SAFETY_ALARM state until the LifeSafetyOperation service has been executed to reset the object. This test needs to be repeated for all reset operations supported. If latching is not supported, then after the time delay expires, the object should enter the NORMAL state and transmit an event notification message.

Configuration Requirements: The IUT shall be configured such that the Event_Enable property has a value of TRUE for the TO-OFFNORMAL~~, TO-FAULT~~ and TO-NORMAL transitions. The ~~Issue_Confirmed_Notifications property~~*'Issue Confirmed Notifications' parameter* shall have a value of TRUE. The event-generating objects shall be in a LIFE_SAFETY_ALARM state at the start of the test. ~~. In the test description below Present_Value is used as the referenced property. If an Event Enrollment object is being tested Present_Value should be replaced by the appropriate property reference.~~

Test Steps:

1. VERIFY Event_Detection_Enable = TRUE

~~1.~~ *2.* MAKE (~~Present_Value~~*pMonitoredValue* have a value ~~x such~~ that ~~x~~ corresponds to a NORMAL state)

~~2.~~ *3.* WAIT *pTimeDelay* ~~TimeDelay~~

~~3.~~ *4.* IF (latching is supported) THEN *{*

~~4.~~ *5.* CHECK ~~Event_State~~ *pCurrentState* = LIFE_SAFETY_ALARM

~~5.~~ *6.* MAKE (the object reset)

~~6.~~ *7.* BEFORE **Notification Fail Time**

~~7.~~ RECEIVE ConfirmedEventNotification-Request,

'Process Identifier' =	(any valid process ID),
'Initiating Device Identifier' =	IUT,
'Event Object Identifier' =	(the intrinsic reporting object being tested or the Event Enrollment object being tested),
'Time Stamp' =	(*T1: any valid time stamp* ~~the current local time~~),
'Notification Class' =	(the configured notification class),
'Priority' =	(the value configured to correspond to a TO-OFFNORMAL transition),
'Event Type' =	CHANGE_OF_LIFE_SAFETY,
'Message Text' =	*(S1, optional: any valid message text),*
'Notify Type' =	EVENT | ALARM,
'AckRequired' =	TRUE | FALSE,
'From State' =	LIFE_SAFETY_ALARM,
'To State' =	NORMAL,
'Event Values' =	~~Present_Value, Mode, Status_Flags, Operation_Expected~~*pMonitoredValue, pMode, pStatusFlags, pOperationExpected*

8. TRANSMIT BACnet-SimpleACK-PDU

9. *IF (Protocol_Revision is present AND Protocol_Revision e 13) THEN*
   VERIFY ~~Status_Flags~~ *pStatusFlags* = (FALSE, FALSE, ?, ?)

10. VERIFY ~~Event_State~~ *pCurrentState* = NORMAL

11. *IF (Protocol_Revision is present AND Protocol_Revision ≥ 1) THEN*
    VERIFY Event_Time_Stamps = (~~the timestamp in step 7,*,*~~)*(*, *, T1)*,

12. VERIFY Event_Message_Texts = (, *, S1)*

}
ELSE *{*
~~12.~~ *13.* BEFORE Notification Fail Time
~~13.~~ RECEIVE ConfirmedEventNotification-Request,

'Process Identifier' =	(any valid process ID),
'Initiating Device Identifier' =	IUT,
~~'~~'Event Object Identifier' =	(the intrinsic reporting object being tested or the Event Enrollment object being tested),
'Time Stamp' =	(*T2: any valid time stamp* ~~the current local time~~),
'Notification Class' =	(the configured notification class),
'Priority' =	(the value configured to correspond to a TO-OFFNORMAL transition),
'Event Type' =	CHANGE_OF_LIFE_SAFETY,
'Message Text' =	*(S1, optional: any valid message text),*
'Notify Type' =	EVENT | ALARM,
'AckRequired' =	TRUE | FALSE,
'From State' =	LIFE_SAFETY_ALARM,
'To State' =	NORMAL,
'Event Values' =	~~Present_Value, Mode, Status_Flags, Operation_Expected~~*pMonitoredValue, pMode, pStatusFlags, pOperationExpected*

14. TRANSMIT BACnet-SimpleACK-PDU
15. *IF (Protocol_Revision is present AND Protocol_Revision e 13) THEN*
    VERIFY ~~Status_Flags~~ *pStatusFlags* = (FALSE, FALSE, ?, ?)
16. VERIFY ~~Event_State~~*pCurrentState* = NORMAL
17. *IF (Protocol_Revision is present AND Protocol_Revision ≥ 1) THEN*
    VERIFY Event_Time_Stamps = (~~the timestamp in step 13, *, *~~)*(*, *,T2)*
18. *VERIFY Event_Message_Texts = (*, *, S2)*

}

[Change **Clause 8.4.8.5**, p. 184]

8.4.8.5 LIFE_SAFETY_ALARM to OFFNORMAL Transition Test

~~Dependencies: ReadProperty Service Execution Tests, 9.18; WriteProperty Service Execution Tests, 9.22.~~

~~BACnet Reference Clauses: 12.12.7, 12.15, 12.16, 13.3.8, 13.8 and Figure 13-9.~~

Purpose: This test case verifies the correct operation of the CHANGE_OF_LIFE_SAFETY event algorithm for objects transitioning ~~between~~ *from* LIFE_SAFETY_ALARM ~~and~~*to* OFFNORMAL event states. ~~It applies to Event Enrollment objects with an Event_Type of CHANGE_OF_LIFE_SAFETY and to intrinsic event reporting for Life Safety Point and Life Safety Zone objects.~~

Test Concept: The object begins the test in a~~n~~ LIFE_SAFETY_ALARM state. ~~The The~~ *pMonitoredValue* ~~Present_Value (referenced property)~~ is made to be in the OFFNORMAL state. If latching is supported, the object should remain in the LIFE_SAFETY_ALARM state until the LifeSafetyOperation service has been executed to reset the object. This test needs to be repeated for all reset operations supported. If latching is not supported, then after the time delay expires, the object should enter the OFFNORMAL state and transmit an event notification message.

Configuration Requirements: The IUT shall be configured such that the Event_Enable property has a value of TRUE for the TO-OFFNORMAL~~, TO-FAULT~~ and TO-NORMAL transitions. The ~~Issue_Confirmed_Notifications property~~*'Issue Confirmed Notifications' parameter* shall have a value of TRUE. The event-generating objects shall be in a LIFE_SAFETY_ALARM state at the start of the test. . ~~In the test description below Present_Value is used as the referenced property. If an Event Enrollment object is being tested, Present_Value should be replaced by the appropriate property reference.~~ It is expected that the TD will often be able to reset the object by using the LifeSafetyOperation service with a

reset or reset-alarm request. ~~It is expected that the TD will often be able to reset the object by using the LifeSafetyOperation service with a reset or reset-alarm request.~~

Test Steps:

1. VERIFY Event_Detection_Enable = TRUE
~~1.~~ *2.* MAKE (~~Present_Value~~ *pMonitoredValue* have a value ~~x such~~ that ~~x~~ corresponds to an OFFNORMAL state)
~~2.~~ *3.* WAIT *pTimeDelay* ~~TimeDelay~~
~~3.~~ *4.* IF (latching is supported) THEN *{*
~~4.~~ *5.* CHECK ~~Event_State~~ *pCurrentState = LIFE_SAFETY_ALARM* ~~OFFNORMAL~~
~~5.~~ *6.* MAKE (the object reset)
~~6.~~ *7.* BEFORE **Notification Fail Time**
~~7.~~ RECEIVE ConfirmedEventNotification-Request,

'Process Identifier' =	(any valid process ID),
'Initiating Device Identifier' =	IUT,
'Event Object Identifier' =	(the intrinsic reporting object being tested or the Event Enrollment object being tested),
'Time Stamp' =	(*T1: any valid time stamp* ~~the current local time~~)
'Notification Class' =	(the configured notification class),
'Priority' =	(the value configured to correspond to a TO-OFFNORMAL transition),
'Event Type' =	CHANGE_OF_LIFE_SAFETY,
'Message Text' =	*(S1, optional: any valid message text),*
'Notify Type' =	EVENT | ALARM,
'AckRequired' =	TRUE | FALSE,
'From State' =	LIFE_SAFETY_ALARM,
'To State' =	OFFNORMAL,
'Event Values' =	~~Present_Value, Mode, Status_Flags, Operation_Expected~~*pMonitoredValue, pMode, pStatusFlags, pOperationExpected*

8. TRANSMIT BACnet-SimpleACK-PDU
9. *IF (Protocol_Revision is present AND Protocol_Revision e 13) THEN*
   VERIFY ~~Status_Flags~~*pStatusFlags* = (~~FALSE~~*TRUE*, FALSE, ?, ?)
10. VERIFY ~~Event_State~~*pCurrentState* = OFFNORMAL
11. *IF (Protocol_Revision is present AND Protocol_Revision ≥ 1) THEN*
    VERIFY Event_Time_Stamps = (~~the timestamp in step 7~~*T1*, *, *)
*12. VERIFY Event_Message_Texts = (S1, *, *)*
}
ELSE *{*
~~12.~~ *13.* BEFORE **Notification Fail Time**
~~13.~~ RECEIVE ConfirmedEventNotification-Request,

'Process Identifier' =	(any valid process ID),
'Initiating Device Identifier' =	IUT,
'Event Object Identifier' =	(the intrinsic reporting object being tested or the Event Enrollment object being tested),
'Time Stamp' =	(*T2*: *any valid time stamp* ~~the current local time~~),
'Notification Class' =	(the configured notification class),
'Priority' =	(the value configured to correspond to a TO-OFFNORMAL transition),
'Event Type' =	CHANGE_OF_LIFE_SAFETY,
'Message Text' =	(S2, optional: any valid message text),
'Notify Type' =	EVENT | ALARM,
'AckRequired' =	TRUE | FALSE,
'From State' =	LIFE_SAFETY_ALARM -
'To State' =	OFFNORMAL,

' Event Values' = ~~Present_Value, Mode, Status_Flags, Operation_Expected~~*pMonitoredValue, pMode, pStatusFlags, pOperationExpected*

14. TRANSMIT BACnet-SimpleACK-PDU
15. *IF (Protocol_Revision is present AND Protocol_Revision e 13) THEN*
    VERIFY ~~Status_Flags~~*pStatusFlags* = (~~FALSE~~*TRUE*, FALSE, ?, ?)
16. VERIFY ~~Event_State~~*pCurrentState* = OFFNORMAL
17. *IF (Protocol_Revision is present AND Protocol_Revision ≥ 1) THEN*
    VERIFY Event_Time_Stamps = (~~the timestamp in step 13~~*T2*, *, *)
18. *VERIFY Event_Message_Texts = (S2, *, *)*
    }

[Change **Clause 8.4.8.6**, p. 185]

8.4.8.6 OFFNORMAL to LIFE_SAFETY_ALARM Transition Test

~~Dependencies: ReadProperty Service Execution Tests, 9.18; WriteProperty Service Execution Tests, 9.22.~~

~~BACnet Reference Clauses: 12.12.7, 12.15, 12.16, 13.3.8, 13.8 and Figure 13-9.~~

Purpose: This test case verifies the correct operation of the CHANGE_OF_LIFE_SAFETY event algorithm for objects transitioning ~~between~~ *from* OFFNORMAL ~~and~~*to* LIFE_SAFETY_ALARM event states. ~~It applies to Event Enrollment objects with an Event_Type of CHANGE_OF_LIFE_SAFETY and to intrinsic event reporting for Life Safety Point and Life Safety Zone objects.~~

Test Concept: The object begins the test in an OFFNORMAL state. ~~The The~~ *pMonitoredValue* ~~Present_Value (referenced property)~~ is made to be in the LIFE_SAFETY_ALARM state. If latching is supported, the object should remain in the OFFNORMAL state until the LifeSafetyOperation service has been executed to reset the object. This test needs to be repeated for all reset operations supported. If latching is not supported, then after the time delay expires, the object should enter the LIFE_SAFETY_ALARM state and transmit an event notification message.

Configuration Requirements: The IUT shall be configured such that the Event_Enable property has a value of TRUE for the TO-OFFNORMAL~~, TO-FAULT~~ and TO-NORMAL transitions. The ~~Issue_Confirmed_Notifications property~~*'Issue Confirmed Notifications' parameter* shall have a value of TRUE. The event-generating objects shall be in an OFFNORMAL state at the start of the test~~. In the test description below pMonitoredValue is used as the referenced property. If an Event Enrollment object is being tested, Present_Value should be replaced by the appropriate property reference~~.It is expected that the TD will often be able to reset the object by using the LifeSafetyOperation service with a reset or reset-alarm request.

Test Steps:

1. VERIFY Event_Detection_Enable = TRUE
~~1.~~ *2.* MAKE (~~Present_Value~~*pMonitoredValue* have a value ~~x such~~ that ~~x~~ corresponds to a~~n~~ LIFE_SAFETY state)
~~2.~~ *3.* WAIT *pTimeDelay*~~TimeDelay~~
~~3.~~ *4.* IF (latching is supported) THEN *{*
~~4.~~ *5.* CHECK ~~Event_State~~*pCurrentState = OFFNORMAL*~~LIFE_SAFETY_ALARM~~
~~5.~~ *6.* MAKE (the object reset)
~~6.~~ *7.* BEFORE **Notification Fail Time**
~~7.~~ RECEIVE ConfirmedEventNotification-Request,
'Process Identifier' = (any valid process ID),
' Initiating Device Identifier' = IUT,
'Event Object Identifier' = (the intrinsic reporting object being tested or the Event Enrollment object being tested),

'Time Stamp' = (*T1*: *any valid time stamp* ~~the current local time~~),
'Notification Class' = (the configured notification class),
'Priority' = (the value configured to correspond to a TO-OFFNORMAL transition),
'Event Type' = CHANGE_OF_LIFE_SAFETY,
'Message Text' = *(S1, optional: any valid message text),*
'Notify Type' = EVENT | ALARM,
~~'~~'AckRequired' = TRUE | FALSE,
'From State' = OFFNORMAL,
'To State' = LIFE_SAFETY_ALARM,
'Event Values' = ~~Present_Value, Mode, Status_Flags, Operation_Expected~~*pMonitoredValue, pMode, pStatusFlags, pOperationExpected*

8. TRANSMIT BACnet-SimpleACK-PDU
9. VERIFY ~~Status_Flags~~*pStatusFlags* = (~~FALSE~~*TRUE*, FALSE, ?, ?)
10 VERIFY ~~Event_State~~*pCurrentState* = LIFE_SAFETY_ALARM
11. *IF (Protocol_Revision is present AND Protocol_Revision ≥ 1) THEN*
    VERIFY Event_Time_Stamps = (~~the timestamp in step 7~~*T1*, *, *)
12. VERIFY Event_Message_Texts = (, *, S2)*

}
ELSE *{*
~~12.~~ *13.* BEFORE **Notification Fail Time**
~~13.~~ RECEIVE ConfirmedEventNotification-Request,
'Process Identifier' = (any valid process ID),
'Initiating Device Identifier' = IUT,
'Event Object Identifier' = (the intrinsic reporting object being tested or the Event Enrollment object being tested),
'Time Stamp' = (*T2: any valid time stamp* ~~the current local time~~),
'Notification Class' = (the configured notification class),
'Priority' = (the value configured to correspond to a TO-OFFNORMAL transition),
'Event Type' = CHANGE_OF_LIFE_SAFETY,
'Message Text' = *(S2, optional: any valid message text),*
'Notify Type' = EVENT | ALARM,
'AckRequired' = TRUE | FALSE,
'From State' = OFFNORMAL,
'To State' = LIFE_SAFETY_ALARM,
'Event Values' = ~~Present_Value, Mode, Status_Flags, Operation_Expected~~*pMonitoredValue, pMode, pStatusFlags, pOperationExpected*

14. TRANSMIT BACnet-SimpleACK-PDU
15. *IF (Protocol_Revision is present AND Protocol_Revision e 13) THEN*
    VERIFY ~~Status_Flags~~*pStatusFlags* = (~~FALSE~~*TRUE*, FALSE, ?, ?)
16. VERIFY ~~Event_State~~*pCurrentState* = LIFE_SAFETY_ALARM
17. *IF (Protocol_Revision is present AND Protocol_Revision ≥ 1) THEN*
    VERIFY Event_Time_Stamps = (~~the timestamp in step 13~~*T2*, *, *)
*18. VERIFY Event_Message_Texts = (S2, *, *)*

}

[Change **Clause 8.4.8.7**, p. 187]

8.4.8.7 Mode Transition Tests when Event State is Maintained

~~Dependencies: ReadProperty Service Execution Tests, 9.18; WriteProperty Service Execution Tests, 9.22.~~

~~BACnet Reference Clauses: 12.12.7, 12.15, 12.16, 13.3.8, 13.8 and Figure 13-9.~~

Purpose: To verify the correct operation of the CHANGE_OF_LIFE_SAFETY event algorithm for objects transitioning between the NORMAL, OFFNORMAL, and LIFE_SAFETY_ALARM event states when a mode change occurs. Tests are conducted when a mode change occurs, but the event state does not change. Tests are also conducted when a mode change occurs simultaneously with an event state change. In this latter case, the test verifies that the notification is immediate rather than waiting for the time delay. ~~It applies to Event Enrollment objects with an Event_Type of CHANGE_OF_LIFE_SAFETY and to intrinsic event reporting for Life Safety Point and Life Safety Zone objects.~~

Test Concept: The object begins the test in a NORMAL state. The Mode is changed. After the time delay expires, the object should transmit an event notification message. This operation is tested in the OFFNORMAL and LIFE_SAFETY_ALARM states as well.

The test is then repeated by changing the Mode property and simultaneously selecting a *pMonitoredValue* designated in ~~the List_Of_Alarm_Values~~*pAlarmValues*. The object should immediately enter the OFFNORMAL state and transmit an event notification message. The *pMonitoredValue*~~Present_Value (referenced property)~~ is then changed to a value corresponding to a NORMAL state, and the Mode is simultaneously written. The object should immediately enter the NORMAL state and transmit an event notification message.

Configuration Requirements: The IUT shall be configured such that the Event_Enable property has a value of TRUE for the TO-OFFNORMAL~~, TO-FAULT~~ and TO-NORMAL transitions. The ~~Issue_Confirmed_Notifications property~~*'Issue Confirmed Notifications' parameter* shall have a value of TRUE. The event-generating objects shall be in a NORMAL state at the start of the test. ~~In the test description below, Present_Value is used as the referenced property. If an Event Enrollment object is being tested, Present_Value should be replaced by the appropriate property reference.~~

Test Steps:

1. VERIFY Event_Detection_Enable = TRUE

~~1.~~ *2.* CHECK ~~Event_State~~*pCurrentState* = NORMAL

~~2.~~ *3.* MAKE (~~Present_Value~~*pMonitoredValue* have a value x such that x corresponds to a NORMAL state)

4. MAKE (pMode = different value that maintains Event_State pCurrentState as NORMAL)

5. WAIT (pTimeDelayNormal)

~~3.~~ *6.* BEFORE **Notification Fail Time**

~~4.~~ RECEIVE ConfirmedEventNotification-Request,

'Process Identifier' =	(any valid process ID),
'Initiating Device Identifier' =	IUT,
'Event Object Identifier' =	(the intrinsic reporting object being tested or the Event Enrollment object being tested),
'Time Stamp' =	(*T1: any valid time stamp* ~~the current local time~~),
'Notification Class' =	(the configured notification class),
'Priority' =	(the value configured to correspond to a TO-NORMAL transition),
'Event Type' =	CHANGE_OF_LIFE_SAFETY,
'Message Text' =	*(S1, optional: any valid message text),*
'Notify Type' =	EVENT | ALARM,
'AckRequired' =	TRUE | FALSE,
'From State' =	NORMAL,
'To State' =	NORMAL,
'Event Values' =	~~Present_Value, Mode, Status_Flags, Operation_Expected~~*pMonitoredValue, pMode, pStatusFlags, pOperationExpected*

~~5.~~ *7.* TRANSMIT BACnet-SimpleACK-PDU

~~6.~~ *8. IF (Protocol_Revision is present AND Protocol_Revision e 13) THEN*

VERIFY ~~Status_Flags~~*pStatusFlags* = (FALSE, FALSE, ?, ?)

~~7.~~ *9.* VERIFY ~~Event_State~~*pCurrentState* = NORMAL

~~8.~~ *10. IF (Protocol_Revision is present AND Protocol_Revision ≥ 1) THEN*

VERIFY Event_Time_Stamps = ~~(the timestamp in step 4, *, *)~~*(*, *, T1)*

11. VERIFY Event_Message_Texts = (, *, S1)*

~~9.~~ *12.* ~~-~~MAKE (~~Present_Value~~*pMonitoredValue* have a value x such that x corresponds to an OFFNORMAL state)

~~10.~~ *13.* MAKE (~~Mode~~*pMode* = different value that maintains ~~Event_State~~*pCurrentState* as OFFNORMAL)

14. WAIT (pTimeDelay)

~~11.~~ *15.* BEFORE **Notification Fail Time**

~~12.~~ RECEIVE ConfirmedEventNotification-Request,

'Process Identifier' =	(any valid process ID),
'Initiating Device Identifier' =	IUT,
'Event Object Identifier' =	(the intrinsic reporting object being tested or the Event Enrollment object being tested),
'Time Stamp' =	(*T2: any valid time stamp* ~~the current local time~~),
'Notification Class' =	(the configured notification class),
'Priority' =	(the value configured to correspond to a TO-OFFNORMAL transition),
'Event Type' =	CHANGE_OF_LIFE_SAFETY,
'Message Text' =	*(S2, optional: any valid message text),*
'Notify Type' =	EVENT | ALARM,
'AckRequired' =	TRUE | FALSE,
'From State' =	NORMAL,
'To State' =	OFFNORMAL,
'Event Values' =	~~Present_Value, Mode, Status_Flags, Operation_Expected~~*pMonitoredValue, pMode, pStatusFlags, pOperationExpected*

~~13.~~ *16.* TRANSMIT BACnet-SimpleACK-PDU

~~14.~~ *17. IF (Protocol_Revision is present AND Protocol_Revision e 13) THEN*

VERIFY ~~Status_Flags~~*pStatusFlags* = (TRUE, FALSE, ?, ?)

~~15.~~ *18.* VERIFY ~~Event_State~~*pCurrentState* = OFFNORMAL

~~16.~~ *19. IF (Protocol_Revision is present AND Protocol_Revision ≥ 1) THEN*

VERIFY Event_Time_Stamps = (~~the timestamp in step 12~~*T2*, *, *)

*20. VERIFY Event_Message_Texts= (S2, *, *)*

17. *21.* MAKE (~~Present_Value~~*pMonitoredValue* have a value x such that x corresponds to a LIFE_SAFETY_ALARM state)

~~18.~~ *22.* MAKE (~~Mode~~*pMode* = different value that maintains ~~Event_State~~*pCurrentState* = LIFE_SAFETY_ALARM)

23. WAIT (pTimeDelay)

~~19.~~ *24.* BEFORE **Notification Fail Time**

~~20.~~ RECEIVE ConfirmedEventNotification-Request,

'Process Identifier' =	(any valid process ID),
'Initiating Device Identifier' =	IUT,
'Event Object Identifier' =	(the intrinsic reporting object being tested or the Event Enrollment object being tested),
'Time Stamp' =	(*T3: any valid time stamp* ~~the current local time~~),
'Notification Class' =	(the configured notification class),
'Priority' =	(the value configured to correspond to a TO-NORMAL transition),
'Event Type' =	CHANGE_OF_LIFE_SAFETY,
'Message Text' =	*(S3, optional: any valid message text),*
'Notify Type' =	EVENT | ALARM,
'AckRequired' =	TRUE | FALSE,
'From State' =	OFFNORMAL,
'To State' =	OFFNORMAL,
'Event Values' =	~~Present_Value, Mode, Status_Flags, Operation_Expected~~*pMonitoredValue, pMode, pStatusFlags, pOperationExpected*

~~21.~~ *25.* TRANSMIT BACnet-SimpleACK-PDU

~~22.~~ *26. IF (Protocol_Revision is present AND Protocol_Revision e 13) THEN*

VERIFY ~~Status_Flags~~*pStatusFlags* = (TRUE, FALSE, ?, ?)

~~23.~~ *27.* VERIFY ~~Event_State~~*pCurrentState* = ~~–~~OFFNORMAL

~~24.~~ *28. IF (Protocol_Revision is present AND Protocol_Revision ≥ 1) THEN*

VERIFY Event_Time_Stamps = (~~the timestamp in step 20~~*T3*, *, *)

*29. VERIFY Event_Message_Texts =(S3, *, *)*

[Change **Clause 8.4.8.8**, p. 189]

8.4.8.8 NORMAL to OFFNORMAL Mode Transition Test

~~Dependencies: ReadProperty Service Execution Tests, 9.18; WriteProperty Service Execution Tests, 9.22.~~

~~BACnet Reference Clauses: 12.12.7, 12.15, 12.16, 13.3.8, 13.8 and Figure 13-9.~~

Purpose: This test case verifies the correct operation of the CHANGE_OF_LIFE_SAFETY event algorithm for objects transitioning ~~between~~ *from* NORMAL ~~and~~*to* OFFNORMAL event states by changing the Mode. ~~It applies to Event Enrollment objects with an Event_Type of CHANGE_OF_LIFE_SAFETY and to intrinsic event reporting for Life Safety Point and Life Safety Zone objects.~~

Test Concept: The object begins the test in a NORMAL state. The Mode is changed to a value forcing the object into an OFFNORMAL state. The object should immediately enter the OFFNORMAL state and transmit an event notification message.

Configuration Requirements: The IUT shall be configured such that the Event_Enable property has a value of TRUE for the TO-OFFNORMAL~~, TO-FAULT~~ and TO-NORMAL transitions. The ~~Issue_Confirmed_Notifications property~~*'Issue Confirmed Notifications' parameter* shall have a value of TRUE. The event-generating objects shall be in a NORMAL state at the start of the test. ~~In the test description below Present_Value is used as the referenced property. If an Event Enrollment object is being tested, Present_Value should be replaced by the appropriate property reference.~~

Test Steps:

1. VERIFY Event_Detection_Enable = TRUE

~~1.~~ *2.* VERIFY ~~Event_State~~*pCurrentState* = NORMAL

~~2.~~ *3.* MAKE (*pMode*~~Mode~~ a different value that forces the state to OFFNORMAL)

~~3.~~ *4.* BEFORE **Notification Fail Time**

RECEIVE ConfirmedEventNotification-Request,

'Process Identifier' =	(any valid process ID),
'Initiating Device Identifier' =	IUT,
'Event Object Identifier' =	(the intrinsic reporting object being tested or the Event Enrollment object being tested),
'Time Stamp' =	(*T1: any valid time stamp* ~~the current local time~~),
'Notification Class' =	(the configured notification class),
'Priority' =	(the value configured to correspond to a TO-OFFNORMAL transition),
'Event Type' =	CHANGE_OF_LIFE_SAFETY,
'Message Text' =	*(S1, optional: any valid message text),*
'Notify Type' =	EVENT | ALARM,
'AckRequired' =	TRUE | FALSE,
'From State' =	NORMAL,
'To State' = ~~—~~	OFFNORMAL,
'Event Values' =	~~Present_Value, Mode, Status_Flags, Operation_Expected~~*pMonitoredValue, pMode, pStatusFlags, pOperationExpected*

~~4.~~ *5.* TRANSMIT BACnet-SimpleACK-PDU

~~5.~~ *6.IF (Protocol_Revision is present AND Protocol_Revision e 13) THEN*

VERIFY ~~Status_Flags~~*pStatusFlags* = (TRUE, FALSE, ?, ?)

~~6.~~ *7.* VERIFY ~~Event_State~~*pCurrentState* = OFFNORMAL

~~7.~~ *8. IF (Protocol_Revision is present AND Protocol_Revision ≥ 1) THEN*

VERIFY Event_Time_Stamps = (~~the timestamp in step 5~~*T1*, *, *)

*9. VERIFY Event_Message_Texts =(S1, *, *)*

[Change **Clause 8.4.8.9**, p. 190]

8.4.8.9 OFFNORMAL to NORMAL Mode Transition Test

~~Dependencies: ReadProperty Service Execution Tests, 9.18; WriteProperty Service Execution Tests, 9.22.~~

~~BACnet Reference Clauses: 12.12.7, 12.15, 12.16, 13.3.8, 13.8 and Figure 13-9.~~

Purpose: This test case verifies the correct operation of the CHANGE_OF_LIFE_SAFETY event algorithm for objects transitioning ~~between~~ *from* OFFNORMAL ~~and~~*to* NORMAL event states by changing the Mode. ~~It applies to Event Enrollment objects with an Event_Type of CHANGE_OF_LIFE_SAFETY and to intrinsic event reporting for Life Safety Point and Life Safety Zone objects.~~

Test Concept: The object begins the test in an OFFNORMAL state. The Mode is changed to a value forcing the object into a NORMAL state. If latching is supported, the object should remain in the OFFNORMAL state until the LifeSafetyOperation service has been executed to reset the object. This test needs to be repeated for all reset operations supported. If latching is not supported, the object should immediately enter the NORMAL state and transmit an event notification message.

Configuration Requirements: The IUT shall be configured such that the Event_Enable property has a value of TRUE for the TO-OFFNORMAL~~, TO-FAULT~~ and TO-NORMAL transitions. The ~~Issue_Confirmed_Notifications property~~*'Issue Confirmed Notifications' parameter* shall have a value of TRUE. The event-generating objects shall be in a OFFNORMAL state at the start of the test. ~~In the test description below Present_Value is used as the referenced property. If an Event Enrollment object is being tested, Present_Value should be replaced by the appropriate property reference.~~ It is expected that the TD will often be able to reset the object by using the LifeSafetyOperation service with a reset or reset-alarm request.

Test Steps:

1. VERIFY Event_Detection_Enable = TRUE

~~1.~~ *2.* MAKE (*pMode*~~Mode~~ a different value that forces the state to NORMAL)

~~2.~~ *3.* IF (latching is supported) THEN *{*

~~3.~~ *4.* CHECK ~~Event_State~~*pCurrentState* = OFFNORMAL

~~4.~~ *5.* MAKE (the object reset)

~~5.~~ *6.* BEFORE **Notification Fail Time**

~~6.~~ RECEIVE ConfirmedEventNotification-Request,

'Process Identifier' =	(any valid process ID),
'Initiating Device Identifier' =	IUT,
'Event Object Identifier' =	(the intrinsic reporting object being tested or the Event Enrollment object being tested),
'Time Stamp' =	(*T1: any valid time stamp* ~~the current local time~~),
'Notification Class' =	(the configured notification class),
'Priority' =	(the value configured to correspond to a TO-OFFNORMAL transition),
'Event Type' =	CHANGE_OF_LIFE_SAFETY,
'Message Text' =	*(S1, optional: any valid message text),*
'Notify Type' =	EVENT | ALARM,
'AckRequired' =	TRUE | FALSE,
'From State' =	OFFNORMAL,

'To State' =	NORMAL,
'Event Values' =	~~Present_Value, Mode, Status_Flags, Operation_Expected~~*pMonitoredValue, pMode, pStatusFlags, pOperationExpected*

7. TRANSMIT BACnet-SimpleACK-PDU
8. *IF (Protocol_Revision is present AND Protocol_Revision e 13) THEN*
   VERIFY ~~Status_Flags~~*pStatusFlags* = (FALSE, FALSE, ?, ?)
9. VERIFY ~~Event_State~~*pCurrentState* = NORMAL
10. *IF (Protocol_Revision is present AND Protocol_Revision ≥ 1) THEN*
    VERIFY Event_Time_Stamps = ~~(the timestamp in step 6, *, *)~~*(*, *, T1)*
11. *VERIFY Event_Message_Texts = (*, *, S1)*

}
ELSE *{*

~~11.~~ *12.* BEFORE **Notification Fail Time**

~~12.~~ RECEIVE ConfirmedEventNotification-Request,

'Process Identifier' =	any valid process ID),
'Initiating Device Identifier' =	IUT,
'Event Object Identifier' =	(the intrinsic reporting object being tested or the Event Enrollment object being tested),
'Time Stamp' =	(T2: *any valid time stamp* ~~the current local time~~),
'Notification Class' =	(the configured notification class),
'Priority' =	(the value configured to correspond to a TO-OFFNORMAL transition),
'Event Type' =	CHANGE_OF_LIFE_SAFETY,
'Message Text' =	*(S2, optional: any valid message text),*
'Notify Type' =	EVENT | ALARM,
'AckRequired' =	TRUE | FALSE,
'From State' =	OFFNORMAL,
'To State' =	NORMAL,
'Event Values' =	~~Present_Value, Mode, Status_Flags, Operation_Expected~~*pMonitoredValue, pMode, pStatusFlags, pOperationExpected*

13. TRANSMIT BACnet-SimpleACK-PDU
14. *IF (Protocol_Revision is present AND Protocol_Revision e 13) THEN*
    VERIFY ~~Status_Flags~~*pStatusFlags* = (FALSE, FALSE, ?, ?)
15. VERIFY ~~Event_State~~*pCurrentState* = NORMAL
16. *IF (Protocol_Revision is present AND Protocol_Revision ≥ 1) THEN*
    VERIFY Event_Time_Stamps = (~~the timestamp in step 12, *, *~~)*(*, *,T2)*
17. *VERIFY Event_Message_Texts = (*, *, S2)*

}

[Change **Clause 8.4.8.10**, p. 191]

8.4.8.10 NORMAL to LIFE_SAFETY_ALARM Mode Transition Test

~~Dependencies: ReadProperty Service Execution Tests, 9.18; WriteProperty Service Execution Tests, 9.22.~~

~~BACnet Reference Clauses: 12.12.7, 12.15, 12.16, 13.3.8, 13.8 and Figure 13-9.~~

Purpose: This test case verifies the correct operation of the CHANGE_OF_LIFE_SAFETY event algorithm for objects transitioning ~~between~~ *from* NORMAL ~~and~~*to* LIFE_SAFETY_ALARM event states by changing the Mode. ~~It applies to Event Enrollment objects with an Event_Type of CHANGE_OF_LIFE_SAFETY and to intrinsic event reporting for Life Safety Point and Life Safety Zone objects.~~

Test Concept: The object begins the test in a NORMAL state. The Mode is changed to a value forcing the object into a LIFE_SAFETY_ALARM state. The object should immediately enter the LIFE_SAFETY_ALARM state and transmit an event notification message. ~~In the test description below Present_Value is used as the referenced property. If an Event Enrollment object is being tested, Present_Value should be replaced by the appropriate property reference.~~

Configuration Requirements: The IUT shall be configured such that the Event_Enable property has a value of TRUE for the TO-OFFNORMAL, TO-FAULT, and TO-NORMAL transitions. The 'Issue Confirmed Notifications' parameter shall have a value of TRUE. The event-generating objects shall be in a NORMAL state at the start of the test.

Test Steps:

1. VERIFY Event_Detection_Enable = TRUE

~~1.~~ *2.* VERIFY ~~Event_State~~*pCurrentState* = NORMAL

~~2.~~ *3.* MAKE (*pMode*~~Mode~~ a different value that forces the state to LIFE_SAFETY_ALARM)

~~3.~~ *4.* BEFORE **Notification Fail Time**

~~4.~~ RECEIVE ConfirmedEventNotification-Request,

'Process Identifier' =	(any valid process ID),
'Initiating Device Identifier' =	IUT,
'Event Object Identifier' =	(the intrinsic reporting object being tested or the Event Enrollment object being tested),
'Time Stamp' =	(T1: *any valid time stamp* ~~the current local time~~),
'Notification Class' =	(the configured notification class),
'Priority' = —	(the value configured to correspond to a TO-OFFNORMAL transition),
'Event Type' =	CHANGE_OF_LIFE_SAFETY,
'Message Text' =	*(S1, optional: any valid message text),*
'Notify Type' =	EVENT | ALARM,
'AckRequired' =	TRUE | FALSE,
'From State' =	NORMAL,
'To State' =	LIFE_SAFETY_ALARM,
'Event Values' =	~~Present_Value, Mode, Status_Flags, Operation_Expected~~*pMonitoredValue, pMode, pStatusFlags, pOperationExpected*

5. TRANSMIT BACnet-SimpleACK-PDU
6. *IF (Protocol_Revision is present AND Protocol_Revision e 13) THEN*
   VERIFY ~~Status_Flags~~*pStatusFlags* = (TRUE, FALSE, ?, ?)
7. VERIFY ~~Event_State~~*pCurrentState* = LIFE_SAFETY_ALARM
8. *IF (Protocol_Revision is present AND Protocol_Revision ≥ 1) THEN*
   VERIFY Event_Time_Stamps = (~~the timestamp in step 4~~*T1*, *, *)
9. *VERIFY Event_Message_Texts = (S1, *, *)*

[Change **Clause 8.4.8.11**, p. 192]

8.4.8.11 LIFE_SAFETY_ALARM to NORMAL Mode Transition Test

~~Dependencies: ReadProperty Service Execution Tests, 9.18; WriteProperty Service Execution Tests, 9.22.~~

~~BACnet Reference Clauses: 12.12.7, 12.15, 12.16, 13.3.8, 13.8 and Figure 13-9.~~

~~-~~Purpose: This test case verifies the correct operation of the CHANGE_OF_LIFE_SAFETY event algorithm for objects transitioning ~~between~~ *from* LIFE_SAFETY_ALARM ~~and~~*to* NORMAL event states by changing the Mode property. ~~It applies to Event Enrollment objects with an Event_Type of CHANGE_OF_LIFE_SAFETY and to intrinsic event reporting for Life Safety Point and Life Safety Zone objects.~~

Test Concept: The object begins the test in a LIFE_SAFETY_ALARM state. The Mode property is changed to a value forcing the object into an NORMAL state. If latching is supported, the object should remain in the LIFE_SAFETY_ALARM state until the LifeSafetyOperation service has been executed to reset the object. This test needs to be repeated for all reset operations supported. If latching is not supported, the object should immediately enter the NORMAL state and transmit an event notification message.

Configuration Requirements: The IUT shall be configured such that the Event_Enable property has a value of TRUE for the TO-OFFNORMAL, ~~TO-FAULT~~ and TO-NORMAL transitions. The ~~Issue_Confirmed_Notifications property~~*'Issue Confirmed Notifications' parameter* shall have a value of TRUE. The event-generating objects shall be in a LIFE_SAFETY_ALARM state at the start of the test. ~~In the test description below Present_Value is used as the referenced property. If an Event Enrollment object is being tested, Present_Value should be replaced by the appropriate property reference.~~ It is expected that the TD will often be able to reset the object by using the LifeSafetyOperation service with a reset or reset-alarm request.

Test Steps:

1. VERIFY Event_Detection_Enable = TRUE
~~1.~~ *2.* MAKE (*pMode*~~Mode~~ a different value that forces the state to NORMAL)
~~2.~~ *3.* IF (latching is supported) THEN *-{*
~~3.~~ *4.* CHECK ~~Event_State~~*pCurrentState* = LIFE_SAFETY_ALARM
~~4.~~ *5.* MAKE (the object reset)
~~5.~~ *6.* BEFORE **Notification Fail Time**
~~6.~~ RECEIVE ConfirmedEventNotification-Request,

'Process Identifier' =	(any valid process ID),
'Initiating Device Identifier' =	IUT,
'Event Object Identifier' =	(the intrinsic reporting object being tested or the Event Enrollment object being tested),
'Time Stamp' =	(*T1*: *any valid time stamp* ~~the current local time~~),
'Notification Class' =	(the configured notification class),
'Priority' =	(the value configured to correspond to a TO-OFFNORMAL transition),
'Event Type' =	CHANGE_OF_LIFE_SAFETY,
'Message Text' =	*(S1, optional: any valid message text),*
'Notify Type' =	EVENT | ALARM,
'AckRequired' =	TRUE | FALSE,
'From State' =	LIFE_SAFETY_ALARM,
'To State' =	NORMAL,
'Event Values' =	~~Present_Value, Mode, Status_Flags, Operation_Expected~~*pMonitoredValue, pMode, pStatusFlags, pOperationExpected*

7. TRANSMIT BACnet-SimpleACK-PDU
8. *IF (Protocol_Revision is present AND Protocol_Revision e 13) THEN*
   VERIFY ~~Status_Flags~~*pStatusFlags* = (FALSE, FALSE, ?, ?)
9. VERIFY ~~Event_State~~*pCurrentState* = NORMAL
10. *IF (Protocol_Revision is present AND Protocol_Revision ≥ 1) THEN*
    VERIFY Event_Time_Stamps = (~~the timestamp in step 6*, *~~)*(*, *, T1)*
11. VERIFY Event_Message_Texts = (, *, S1)*
}
ELSE *{*
~~11.~~ *12.* BEFORE **Notification Fail Time**
~~12.~~ RECEIVE ConfirmedEventNotification-Request,

'Process Identifier' =	(any valid process ID),
'Initiating Device Identifier' =	IUT,

'Event Object Identifier' =	(the intrinsic reporting object being tested or the Event Enrollment object being tested),
'Time Stamp' =	(*T2: any valid time stamp* ~~the current local time~~),
'Notification Class' =	(the configured notification class),
'Priority' =	(the value configured to correspond to a TO-OFFNORMAL transition),
'Event Type' =	CHANGE_OF_LIFE_SAFETY,
'Message Text' =	*(S2, optional: any valid message text),*
'Notify Type' =	EVENT | ALARM,
'AckRequired' =	TRUE | FALSE,
'From State' =	LIFE_SAFETY_ALARM,
'To State' =	NORMAL,
'Event Values' =	~~Present_Value, Mode, Status_Flags, Operation_Expected~~*pMonitoredValue, pMode, pStatusFlags, pOperationExpected*

13. TRANSMIT BACnet-SimpleACK-PDU
14. *IF (Protocol_Revision is present AND Protocol_Revision e 13) THEN*
    VERIFY ~~Status_Flags~~*pStatusFlags* = (FALSE, FALSE, ?, ?)
15. VERIFY ~~Event_State~~*pCurrentState* = NORMAL
16. *IF (Protocol_Revision is present AND Protocol_Revision ≥ 1) THEN*
    VERIFY Event_Time_Stamps = (~~the timestamp in step 12,*, *~~)*(*, *, T2)*
17. *VERIFY Event_Message_Texts = (*, *, S2)*

}

[Change **Clause 8.4.8.12**, p. 193]

8.4.8.12 LIFE_SAFETY_ALARM to OFFNORMAL Mode Transition Test

~~Dependencies: ReadProperty Service Execution Tests, 9.18; WriteProperty Service Execution Tests, 9.22.~~

~~BACnet Reference Clauses: 12.12.7, 12.15, 12.16, 13.3.8, 13.8 and Figure 13-9.~~

Purpose: This test case verifies the correct operation of the CHANGE_OF_LIFE_SAFETY event algorithm for objects transitioning ~~between~~ *from* LIFE_SAFETY_ALARM ~~and~~*to* OFFNORMAL event states. ~~It applies to Event Enrollment objects with an Event_Type of CHANGE_OF_LIFE_SAFETY and to intrinsic event reporting for Life Safety Point and Life Safety Zone objects.~~

Test Concept: The object begins the test in a LIFE_SAFETY_ALARM state. The Mode property is changed to a value forcing the object into an OFFNORMAL state. If latching is supported, the object should remain in the LIFE_SAFETY_ALARM state until the LifeSafetyOperation service has been executed to reset the object. This test needs to be repeated for all reset operations supported. If latching is not supported, the object should immediately enter the OFFNORMAL state and transmit an event notification message.

Configuration Requirements: The IUT shall be configured such that the Event_Enable property has a value of TRUE for the TO-OFFNORMAL~~, TO-FAULT~~ and TO-NORMAL transitions. The ~~Issue_Confirmed_Notifications property~~*'Issue Confirmed Notifications' parameter* shall have a value of TRUE. The event-generating objects shall be in a LIFE_SAFETY_ALARM state at the start of the test. ~~In the test description below Present_Value is used as the referenced property. If an Event Enrollment object is being tested, Present_Value should be replaced by the appropriate property reference.~~ It is expected that the TD will often be able to reset the object by using the LifeSafetyOperation service with a reset or reset-alarm request.

Test Steps:

1. VERIFY Event_Detection_Enable = TRUE

~~1.~~ *2.* MAKE (*pMode* Mode a different value that forces the state to OFFNORMAL)

~~2.~~ *3.* IF (latching is supported) THEN *{*

~~3.~~ *4.* CHECK ~~Event_State~~*pCurrentState* = OFFNORMAL

~~4.~~ *5.* MAKE (the object reset)

~~5.~~ *6.* BEFORE **Notification Fail Time**

~~6.~~ RECEIVE ConfirmedEventNotification-Request,

'Process Identifier' =	(any valid process ID),
'Initiating Device Identifier' =	IUT,
'Event Object Identifier' =	(the intrinsic reporting object being tested or the Event Enrollment object being tested),
'Time Stamp' =	(*T1: any valid time stamp* ~~the current local time~~),
'Notification Class' =	(the configured notification class),
'Priority' =	(the value configured to correspond to a TO-OFFNORMAL transition),
'Event Type' =	CHANGE_OF_LIFE_SAFETY,
'Message Text' =	*(S1, optional: any valid message text),*
'Notify Type' =	EVENT | ALARM,
'AckRequired' =	TRUE | FALSE,
'From State' =	LIFE_SAFETY_ALARM,
'To State' =	OFFNORMAL,
'Event Values' =	~~Present_Value, Mode, Status_Flags, Operation_Expected~~*pMonitoredValue, pMode, pStatusFlags, pOperationExpected*

7. TRANSMIT BACnet-SimpleACK-PDU

8. *IF (Protocol_Revision is present AND Protocol_Revision e 13) THEN*
   VERIFY ~~Status_Flags~~*pStatusFlags* = (~~FALSE~~*TRUE*, FALSE, ?, ?)

9. VERIFY ~~Event_State~~*pCurrentState* = OFFNORMAL

10. *IF (Protocol_Revision is present AND Protocol_Revision ≥ 1) THEN*
    VERIFY Event_Time_Stamps = (~~the timestamp in step 6~~*T1*, *, *)

*11. VERIFY Event_Message_Texts = (S1, *, *)*

}

ELSE *{*

~~11.~~ *12.* BEFORE **Notification Fail Time**

~~12.~~ RECEIVE ConfirmedEventNotification-Request,

'Process Identifier' =	(any valid process ID),
'Initiating Device Identifier' =	IUT,
'Event Object Identifier' =	(the intrinsic reporting object being tested or the Event Enrollment object being tested),
'Time Stamp' =	(*T2: any valid time stamp* ~~the current local time~~),
'Notification Class' =	(the configured notification class),
'Priority' =	(the value configured to correspond to a TO-OFFNORMAL transition),
'Event Type' =	CHANGE_OF_LIFE_SAFETY,
'Message Text' =	*(S2, optional: any valid message text),*
'Notify Type' =	EVENT | ALARM,
'AckRequired' =	TRUE | FALSE,
'From State' =	LIFE_SAFETY_ALARM,
'To State' =	OFFNORMAL,
'Event Values' =	~~Present_Value, Mode, Status_Flags, Operation_Expected~~*pMonitoredValue, pMode, pStatusFlags, pOperationExpected*

13. TRANSMIT BACnet-SimpleACK-PDU

14. *IF (Protocol_Revision is present AND Protocol_Revision e 13) THEN*
    VERIFY ~~Status_Flags~~*pStatusFlags* = (FALSE, FALSE, ?, ?)

15. VERIFY ~~Event_State~~*pCurrentState* = OFFNORMAL

16. *IF (Protocol_Revision is present AND Protocol_Revision ≥ 1) THEN*

VERIFY Event_Time_Stamps = (~~the timestamp in step 12~~*T2*, *, *)

*17. VERIFY Event_Message_Texts = (S2, *, *)*

}

[Change **Clause 8.4.8.13**, p. 194]

8.4.8.13 OFFNORMAL to LIFE_SAFETY_ALARM Mode Transition Test

~~Dependencies: ReadProperty Service Execution Tests, 9.18; WriteProperty Service Execution Tests, 9.22.~~

~~BACnet Reference Clauses: 12.21.7, 12.15, 12.16, 13.3.8, 13.8 and Figure 13-9.~~

Purpose: This test case verifies the correct operation of the CHANGE_OF_LIFE_SAFETY event algorithm for objects transitioning ~~between~~ *from* the OFFNORMAL ~~and~~*to* LIFE_SAFETY_ALARM event states by changing the Mode property. ~~It applies to Event Enrollment objects with an Event_Type of CHANGE_OF_LIFE_SAFETY and to intrinsic event reporting for Life Safety Point and Life Safety Zone objects.~~

Test Concept: The object begins the test in an OFFNORMAL state. The Mode is changed to a value forcing the object into a LIFE_SAFETY_ALARM state. If latching is supported, the object should remain in the OFFNORMAL state until the LifeSafetyOperation service has been executed to reset the object. This test needs to be repeated for all reset operations supported. If latching is not supported, the object should immediately enter the LIFE_SAFETY_ALARM state and transmit an event notification message.

Configuration Requirements: The IUT shall be configured such that the Event_Enable property has a value of TRUE for the TO-OFFNORMAL~~, TO FAULT~~ and TO-NORMAL transitions. The ~~Issue_Confirmed_Notifications property~~*'Issue Confirmed Notifications' parameter* shall have a value of TRUE. The event-generating objects shall be in a LIFE_SAFETY_ALARM state at the start of the test. ~~In the test description below Present_Value is used as the referenced property. If an Event Enrollment object is being tested, Present_Value should be replaced by the appropriate property reference.~~ It is expected that the TD will often be able to reset the object by using the LifeSafetyOperation service with a reset or reset-alarm request.

Test Steps:

1. VERIFY Event_Detection_Enable = TRUE

~~1.~~ *2.* MAKE (*pMode*~~Mode~~ a different value that forces the state to LIFE_SAFETY_ALARM)

~~2.~~ *3.* IF (latching is supported) THEN *{*

~~3.~~ *4.* CHECK ~~Event_State~~*pCurrentState* = OFFNORMAL

~~4.~~ *5.* MAKE (the object reset)

~~5.~~ *6.* BEFORE **Notification Fail Time**

~~6.~~ RECEIVE ConfirmedEventNotification-Request,

'Process Identifier' =	(any valid process ID),
'Initiating Device Identifier' =	IUT,
'Event Object Identifier' =	(the intrinsic reporting object being tested or the Event Enrollment object being tested),
'Time Stamp' =	(*T1: any valid time stamp* ~~the current local time~~),
'Notification Class' =	(the configured notification class),
'Priority' =	(the value configured to correspond to a ~~TO-OFFNORMAL~~*TO-OFFNORMAL* transition),
'Event Type' =	CHANGE_OF_LIFE_SAFETY,
'Message Text' =	*(S1, optional: any valid message text),*
'Notify Type' =	EVENT | ALARM,
'AckRequired' =	TRUE | FALSE,
'From State' =	LIFE_SAFETY_ALARM,
'To State' =	OFFNORMAL,

'Event Values' = ~~Present_Value, Mode, Status_Flags, Operation_Expected~~*pMonitoredValue, pMode, pStatusFlags, pOperationExpected*

7. TRANSMIT BACnet-SimpleACK-PDU
8. *IF (Protocol_Revision is present AND Protocol_Revision e 13) THEN*
   VERIFY ~~Status_Flags~~*pStatusFlags* = (~~FALSE~~*TRUE*, FALSE, ?, ?)
9. VERIFY ~~Event_State~~*pCurrentState* = OFFNORMAL
10. *IF (Protocol_Revision is present AND Protocol_Revision ≥ 1) THEN*
    VERIFY Event_Time_Stamps = (~~the timestamp in step 6~~*T1*, *, *)
11. *VERIFY Event_Message_Texts = (S1, *, *)*

}
ELSE *{*

~~11.~~ *12.* BEFORE **Notification Fail Time**

~~12.~~ RECEIVE ConfirmedEventNotification-Request,

'Process Identifier' =	(any valid process ID),
'Initiating Device Identifier' =	IUT,
'Event Object Identifier' =	(the intrinsic reporting object being tested or the Event Enrollment object being tested),
'Time Stamp' =	(*T2*: *any valid time stamp* ~~the current local time~~),
'Notification Class' =	(the configured notification class),
'Priority' =	(the value configured to correspond to a TO-OFFNORMAL transition),
'Event Type' =	CHANGE_OF_LIFE_SAFETY,
'Message Text' =	*(S2, optional: any valid message text),*
'Notify Type' =	EVENT | ALARM,
'AckRequired' =	TRUE | FALSE,
'From State' =	LIFE_SAFETY_ALARM,
'To State' =	OFFNORMAL,
'Event Values' =	~~Present_Value, Mode, Status_Flags, Operation_Expected~~ *pMonitoredValue, pMode, pStatusFlags, pOperationExpected*

13. TRANSMIT BACnet-SimpleACK-PDU
14. *IF (Protocol_Revision is present AND Protocol_Revision e 13) THEN*
    VERIFY ~~Status_Flags~~*pStatusFlags*= (~~FALSE~~*TRUE*, FALSE, ?, ?)
15. VERIFY ~~Event_State~~*pCurrentState* = OFFNORMAL
16. *IF (Protocol_Revision is present AND Protocol_Revision ≥ 1) THEN*
    VERIFY Event_Time_Stamps = (~~the timestamp in step 12~~*T2*, *, *)
17. *VERIFY Event_Message_Texts = (S2, *, *)*

}

[Remove **Clause 8.4.8.14**, p. 196]
[Reviewer Note: Addendum *af* to ANSI/ASHRAE 135-2010 introduced a new FAULT_LIFE_SAFETY fault algorithm which is covered by another test proposed in this addendum.]

~~**8.4.8.14 TO-FAULT Transition Tests**~~

~~Dependencies: ReadProperty Service Execution Tests, 9.18; WriteProperty Service Execution Tests, 9.22.~~

~~BACnet Reference Clauses: 12.12.7, 12.15, 12.16, 13.3.8, 13.8 and Figure 13-9.~~

~~Purpose: This test case verifies the correct operation of the CHANGE_OF_LIFE_SAFETY event algorithm for objects transitioning between the NORMAL and FAULT event states. It applies to intrinsic event reporting for Life Safety Point and Life Safety Zone objects.~~

~~Test Concept: The object begins the test in a NORMAL state. The Present_Value (referenced property) is changed to one of the values designated in List_Of_Fault_Values. After the time delay expires, the object should enter the FAULT state and transmit an event notification message. This test also incorporates the return transition from FAULT to NORMAL.~~

~~Configuration Requirements: The IUT shall be configured such that the Event_Enable property has a value of TRUE for the TO-OFFNORMAL, TO-FAULT and TO-NORMAL transitions. The Issue_Confirmed_Notifications property shall have a value of TRUE. The event-generating objects shall be in a NORMAL state at the start of the test.~~

~~In the test description below, Present_Value is used as the referenced property.~~

~~Test Steps:~~

~~1. VERIFY Event_State = NORMAL~~
~~2. MAKE (Present_Value have a value x such that x corresponds to a FAULT state)~~
~~3. BEFORE **Notification Fail Time**~~
~~4. RECEIVE ConfirmedEventNotification-Request,~~

~~'Process Identifier' = (any valid process ID),~~
~~'Initiating Device Identifier' = IUT,~~
~~'Event Object Identifier' = (the intrinsic reporting object being tested),~~
~~'Time Stamp' = (the current local time),~~
~~'Notification Class' = (the configured notification class),~~
~~'Priority' = (the value configured to correspond to a TO-FAULT transition),~~
~~'Event Type' = CHANGE_OF_LIFE_SAFETY,~~
~~'Notify Type' = EVENT | ALARM,~~
~~'AckRequired' = TRUE | FALSE,~~
~~'From State' = NORMAL,~~
~~'To State' = FAULT,~~
~~'Event Values' = Present_Value, Mode, Status_Flags, Operation_Expected~~

~~5. TRANSMIT BACnet-SimpleACK-PDU~~
~~6. VERIFY Status_Flags = (TRUE, FALSE, ?, ?)~~
~~7. VERIFY Event_State = FAULT~~
~~8. VERIFY Event_Time_Stamps = (the timestamp in step 5, *, *)~~
~~9. MAKE (Present_Value have a value x such that x corresponds to a NORMAL state)~~
~~10. BEFORE **Notification Fail Time**~~
~~11. RECEIVE ConfirmedEventNotification-Request,~~
~~'Process Identifier' = (any valid process ID),~~
~~'Initiating Device Identifier' = IUT,~~
~~'Event Object Identifier' = (the intrinsic reporting object being tested),~~
~~'Time Stamp' = (the current local time),~~
~~'Notification Class' = (the configured notification class),~~
~~'Priority' = (the value configured to correspond to a TO-NORMAL transition),~~
~~'Event Type' = CHANGE_OF_LIFE_SAFETY,~~
~~'Notify Type' = EVENT | ALARM,~~
~~'AckRequired' = TRUE | FALSE,~~
~~'From State' = FAULT,~~
~~'To State' = NORMAL,~~
~~'Event Values' = Present_Value, Mode, Status_Flags, Operation_Expected~~

~~12. TRANSMIT BACnet-SimpleACK-PDU~~
~~13. VERIFY Status_Flags = (TRUE, FALSE, ?, ?)~~
~~14. VERIFY Event_State = NORMAL~~
~~15. VERIFY Event_Time_Stamps = (the timestamp in step 13, *, *)~~

[Change **Clause 8.4.9**, p. 197]

8.4.9 EXTENDED Tests ***(ConfirmedEventNotification)***

~~Dependencies: None~~

~~BACnet Reference Clauses: 13.2 and 13.3~~

Purpose: ~~This test verifies the correct generation of extended ConfirmedEventNotification messages. It applies to event generating objects with an Event_Type of EXTENDED. This applies to any Event Enrollment object that uses a proprietary algorithm.~~*To verify the correct generation of EXTENDED event notifications.*

Test Concept: ~~The event generating object begins the test in a NORMAL state.~~ The event generating object is made to transition to any state by any means necessary. The resulting ConfirmedEventNotification message is received and verified.

Configuration Requirements: The IUT shall be configured such that the Event_Enable property has a value of TRUE for whichever transition shall be used for the test. The ~~Issue_Confirmed_Notifications property~~*'Issue Confirmed Notifications' parameter* shall have a value of TRUE. ~~The event generating object shall be in a NORMAL state at the start of the test. .~~*D1 and D2 are vendor specific delays (either of them or both may be zero).*

Test Steps:

~~1. VERIFY Event_State = NORMAL~~
~~2. MAKE (the event generating object transition)~~
~~3. WAIT (Time_Delay)~~
~~4. BEFORE **Notification Fail Time**~~
~~RECEIVE ConfirmedEventNotification-Request,~~
~~'Process Identifier' = (any valid process ID),~~
~~'Initiating Device Identifier' = IUT,~~
~~'Event Object Identifier' = (the event generating object being tested),~~
~~'Time Stamp' = (the current local time),~~
~~'Notification Class' = (the configured notification class, or absent if the event generating object is using a Recipient property instead),~~
~~'Priority' = (the value configured for this transition),~~
~~'Event Type' = EXTENDED,~~
~~'Notify Type' = EVENT | ALARM,~~
~~'AckRequired' = TRUE | FALSE,~~
~~'From State' = NORMAL,~~
~~'To State' = (the state the object was made to transition to),~~
~~'Event Values' = VendorId, extendedEventType, and any other values as defined by the vendor~~

1. IF (the object generates TO-OFFNORMAL transitions) THEN
2. READ CS1 = pCurrentState
3. MAKE (an OFFNORMAL condition exist)
4. WAIT D1
*5. BEFORE **Notification Fail Time***
RECEIVE ConfirmedEventNotification-Request,
'Process Identifier' = (any valid process ID),
'Initiating Device Identifier' = IUT,
'Event Object Identifier' = (the event generating object),
'Time Stamp' = (TS1: the current local time),
'Notification Class' = (the configured notification class),
'Priority' = (the value configured for TO_OFFNORMAL),
'Event Type' = EXTENDED,
'Message Text' = (optional, any valid message text),
'Notify Type' = EVENT | ALARM,
'AckRequired' = TRUE | FALSE,
'From State' = CS1,
'To State' = (CS2: any offnormal valid event state),

'Event Values' = (pVendorId: any valid vendor id),
(pEventType: any valid event-type),
(a list of 0 or more valid parameters as defined by the Vendor)

6. TRANSMIT BACnet-SimpleACK-PDU
7. IF (Protocol_Revision is present AND Protocol_Revision e 13) THEN
VERIFY pStatusFlags = (TRUE, FALSE,?,?)
8. VERIFY pCurrentState = CS2
*9. VERIFY Event_Time_Stamps = (TS1, *, *)*
10. IF (the object generates TO_NORMAL transitions) THEN
11. READ CS2 = pCurrentState
12. MAKE (a NORMAL condition exist)
13. WAIT D2
*14. BEFORE **Notification Fail Time***
RECEIVE ConfirmedEventNotification-Request,
'Process Identifier' = (any valid process ID),
'Initiating Device Identifier' = IUT,
'Event Object Identifier' = (the intrinsic reporting object being tested),
'Time Stamp' = (TS2: the current local time),
'Notification Class' = (the configured notification class),
'Priority' = (the value configured for TO-NORMAL),
'Event Type' = EXTENDED,
'Message Text' = (optional, any valid message text),
'Notify Type' = EVENT | ALARM,
'AckRequired' = TRUE | FALSE,
'From State' = CS2,
'To State' = NORMAL,
'Event Values' = (pVendorId: any valid vendor id),
(pEventType: any valid event-type),
(a list of 0 or more valid parameters as defined by the Vendor)
15. TRANSMIT BACnet-SimpleACK-PDU
16. IF (Protocol_Revision is present AND Protocol_Revision e 13) THEN
VERIFY pStatusFlags = (FALSE, FALSE, ?, ?)
17. VERIFY pCurrentState = NORMAL
18. VERIFY Event_Time_Stamps = (, *, TS2)*

~~Passing Result: The 'Message Text' parameter is omitted in the test description because it is optional. The IUT may include this parameter in the notification messages.~~

[Change **Clause 8.4.10**, p. 197]

~~**8.4.10**~~ *7.3.2.21.1.2* Network Priority Test
~~Dependencies: ReadProperty Service Execution Tests, 9.18; WriteProperty Service Execution Tests, 9.22.~~

~~BACnet Reference Clause: 13.4.1~~

Purpose: To verify that the correct network priority is used when transmitting EventNotification requests.

Test Concept: The IUT is made to generate a ConfirmedEventNotification with a specific priority. The network priority in the NPCI is checked to ensure that it is correct based on the priority of the event.

~~Test~~ Configuration *Requirements*: The IUT is configured with an event generating object, E1, that refers to Notification Class object, N1.

Test Steps:

1. IF (the IUT can generate ConfirmedEventNotifications with an event priority in the range 0..63) THEN
2. MAKE (The IUT initiate a ConfirmedEventNotification-Request with a event priority in the range 0..63)
3. BEFORE **Notification Fail Time**
RECEIVE ConfirmedEventNotification-Request,
'~~"~~Network Priority'~~"~~ = 3
'Process Identifier' = (any valid process ID),
'Initiating Device Identifier' = IUT,
'Event Object Identifier' = (the intrinsic reporting object or the Event Enrollment object being tested),
'Time Stamp' = (~~the current local time of the IUT~~*any valid time stamp*),
'Notification Class' = (the configured notification class),
'Priority' = (a value in the range 0..63),
'Event Type' = (any valid value),
'Message Text' = *(optional, any valid message text),*
'Notify Type' = EVENT | ALARM,
'AckRequired' = TRUE | FALSE,
'From State' = (any valid value),
'To State' = (any valid value),
'Event Values' = (any valid valid)
4. TRANSMIT BACnet-SimpleACK-PDU
5. IF (the IUT can generate ConfirmedEventNotifications with an event priority in the range 64..127) THEN
6. MAKE (The IUT initiate a ConfirmedEventNotification-Request with an event priority in the range 64..127)
7. BEFORE **Notification Fail Time**
RECEIVE ConfirmedEventNotification-Request,
'~~"~~Network Priority'~~"~~ = 2
'Process Identifier' = (any valid process ID),
'Initiating Device Identifier' = IUT,
'Event Object Identifier' = (the intrinsic reporting object or the Event Enrollment object being tested),
'Time Stamp' = (~~the current local time of the IUT~~*any valid time stamp*),
'Notification Class' = (the configured notification class),
'Priority' = (a value in the range 64..127),
'Event Type' = (any valid value),
'Message Text' = *(optional, any valid message text),*
'Notify Type' = EVENT | ALARM,
'AckRequired' = TRUE | FALSE,
'From State' = (any valid value),
'To State' = (any valid value),
'Event Values' = (any valid valid)
8. TRANSMIT BACnet-SimpleACK-PDU
9. IF (the IUT can generate ConfirmedEventNotifications with an event priority in the range 128..191) THEN
10. MAKE (the IUT initiate a ConfirmedEventNotification-Request with a event priority in the range 128..191)
11. BEFORE **Notification Fail Time**
RECEIVE ConfirmedEventNotification-Request,
'~~"~~Network Priority'~~"~~ = 1
'Process Identifier' = (any valid process ID),
'Initiating Device Identifier' = IUT,
'Event Object Identifier' = (the intrinsic reporting object or the Event Enrollment object being tested),
'Time Stamp' = (~~the current local time of the IUT~~*any valid time stamp*),
'Notification Class' = (the configured notification class),
'Priority' = (a value in the range 128..191),
'Event Type' = (any valid value),
'Message Text' = *(optional, any valid message text),*
'Notify Type' = EVENT | ALARM,
'AckRequired' = TRUE | FALSE,

'From State' = (any valid value),
'To State' = (any valid value),
'Event Values' = (any valid valid)

12. TRANSMIT BACnet-SimpleACK-PDU
13. IF (the IUT can generate ConfirmedEventNotifications with an event priority in the range 192..255) THEN
14. MAKE (The IUT initiate a ConfirmedEventNotification-Request with a event priority in the range 192..255)
15. BEFORE **Notification Fail Time**
RECEIVE ConfirmedEventNotification-Request,
~~"~~Network Priority'~~"~~ = 0
'Process Identifier' = (any valid process ID),
'Initiating Device Identifier' = IUT,
'Event Object Identifier' = (the intrinsic reporting object or the Event Enrollment object being tested),
'Time Stamp' = (~~the current local time of the IUT~~*any valid time stamp*),
'Notification Class' = (the configured notification class),
'Priority' = (a value in the range 192..255),
'Event Type' = (any valid value),
'Message Text' = *(optional, any valid message text),*
'Notify Type' = EVENT | ALARM,
'AckRequired' = TRUE | FALSE,
'From State' = (any valid value),
'To State' = (any valid value),
'Event Values' = (any valid valid)
16. TRANSMIT BACnet-SimpleACK-PDU

[Add new **Clause 8.4.10**, p. 197]
[Reviewer Note: New event algorithm for Protocol_Revision 10.]

8.4.10 DOUBLE_OUT_OF_RANGE Test (ConfirmedEventNotification)

Purpose: To verify the correct operation of the DOUBLE_OUT_OF_RANGE event algorithm.

Test Concept: This test is the same as 8.4.6, except that the Event_Type is DOUBLE_OUT_OF_RANGE instead of OUT_OF_RANGE.

[Add new **Clause 8.4.11**, p. 199]
[Reviewer Note: New event algorithm for Protocol_Revision 10.]

8.4.11 SIGNED_OUT_OF_RANGE Test (ConfirmedEventNotification)

Purpose: To verify the correct operation of the SIGNED_OUT_OF_RANGE event algorithm.

Test Concept: This test is the same as 8.4.6, except that the Event_Type is SIGNED_OUT_OF_RANGE instead of OUT_OF_RANGE.

[Add new **Clause 8.4.12**, p. 199]
[Reviewer Note: New event algorithm for Protocol_Revision 10.]

8.4.12 UNSIGNED_OUT_OF_RANGE Test (ConfirmedEventNotification)

Purpose: To verify the correct operation of the UNSIGNED_OUT_OF_RANGE event algorithm.

Test Concept: This test is the same as 8.4.6, except that the Event_Type is UNSIGNED_OUT_OF_RANGE instead of OUT_OF_RANGE.

[Add new **Clause 8.4.13**, p. 199]
[Reviewer Note: New event algorithm for Protocol_Revision 10.]

8.4.13 CHANGE_OF_CHARACTERSTRING Test (ConfirmedEventNotification)

Purpose: To verify the correct operation of the CHANGE_OF_CHARACTERSTRING event algorithm.

Test Concept: The object begins the test in a NORMAL state. pMonitoredValue is changed to a value that is one of the values designated in pAlarmValues. After the time delay expires, the object should enter the OFFNORMAL state and transmit an event notification message. The pMonitoredValue is then changed to a different value in the pAlarmValues. After the time delay expires, the object should enter the OFFNORMAL state and transmit an event notification message. pMonitoredValue is then changed to a value corresponding to a NORMAL state. After the time delay, the object should enter the NORMAL state and transmit an event notification message. If the IUT claims conformance to Protocol_Revision 12 or lower, the transition to and from the FAULT state is also tested.

Configuration Requirements: The IUT shall be configured such that the Event_Enable property has a value of TRUE for the TO-OFFNORMAL, TO-FAULT, and TO-NORMAL transitions. The 'Issue Confirmed Notifications' parameter shall have a value of TRUE. The property and Event_Parameters element, respectively, represented by pAlarmValues shall be non-empty. The event-generating object shall be in a NORMAL state at the start of the test.

If the IUT claims conformance to Protocol_Revision 12 or lower and supports intrinsic reporting for CharacterString Value objects, the intrinsic reporting object shall be configured with at least one of the two properties, Alarm_Values (referred to as pAlarmValues in the test steps) and Fault_Values (referred to as pFaultValues in the test steps), containing at least one characterstring.

If the IUT claims conformance to Protocol_Revision 12 or lower and supports algorithmic change reporting with an Event_Type of CHANGE_OF_CHARACTERSTRING, the List_Of_Alarm_Values parameter of the Event_Parameters property (referred to as pAlarmValues in the test steps) shall contain at least one characterstring.

If the IUT claims conformance to Protocol_Revision 13 or greater and supports the CHANGE_OF_CHARACTERSTRING algorithm, the IUT shall be configured with at least one characterstring for pAlarmValues.

Test Steps:

1. VERIFY pCurrentState = NORMAL
2. IF ((Protocol_Revision is present AND Protocol_Revision ≥ 13) OR ((Protocol_Revision is present AND Protocol_Revision d 12) AND (pAlarmValues contains at least one characterstring))) THEN {
3. IF (pMonitoredValue is writable) THEN
   WRITE pMonitoredValue = (a value from pAlarmValues)
   ELSE
   MAKE (pMonitoredValue have a value from pAlarmValues)
4. WAIT (pTimeDelay)
5. BEFORE **Notification Fail Time**
   RECEIVE ConfirmedEventNotification-Request,
   'Process Identifier' = (any valid process ID),
   'Initiating Device Identifier' = IUT,
   'Event Object Identifier' = (the object being tested),
   'Time Stamp' = (Toffnormal: any valid time stamp),
   'Notification Class' = (the configured notification class),
   'Priority' = (the value configured to correspond to a TO-OFFNORMAL transition),
   'Event Type' = CHANGE_OF_CHARACTERSTRING,
   'Message Text' = (optional, any valid message text),
   'Notify Type' = EVENT | ALARM,
   'AckRequired' = TRUE | FALSE,
   'From State' = NORMAL,

```
            'To State' =                OFFNORMAL,
            'Event Values' =            pMonitoredValue, pStatusFlags, pAlarmValues(matching list element)
6.      TRANSMIT BACnet-SimpleACK-PDU
7.      IF (Protocol_Revision is present AND Protocol_Revision e 13) THEN
            VERIFY pStatusFlags = (TRUE, FALSE,?,?)
8.      VERIFY pCurrentState = OFFNORMAL
9.      IF (Protocol_Revision is present AND Protocol_Revision ≥ 1) THEN
            VERIFY Event_Time_Stamps = (Toffnormal, *, *)
10.     IF (pAlarmValues has more than 1 entry) THEN {
11.         IF (pMonitoredValue is writable) THEN
                WRITE pMonitoredValue = (a value from pAlarmValues not used in prior steps)
            ELSE
                MAKE (pMonitoredValue have a value from pAlarmValues not used in prior steps)
12.         WAIT (pTimeDelay)
13.         BEFORE Notification Fail Time
                RECEIVE ConfirmedEventNotification-Request,
                    'Process Identifier' =          (any valid process ID),
                    'Initiating Device Identifier' = IUT,
                    'Event Object Identifier' =     (the object being tested),
                    'Time Stamp' =                  (Toffnormal: any valid time stamp),
                    'Notification Class' =          (the configured notification class),
                    'Priority' =                    (the value configured to correspond to a TO-OFFNORMAL
                                                     transition),
                    'Event Type' =                  CHANGE_OF_CHARACTERSTRING,
                    'Message Text' =                (optional, any valid message text),
                    'Notify Type' =                 EVENT | ALARM,
                    'AckRequired' =                 TRUE | FALSE,
                    'From State' =                  OFFNORMAL,
                    'To State' =                    OFFNORMAL,
                    'Event Values' =                pMonitoredValue, pStatusFlags, pAlarmValues(matching list
                                                     element)
14.         TRANSMIT BACnet-SimpleACK-PDU
15.         IF (Protocol_Revision is present AND Protocol_Revision e 13) THEN
                VERIFY pStatusFlags = (TRUE, FALSE,?,?)
16.         VERIFY pCurrentState = OFFNORMAL
17.         IF (Protocol_Revision is present AND Protocol_Revision ≥ 1) THEN
                VERIFY Event_Time_Stamps = (Toffnormal, *, *)
        }
18.     IF (pMonitoredValue is writable) THEN
            WRITE pMonitoredValue = (a value that corresponds to a NORMAL state)
        ELSE
            MAKE (pMonitoredValue have a value that corresponds to a NORMAL state)
19.     WAIT (pTimeDelayNormal)
20.     BEFORE Notification Fail Time
            RECEIVE ConfirmedEventNotification-Request,
                'Process Identifier' =          (any valid process ID),
                'Initiating Device Identifier' = IUT,
                'Event Object Identifier' =     (the object being tested),
                'Time Stamp' =                  (Tnormal: any valid time stamp),
                'Notification Class' =          (the configured notification class),
                'Priority' =                    (the value configured to correspond to a TO-NORMAL transition),
                'Event Type' =                  CHANGE_OF_CHARACTERSTRING,
                'Message Text' =                (optional, any valid message text),
                'Notify Type' =                 EVENT | ALARM,
                'AckRequired' =                 TRUE | FALSE,
                'From State' =                  OFFNORMAL,
```

```
                'To State' =                NORMAL,
                'Event Values' =            pMonitoredValue, pStatusFlags, pAlarmValues(matching element)
21.     TRANSMIT BACnet-SimpleACK-PDU
22.     IF (Protocol_Revision is present AND Protocol_Revision e 13) THEN
            VERIFY pStatusFlags = (FALSE, FALSE, ?, ?)
23.     VERIFY pCurrentState = NORMAL
24.     IF (Protocol_Revision is present AND Protocol_Revision ≥ 1) THEN
            VERIFY Event_Time_Stamps = (Toffnormal, *, Tnormal)
    }
25. IF ((Protocol_Revision is present AND Protocol_Revision d 12) AND (intrinsic reporting is being tested) AND (the
        intrinsic reporting object is configured with pFaultValues containing at least one characterstring)) THEN {
26.     IF (pMonitoredValue is writable) THEN
            WRITE pMonitoredValue = (a value from pFaultValues)
        ELSE
            MAKE (pMonitoredValue have a value from pFaultValues)
27.     BEFORE Notification Fail Time
            RECEIVE ConfirmedEventNotification-Request,
                'Process Identifier' =          (any valid process ID),
                'Initiating Device Identifier' = IUT,
                'Event Object Identifier' =     (the intrinsic reporting object being tested),
                'Time Stamp' =                  (Tfault: any valid time stamp),
                'Notification Class' =          (the configured notification class),
                'Priority' =                    (the value configured to correspond to a TO-FAULT transition),
                'Event Type' =                  CHANGE_OF_CHARACTERSTRING,
                'Message Text' =                (optional, any valid message text),
                'Notify Type' =                 EVENT | ALARM,
                'AckRequired' =                 TRUE | FALSE,
                'From State' =                  NORMAL,
                'To State' =                    FAULT,
                'Event Values' =                pMonitoredValue, pStatusFlags, (any characterstring)
28.     TRANSMIT BACnet-SimpleACK-PDU
29      VERIFY pCurrentState = FAULT
30.     IF (Protocol_Revision is present AND Protocol_Revision ≥ 1) THEN
            VERIFY Event_Time_Stamps = (Toffnormal, Tfault, Tnormal)
31.     VERIFY pCurrentReliability = MULTI_STATE_FAULT
32.     IF (pMonitoredValue is writable) THEN
            WRITE pMonitoredValue = (a value that corresponds to a NORMAL state)
        ELSE
            MAKE (pMonitoredValue have a value that corresponds to a NORMAL state)
33.     BEFORE Notification Fail Time
              RECEIVE ConfirmedEventNotification-Request,
                'Process Identifier' =          (any valid process ID),
                'Initiating Device Identifier' = IUT,
                'Event Object Identifier' =     (the intrinsic reporting object being tested),
                'Time Stamp' =                  (Tnormal: any valid time stamp),
                'Notification Class' =          (the configured notification class),
                'Priority' =                    (the value configured to correspond to a TO-NORMAL transition),
                'Event Type' =                  CHANGE_OF_CHARACTERSTRING,
                'Message Text' =                (optional, any valid message text),
                'Notify Type' =                 EVENT | ALARM,
                'AckRequired' =                 TRUE | FALSE,
                'From State' =                  FAULT,
                'To State' =                    NORMAL,
                'Event Values' =                pMonitoredValue, pStatusFlags, any characterstring
34.     TRANSMIT BACnet-SimpleACK-PDU
35.     VERIFY pCurrentState = NORMAL
```

36. IF (Protocol_Revision is present AND Protocol_Revision ≥ 1) THEN
VERIFY Event_Time_Stamps = (Toffnormal, Tfault, Tnormal)

}

Notes to Tester: The time stamps indicated by "*" in steps 9 and 17 can have a value that indicates an unspecified time or a time that precedes the timestamp of the first received notification. pCurrentReliability refers to the Reliability property of the event-generating object for this test.

[Add new **Clause 8.4.14**, p. 199]

8.4.14 UNSIGNED_RANGE Test (ConfirmedEventNotification Test)

Purpose: To verify the correct operation of the UNSIGNED_RANGE event algorithm.

Test Concept: This test is the same as 8.4.6, except that the Event_Type is UNSIGNED_RANGE instead of OUT_OF_RANGE, and there is no *p*Deadband. If pMonitoredValue is not under the tester's control in the IUT, then pHighLimit and/or pLowLimit are modified to generate event notifications. The object begins the test in a NORMAL state. pMonitoredValue is raised to a value that is above the high limit. After the time delay expires, the object should enter the HIGH_LIMIT state and transmit an event notification message. pMonitoredValue is lowered to a value that is below the high limit. After the time delay expires, the object should enter the NORMAL state and issue an event notification. The same process is repeated to test the low limit.

Configuration Requirements: If possible, the IUT shall be configured such that the Event_Enable property has a value of TRUE for the TO_OFFNORMAL and TO_NORMAL transitions. If possible, pLimitEnable shall have a value of TRUE for both HighLimit and LowLimit events. The 'Issue Confirmed Notifications' parameter in the Recipient_List of the configured Notification Class shall have a value of TRUE. The Recipient_List of the configured Notification Class shall contain the TD, thus ensuring that notifications are emitted. The event-generating objects shall be in a NORMAL state at the start of the test.

Test Steps:

1. VERIFY pCurrentState = NORMAL
2. IF (pMonitoredValue is writable) THEN
   WRITE pMonitoredValue = (a value x: (x > pHighLimit))
   ELSE
   MAKE (pMonitoredValue have a value x: (x > pHighLimit))
3. WAIT (pTimeDelay)
4. BEFORE **Notification Fail Time**
   RECEIVE ConfirmedEventNotification-Request,
   'Process Identifier' = (any valid process ID),
   'Initiating Device Identifier' = IUT,
   'Event Object Identifier' = (the object being tested),
   'Time Stamp' = (Toffnormal: any valid time stamp),
   'Notification Class' = (the configured notification class),
   'Priority' = (the value configured to correspond to a TO_OFFNORMAL transition),
   'Event Type' = UNSIGNED_RANGE,
   'Message Text' = (optional, any valid message text),
   'Notify Type' = EVENT | ALARM,
   'AckRequired' = TRUE | FALSE,
   'From State' = NORMAL,
   'To State' = HIGH_LIMIT,
   'Event Values' = pMonitoredValue, pStatusFlags, pHighLimit
5. TRANSMIT BACnet-SimpleACK-PDU
6. IF (Protocol_Revision is present AND Protocol_Revision ≥ 13)) THEN
   VERIFY pStatusFlags = (TRUE, FALSE, ?, ?)
7. VERIFY pCurrentState = HIGH_LIMIT
8. IF (Protocol_Revision is present AND Protocol_Revision ≥ 1) THEN

VERIFY Event_Time_Stamps = (Toffnormal, *, *)

9. IF (pMonitoredValue is writable) THEN
   WRITE pMonitoredValue = (a value x: (pLowLimit < x < pHighLimit))
   ELSE
   MAKE (pMonitoredValue have a value x: (pLowLimit < x < pHighLimit))
10. WAIT (pTimeDelayNormal)
11. BEFORE **Notification Fail Time**
    RECEIVE ConfirmedEventNotification-Request,
    'Process Identifier' = (any valid process ID),
    'Initiating Device Identifier' = IUT,
    'Event Object Identifier' = (the object being tested),
    'Time Stamp' = (Tnormal: any valid time stamp),
    'Notification Class' = (the configured notification class),
    'Priority' = (the value configured to correspond to a TO_NORMAL transition),
    'Event Type' = UNSIGNED_RANGE,
    'Message Text' = (optional, any valid message text),
    'Notify Type' = EVENT | ALARM,
    'AckRequired' = TRUE | FALSE,
    'From State' = HIGH_LIMIT,
    'To State' = NORMAL,
    'Event Values' = pMonitoredValue, pStatusFlags, pHighLimit
12. TRANSMIT BACnet-SimpleACK-PDU
13. IF (Protocol_Revision is present AND Protocol_Revision ≥ 13)) THEN
    VERIFY pStatusFlags = (FALSE, FALSE, ?, ?)
14. VERIFY pCurrentState = NORMAL
15. IF (Protocol_Revision is present AND Protocol_Revision ≥ 1) THEN
    VERIFY Event_Time_Stamps = (Toffnormal, *, Tnormal)
16. IF (pMonitoredValue is writable) THEN
    WRITE pMonitoredValue = (a value x: (x < pLowLimit))
    ELSE
    MAKE (pMonitoredValue have a value x: (x < pLowLimit))
17. WAIT (pTimeDelay)
18. BEFORE **Notification Fail Time**
    RECEIVE ConfirmedEventNotification-Request,
    'Process Identifier' = (any valid process ID),
    'Initiating Device Identifier' = IUT,
    'Event Object Identifier' = (the object being tested),
    'Time Stamp' = (Tlowlimit: any valid time stamp),
    'Notification Class' = (the configured notification class),
    'Priority' = (the value configured to correspond to a TO_OFFNORMAL transition),
    'Event Type' = UNSIGNED_RANGE,
    'Message Text' = (optional, any valid message text),
    'Notify Type' = EVENT | ALARM,
    'AckRequired' = TRUE | FALSE,
    'From State' = NORMAL,
    'To State' = LOW_LIMIT,
    'Event Values' = pMonitoredValue, pStatusFlags, pLowLimit
19. TRANSMIT BACnet-SimpleACK-PDU
20. IF (Protocol_Revision is present AND Protocol_Revision ≥ 13)) THEN
    VERIFY pStatusFlags = (TRUE, FALSE, ?, ?)
21. VERIFY pCurrentState = LOW_LIMIT
22. IF (Protocol_Revision is present AND Protocol_Revision ≥ 1) THEN
    VERIFY Event_Time_Stamps = (Tlowlimit, *, Tnormal)
23. IF (pMonitoredValue is writable) THEN
    WRITE pMonitoredValue = (a value x: (Low_Limit < x < High_Limit))
    ELSE

MAKE (pMonitoredValue have a value x: (Low_Limit < x < High_Limit))
24. WAIT (pTimeDelayNormal)
25. BEFORE **Notification Fail Time**
RECEIVE ConfirmedEventNotification-Request,
'Process Identifier' = (any valid process ID),
'Initiating Device Identifier' = IUT,
'Event Object Identifier' = (the object being tested),
'Time Stamp' = (Tlowtonormal: any valid time stamp),
'Notification Class' = (the configured notification class),
'Priority' = (the value configured to correspond to a TO_NORMAL transition),
'Event Type' = UNSIGNED_RANGE,
'Message Text' = (optional, any valid message text),
'Notify Type' = EVENT | ALARM,
'AckRequired' = TRUE | FALSE,
'From State' = LOW_LIMIT,
'To State' = NORMAL,
'Event Values' = pMonitoredValue, pStatusFlags, pLowLimit
26. TRANSMIT BACnet-SimpleACK-PDU
27. IF (Protocol_Revision is present AND Protocol_Revision ≥ 13)) THEN
VERIFY pStatusFlags = (FALSE, FALSE, ?, ?)
28. VERIFY pCurrentState = NORMAL
29. IF (Protocol_Revision is present AND Protocol_Revision ≥ 1) THEN
VERIFY Event_Time_Stamps = (Tlowlimit, *, Tlowtonormal)

Notes to Tester: The time stamps indicated by "*" can have a value that indicates an unspecified time or a time that precedes the timestamp of the first received notification.

[Add new **Clause 8.4.15**, p. 199]
[Reviewer Note: New event algorithm for Protocol_Revision 11.]

8.4.15 CHANGE_OF_STATUS_FLAGS Test (ConfirmedEventNotification)

Purpose: To verify the correct operation of the CHANGE_OF_STATUS_FLAGS event algorithm.

Test Concept: The object, O1, begins the test in a NORMAL state. pMonitoredValue is changed such that a logical AND of pMonitoredValue and pSelectedFlags results in at least one bits set. After pTimeDelay expires, the object shall enter the OFFNORMAL state and transmit an event notification message. pMonitoredValue is then changed such that a logical AND of pMonitoredValue and pSelectedFlags results in no bits set. After pTimeDelayNormal expires, the object shall enter the NORMAL state and transmit an event notification message.

Configuration Requirements: O1 shall be configured such that the Event_Enable property has a value of TRUE for the TO-OFFNORMAL and TO-NORMAL transitions. The 'Issue Confirmed Notifications' parameter in the Recipient_List of the configured Notification Class shall have a value of TRUE. The Recipient_List of the configured Notification Class shall contain the TD. The event-generating object shall be in a NORMAL state at the start of the test.

Test Steps:

1. VERIFY pCurrentState = NORMAL
2. MAKE (pMonitoredValue AND pSelectedFlags <> {FALSE, FALSE, FALSE, FALSE})
3. WAIT (pTimeDelay)
4. BEFORE **Notification Fail Time**
RECEIVE ConfirmedEventNotification-Request,
'Process Identifier' = (any valid process ID),
'Initiating Device Identifier' = IUT,

'Event Object Identifier' = O1,
'Time Stamp' = (any valid time stamp),
'Notification Class' = (the notification class configured for O1),
'Priority' = (the value configured for the transition),
'Event Type' = CHANGE_OF_STATUS_FLAGS,
'Message Text' = (optional, any valid message text),
'Notify Type' = EVENT | ALARM,
'AckRequired' = TRUE | FALSE,
'From State' = NORMAL,
'To State' = OFFNORMAL,
'Event Values' = (optional, pPresentValue), pMonitoredValue

5. TRANSMIT BACnet-SimpleACK-PDU
6. IF (Protocol_Revision is present AND Protocol_Revision e 13) THEN
   VERIFY pStatusFlags = {TRUE, FALSE,?,?}
7. VERIFY pCurrentState = OFFNORMAL
8. MAKE (pMonitoredValue AND pSelectedFlags = {FALSE, FALSE, FALSE, FALSE})
9. WAIT (pTimeDelayNormal)
10. BEFORE **Notification Fail Time**
    RECEIVE ConfirmedEventNotification-Request,
    'Process Identifier' = (any valid process ID),
    'Initiating Device Identifier' = IUT,
    'Event Object Identifier' = O1
    'Time Stamp' = (any valid time stamp),
    'Notification Class' = (the notification class configured for O1),
    'Priority' = (the value configured for the transition),
    'Event Type' = CHANGE_OF_STATUS_FLAGS,
    'Message Text' = (optional, any valid message text),
    'Notify Type' = EVENT | ALARM,
    'AckRequired' = TRUE | FALSE,
    'From State' = OFFNORMAL,
    'To State' = NORMAL,
    'Event Values' = (optional, pPresentValue), pMonitoredValue
11. TRANSMIT BACnet-SimpleACK-PDU
12. IF (Protocol_Revision is present AND Protocol_Revision e 13) THEN
    VERIFY pStatusFlags = {FALSE, FALSE, ?, ?}
13. VERIFY pCurrentState = NORMAL

[Add new **Clause 8.4.16**, p. 199]

8.4.16 Proprietary Algorithms Test (ConfirmedEventNotifications)

Purpose: This test verifies the correct generation of ConfirmedEventNotification messages conveying a notification of the complex form. This applies to any non-Event Enrollment object that uses a proprietary event algorithm.

Test Concept: The event generating object begins the test in a NORMAL state. The event generating object is made to transition to any state by any means necessary. The resulting ConfirmedEventNotification message is received and verified.

Configuration Requirements: The IUT shall be configured such that the Event_Enable property has a value of TRUE for whichever transition shall be used for the test. The 'Issue Confirmed Notifications' parameter shall have a value of TRUE. D1 is a vendor specific delay (may be zero). The event-generating object shall be in a NORMAL state at the start of the test.

Test Steps:

1. VERIFY pCurrentState = NORMAL
2. MAKE (a condition exist that will cause the event generating object to transition)

3. WAIT D1
4. BEFORE **Notification Fail Time**
   RECEIVE ConfirmedEventNotification-Request,
   'Process Identifier' = (any valid process ID),
   'Initiating Device Identifier' = IUT,
   'Event Object Identifier' = (the event generating object being tested),
   'Time Stamp' = (any valid time stamp),
   'Notification Class' = (the configured notification class),
   'Priority' = (the value configured for this transition),
   'Event Type' = (the proprietary event type specified by the vendor),
   'Message Text' = (optional, any valid message text),
   'Notify Type' = EVENT | ALARM,
   'AckRequired' = TRUE | FALSE,
   'From State' = NORMAL,
   'To State' = (the state the object was made to transition to),
   'Event Values' = (a list of BACnetPropertyValue reporting the set of property values as defined by the vendor)
5. TRANSMIT BACnet-SimpleACK-PDU

[Add new **Clause 8.4.17**, p. 199]

8.4.17 CHANGE_OF_RELIABILITY Tests (ConfirmedEventNotifications)

8.4.17.1 CHANGE_OF_RELIABILITY with the NONE fault Algorithm (ConfirmedEventNotifications)

Purpose: To verify the correct operation of the NONE fault algorithm.

Test Concept: The test concept corresponds to 8.5.17.1

Configuration Requirements: The configuration requirements are identical to those in 8.5.17.1, except that the 'Issue Confirmed Notifications' parameter shall have a value of TRUE.

Test Steps: The test steps for this test case are identical to the test steps in 8.5.17.1, except that the UnconfirmedEventNotification requests are ConfirmedEventNotification requests and the TD acknowledges receiving the notifications.

Notes to Tester: The passing results for this test case are identical to the ones in 8.5.17.1, except that the event notifications shall be conveyed using a ConfirmedEventNotification service request.

8.4.17.2 CHANGE_OF_RELIABILITY with the FAULT_CHARACTERSTRING Algorithm (ConfirmedEventNotifications)

Purpose: To verify the correct operation of the FAULT_CHARACTERSTRING fault algorithm.

Test Concept: The test concept corresponds to 8.5.17.2

Configuration Requirements: The configuration requirements are identical to those in 8.5.17.2, except that the 'Issue Confirmed Notifications' parameter shall have a value of TRUE.

Test Steps: The test steps for this test case are identical to the test steps in 8.5.17.2, except that the UnconfirmedEventNotification requests are ConfirmedEventNotification requests and the TD acknowledges receiving the notifications.

Notes to Tester: The passing results for this test case are identical to the ones in 8.5.17.2, except that the event notifications shall be conveyed using a ConfirmedEventNotification service request.

8.4.17.3 CHANGE_OF_RELIABILITY with the FAULT_EXTENDED Algorithm (ConfirmedEventNotifications)

Purpose: To verify the correct operation of the FAULT_EXTENDED fault algorithm.

Test Concept: The test concept corresponds to 8.5.17.3

Configuration Requirements: The configuration requirements are identical to those in 8.5.17.3, except that the 'Issue Confirmed Notifications' parameter shall have a value of TRUE.

Test Steps: The test steps for this test case are identical to the test steps in 8.5.17.3, except that the UnconfirmedEventNotification requests are ConfirmedEventNotification requests and the TD acknowledges receiving the notifications.

Notes to Tester: The passing results for this test case are identical to the ones in 8.5.17.3, except that the event notifications shall be conveyed using a ConfirmedEventNotification service request.

8.4.17.4 CHANGE_OF_RELIABILITY with the FAULT_LIFE_SAFETY Algorithm (ConfirmedEventNotifications)

Purpose: To verify the correct operation of the FAULT_LIFE_SAFETY fault algorithm.

Test Concept: The test concept corresponds to 8.5.17.4

Configuration Requirements: The configuration requirements are identical to those in 8.5.17.4, except that the 'Issue Confirmed Notifications' parameter shall have a value of TRUE.

Test Steps: The test steps for this test case are identical to the test steps in 8.5.17.4, except that the UnconfirmedEventNotification requests are ConfirmedEventNotification requests and the TD acknowledges receiving the notifications.

Notes to Tester: The passing results for this test case are identical to the ones in 8.5.17.4, except that the event notifications shall be conveyed using a ConfirmedEventNotification service request.

8.4.17.5 CHANGE_OF_RELIABILITY with the FAULT_STATE Algorithm (ConfirmedEventNotifications)

Purpose: To verify the correct operation of the FAULT_STATE fault algorithm.

Test Concept: The test concept corresponds to 8.5.17.5

Configuration Requirements: The configuration requirements are identical to those in 8.5.17.5, except that the 'Issue Confirmed Notifications' parameter shall have a value of TRUE.

Test Steps: The test steps for this test case are identical to the test steps in 8.5.17.5, except that the UnconfirmedEventNotification requests are ConfirmedEventNotification requests and the TD acknowledges receiving the notifications.

Notes to Tester: The passing results for this test case are identical to the ones in 8.5.17.5, except that the event notifications shall be conveyed using a ConfirmedEventNotification service request.

8.4.17.6 CHANGE_OF_RELIABILITY with the FAULT_STATUS_FLAGS Algorithm (ConfirmedEventNotifications)

Purpose: To verify the correct operation of the FAULT_STATUS_FLAGS fault algorithm.

Test Concept: The test concept corresponds to 8.5.17.6

Configuration Requirements: The configuration requirements are identical to those in 8.5.17.6, except that the 'Issue Confirmed Notifications' parameter shall have a value of TRUE.

Test Steps: The test steps for this test case are identical to the test steps in 8.5.17.6, except that the UnconfirmedEventNotification requests are ConfirmedEventNotification requests and the TD acknowledges receiving the notifications.

Notes to Tester: The passing results for this test case are identical to the ones in 8.5.17.6, except that the event notifications shall be conveyed using a ConfirmedEventNotification service request.

8.4.17.7 CHANGE_OF_RELIABILITY for Event Enrollment Fault Condition Precedence (ConfirmedEventNotifications)

Purpose: To verify that the Event Enrollment object's fault detection gives precedence to faults in the Event Enrollment object, then faults in the monitored object, and finally faults detected by the configured Fault algorithm.

Test Concept: The test concept corresponds to 8.5.17.7

Configuration Requirements: The configuration requirements are identical to those in 8.5.17.7, except that the 'Issue Confirmed Notifications' parameter shall have a value of TRUE.

Test Steps: The test steps for this test case are identical to the test steps in 8.5.17.7, except that the UnconfirmedEventNotification requests are ConfirmedEventNotification requests and the TD acknowledges receiving the notifications.

Notes to Tester: The passing results for this test case are identical to the ones in 8.5.17.7, except that the event notifications shall be conveyed using a ConfirmedEventNotification service request.

8.4.17.8 CHANGE_OF_RELIABILITY of Event Enrollment Object, Monitored Object Fault (ConfirmedEventNotifications)

Purpose: To verify the proper operation of the Event Enrollment object's fault detection when the monitored object enters the fault state.

Test Concept: The test concept corresponds to 8.5.17.8

Configuration Requirements: The configuration requirements are identical to those in 8.5.17.8, except that the 'Issue Confirmed Notifications' parameter shall have a value of TRUE.

Test Steps: The test steps for this test case are identical to the test steps in 8.5.17.8, except that the UnconfirmedEventNotification requests are ConfirmedEventNotification requests and the TD acknowledges receiving the notifications.

Notes to Tester: The passing results for this test case are identical to the ones in 8.5.17.8, except that the event notifications shall be conveyed using a ConfirmedEventNotification service request.

8.4.17.9 CHANGE_OF_RELIABILITY of Event Enrollment Object Fault (ConfirmedEventNotifications)

Purpose: To verify the Event Enrollment object generates a fault event when the object enters into fault due to an internal unreliable operation.

Test Concept: The test concept corresponds to 8.5.17.9

Configuration Requirements: The configuration requirements are identical to those in 8.5.17.9, except that the 'Issue Confirmed Notifications' parameter shall have a value of TRUE.

Test Steps: The test steps for this test case are identical to the test steps in 8.5.17.9, except that the UnconfirmedEventNotification requests are ConfirmedEventNotification requests and the TD acknowledges receiving the notifications.

Notes to Tester: The passing results for this test case are identical to the ones in 8.5.17.9, except that the event notifications shall be conveyed using a ConfirmedEventNotification service request.

8.4.17.10 After FAULT-to-NORMAL, Re-Notification of OFFNORMAL (ConfirmedEventNotifications)

Purpose: To verify that objects go to the NORMAL state after leaving the FAULT state, then transition to OFFNORMAL if the condition still exists.

Test Concept: The test concept corresponds to 8.5.17.10

Configuration Requirements: The configuration requirements are identical to those in 8.5.17.10, except that the 'Issue Confirmed Notifications' parameter shall have a value of TRUE.

Test Steps: The test steps for this test case are identical to the test steps in 8.5.17.10, except that the UnconfirmedEventNotification requests are ConfirmedEventNotification requests and the TD acknowledges receiving the notifications.

Notes to Tester: The passing results for this test case are identical to the ones in 8.5.17.10, except that the event notifications shall be conveyed using a ConfirmedEventNotification service request.

8.4.17.11 CHANGE_OF_RELIABILITY with Internal Object Fault (ConfirmedEventNotifications)

Purpose: To verify that fault conditions, unrelated to fault algorithms, are detected and reported.

Test Concept: The test concept corresponds to 8.5.17.11

Configuration Requirements: The configuration requirements are identical to those in 8.5.17.11, except that the 'Issue Confirmed Notifications' parameter shall have a value of TRUE.

Test Steps: The test steps for this test case are identical to the test steps in 8.5.17.11, except that the UnconfirmedEventNotification requests are ConfirmedEventNotification requests and the TD acknowledges receiving the notifications.

Notes to Tester: The passing results for this test case are identical to the ones in 8.5.17.11, except that the event notifications shall be conveyed using a ConfirmedEventNotification service request.

[Change **Clause 8.5**, p. 199]

8.5 UnconfirmedEventNotification Service Initiation Tests

This clause defines the tests necessary to demonstrate support for initiating UnconfirmedEventNotification service requests. The UnconfirmedEventNotification tests are specific to the event detection algorithm used. For each object type that supports intrinsic event reporting, the IUT shall pass the tests for the event detection algorithm that applies to that object type. If the IUT supports the Event Enrollment object, it shall pass the tests for the event detection algorithm that corresponds to each event type supported.

For devices that implement a read-only Recipient_List property for all instances of Notification Class objects, and are exclusively configured for UnconfirmedEventNotification service initiation, and the device can not contain a Notification Forwarder object, the Recipient_List property is required to have a single entry with the recipient value corresponding to a local broadcast that is effective for all days, all times, and all transitions, and shall issue unconfirmed notifications using Process Identifier 0 exclusively.

[Change **Clause 8.5.1**, p. 199]

8.5.1 CHANGE_OF_BITSTRING Tests ~~**(UnconfirmedEventNotification)**~~

~~Dependencies: ReadProperty Service Execution Tests, 9.18; WriteProperty Service Execution Tests, 9.22.~~

~~BACnet Reference Clauses: 12.12, 13.3.1, and 13.9.~~

Purpose: To verify the correct operation of the ~~Change of Bitstring~~*CHANGE_OF_BITSTRING* event algorithm.~~ This test applies to Event Enrollment objects with an Event_Type of CHANGE_OF_BITSTRING.~~

Test Concept: The test concept corresponds to 8.4.1.

Configuration Requirements: ~~The IUT shall be configured such that the Event_Enable property has a value of TRUE for the TO-OFFNORMAL and TO-NORMAL transitions. The~~ *The configuration requirements are identical to those in 8.4.1, except that the* ~~Issue_Confirmed_Notifications property~~*'Issue Confirmed Notifications' parameter* shall have a value of FALSE.~~ The event-generating objects shall be in a NORMAL state at the start of the test.~~

Test Steps: The test steps for this test case are identical to the test steps in 8.4.1, except that the ConfirmedEventNotification requests are UnconfirmedEventNotification requests and the TD does not acknowledge receiving the notifications.

Notes to Tester: The passing results for this test case are identical to the ones in 8.4.1, except that the event notifications shall be conveyed using an UnconfirmedEventNotification service request. The MAC address used for these messages shall be either a broadcast that reaches the local network of the TD or the MAC address of the TD.

[Change **Clause 8.5.2**, p. 199]

8.5.2 CHANGE_OF_STATE Tests ~~**(UnconfirmedEventNotification)**~~

~~Dependencies: ReadProperty Service Execution Tests, 9.18; WriteProperty Service Execution Tests, 9.22.~~

~~BACnet Reference Clauses: 12.6, 12.8, 12.18, 12.20, 13.2,13.3.2, and 13.9.~~

Purpose: To verify the correct operation of the CHANGE_OF_STATE event algorithm~~. This test applies to Event Enrollment objects with an Event_Type of CHANGE_OF_STATE and to intrinsic event reporting for Binary Input, Binary Value, Multi-state Input and Multi-state Value objects.~~

Test Concept: The test concept corresponds to 8.4.2.

Configuration Requirements: ~~The IUT shall be configured such that the Event_Enable property has a value of TRUE for the TO-OFFNORMAL and TO-NORMAL transitions. The~~ *The configuration requirements are identical to those in 8.4.2, except that the* ~~Issue_Confirmed_Notifications property~~*'Issue Confirmed Notifications' parameter* shall have a value of FALSE.~~The event-generating objects shall be in a NORMAL state at the start of the test.~~

Test Steps: The test steps for this test case are identical to the test steps in 8.4.2, except that the ConfirmedEventNotification requests are UnconfirmedEventNotification requests and the TD does not acknowledge receiving the notifications.

Notes to Tester: The passing results for this test case are identical to the ones in 8.4.2, except that the event notifications shall be conveyed using an UnconfirmedEventNotification service request. The MAC address used for these messages shall be either a broadcast that reaches the local network of the TD or the MAC address of the TD.

[Change **Clause 8.5.3**, p. 200]

8.5.3 CHANGE_OF_VALUE Tests *(UnconfirmedEventNotification)*

This clause defines the tests necessary to demonstrate support for the CHANGE_OF_VALUE event algorithm. The CHANGE_OF_VALUE algorithm can be applied to both numerical and bitstring datatypes. The IUT shall pass the tests for both applications.

8.5.3.1 Numerical Algorithm *(UnconfirmedEventNotification)*

~~The test in this clause applies to use of the CHANGE_OF_VALUE algorithm applied to Integer or Real datatypes.~~

~~Dependencies: ReadProperty Service Execution Tests, 9.18; WriteProperty Service Execution Tests, 9.22.~~

~~BACnet Reference Clauses: 12.12, 13.3.3, and 13.9.~~

Purpose: To verify the correct operation of the CHANGE_OF_VALUE event algorithm *applied to Integer or Real datatypes*~~as applied to numerical datatypes. This test applies to Event Enrollment objects with an Event_Type of CHANGE_OF_VALUE.~~

Test Concept: The test concept corresponds to 8.4.3.1.

Configuration Requirements: ~~The IUT shall be configured such that the Event_Enable property has a value of TRUE for the TO-NORMAL transition. The~~ *The configuration requirements are identical to those in 8.4.3.1, except that the* ~~Issue_Confirmed_Notifications property~~*'Issue Confirmed Notifications' parameter* shall have a value of FALSE.~~The event-generating object shall be in a NORMAL state at the start of the test.~~

Test Steps: The test steps for this test case are identical to the test steps in 8.4.3.1, except that the ConfirmedEventNotification requests are UnconfirmedEventNotification requests and the TD does not acknowledge receiving the notifications.

Notes to Tester: The passing results for this test case are identical to the ones in 8.4.3.1, except that the event notifications shall be conveyed using an UnconfirmedEventNotification service request. The MAC address used for these messages shall be either a broadcast that reaches the local network of the TD or the MAC address of the TD.

8.5.3.2 Bitstring Algorithm *(UnconfirmedEventNotification)*

~~The test in this clause applies to use of the CHANGE_OF_VALUE algorithm applied to Bitstring datatypes.~~

~~Dependencies: ReadProperty Service Execution Tests, 9.18; WriteProperty Service Execution Tests, 9.22.~~

~~BACnet Reference Clauses: 12.12, 13.3.3, and 13.9.~~

Purpose: To verify the correct operation of the CHANGE_OF_STATE event algorithm *applied to Bitstring datatypes*. ~~This test applies to Event Enrollment objects with an Event_Type of CHANGE_OF_VALUE.~~

Test Concept: The test concept corresponds to 8.4.3.2.

Configuration Requirements: ~~The IUT shall be configured such that the Event_Enable property has a value of TRUE for the TO-OFFNORMAL and TO-NORMAL transitions. The~~ *The configuration requirements are identical to those in 8.4.3.2, except that the* ~~Issue_Confirmed_Notifications property~~*'Issue Confirmed Notifications' parameter* shall have a value of FALSE. ~~The event-generating objects shall be in a NORMAL state at the start of the test.~~

Test Steps: The test steps for this test case are identical to the test steps in 8.4.3.2, except that the ConfirmedEventNotification requests are UnconfirmedEventNotification requests and the TD does not acknowledge receiving the notifications.

Notes to Tester: The passing results for this test case are identical to the ones in 8.4.3.2, except that the event notifications shall be conveyed using an UnconfirmedEventNotification service request. The MAC address used for these messages shall be either a broadcast that reaches the local network of the TD or the MAC address of the TD.

[Change **Clause 8.5.4**, p. 201]

8.5.4 COMMAND_FAILURE Tests ***(UnconfirmedEventNotification)***

~~Dependencies: ReadProperty Service Execution Tests, 9.18; WriteProperty Service Execution Tests, 9.22.~~

~~BACnet Reference Clauses: 12.7, 12.12, 12.19 13.2, 13.3.4, and 13.9.~~

Purpose: To verify the correct operation of the COMMAND_FAILURE algorithm. ~~It applies to Event Enrollment objects with an Event_Type of COMMAND_FAILURE and to intrinsic event reporting for Binary Output and Multi-State Output objects.~~

Test Concept: The test concept corresponds to 8.4.4.

Configuration Requirements: ~~The IUT shall be configured such that the Event_Enable property has a value of TRUE for the TO-OFFNORMAL and TO-NORMAL transitions. The~~ *The configuration requirements are identical to those in 8.4.4, except that the* ~~Issue_Confirmed_Notifications property~~*'Issue Confirmed Notifications' parameter* shall have a value of FALSE. ~~The event-generating object shall be in a NORMAL state at the start of the test. The Feedback_Value property shall be decoupled from the input signal that is normally used to verify the output so that it can be independently manipulated.~~

Test Steps: The test steps for this test case are identical to the test steps in 8.4.4, except that the ConfirmedEventNotification requests are UnconfirmedEventNotification requests and the TD does not acknowledge receiving the notifications.

Notes to Tester: The passing results for this test case are identical to the ones in 8.4.4, except that the event notifications shall be conveyed using an UnconfirmedEventNotification service request. The MAC address used for these messages shall be either a broadcast that reaches the local network of the TD or the MAC address of the TD.

[Change **Clause 8.5.5**, p. 201]

8.5.5 FLOATING_LIMIT Tests ***(UnconfirmedEventNotification)***

~~Dependencies: ReadProperty Service Execution Tests, 9.18; WriteProperty Service Execution Tests, 9.22.~~

~~BACnet Reference Clauses: 12.12, 12.17, 13.2, 13.3.5, and 13.9.~~

Purpose: To verify the correct operation of the Floating Limit event algorithm. ~~This test applies to Event Enrollment objects with an Event_Type of FLOATING_LIMIT and to Loop objects that support intrinsic reporting. When testing Loop objects both High_Diff_Limit and Low_Diff_Limit shall be replaced by Error_Limit in the test description below~~

Test Concept: The test concept corresponds to 8.4.5.

Configuration Requirements: ~~The IUT shall be configured such that the Event_Enable property has a value of TRUE for the TO-OFFNORMAL and TO-NORMAL transitions. The~~ *The configuration requirements are identical to those in 8.4.5, except that the* ~~Issue_Confirmed_Notifications property~~*'Issue Confirmed Notifications' parameter* shall have a value of FALSE.~~The event-generating objects shall be in a NORMAL state at the start of the test.~~

Test Steps: The test steps for this test case are identical to the test steps in 8.4.5, except that the ConfirmedEventNotification requests are UnconfirmedEventNotification requests and the TD does not acknowledge receiving the notifications.

Notes to Tester: The passing results for this test case are identical to the ones in 8.4.5, except that the event notifications shall be conveyed using an UnconfirmedEventNotification service request. The MAC address used for these messages shall be either a broadcast that reaches the local network of the TD or the MAC address of the TD.

[Change **Clause 8.5.6**, p. 201]

8.5.6 OUT_OF_RANGE Tests ***(UnconfirmedEventNotification)***

~~Dependencies: ReadProperty Service Execution Tests, 9.18; WriteProperty Service Execution Tests, 9.22.~~

~~BACnet Reference Clauses: 12.2, 12.3, 12.4, 12.12, 13.2, 13.3.6, and 13.9.~~

Purpose: To verify the correct operation of the OUT_OF_RANGE event algorithm.~~This test applies to Event Enrollment objects with an Event_Type of OUT_OF_RANGE and to intrinsic event reporting for Analog Input, Analog Output, and Analog Value objects.~~

Test Concept: The test concept corresponds to 8.4.6.

Configuration Requirements: ~~The IUT shall be configured such that the Event_Enable property has a value of TRUE for the TO-OFFNORMAL and TO-NORMAL transitions. For objects using intrinsic reporting the Limit_Enable property shall have a value of TRUE for both HighLimit and LowLimit events. The~~ *The configuration requirements are identical to those in 8.4.6, except that the* ~~Issue_Confirmed_Notifications property~~*'Issue Confirmed Notifications' parameter* shall have a value of FALSE.~~The event-generating objects shall be in a NORMAL state at the start of the test.~~

Test Steps: The test steps for this test case are identical to the test steps in 8.4.6, except that the ConfirmedEventNotification requests are UnconfirmedEventNotification requests and the TD does not acknowledge receiving the notifications.

Notes to Tester: The passing results for this test case are identical to the ones in 8.4.6, except that the event notifications shall be conveyed using an UnconfirmedEventNotification service request. The MAC address used for these messages shall be either a broadcast that reaches the local network of the TD or the MAC address of the TD.

[Change **Clause 8.5.7**, p. 202]

8.5.7 BUFFER_READY Tests ***(UnconfirmedEventNotification)***

~~Dependencies: ReadProperty Service Execution Tests, 9.18.~~

~~BACnet Reference Clauses: 12.12, 12.25, 13.2, 13.3.7, and 13.9.~~

Purpose: To verify the correct operation of the BUFFER_READY event algorithm.~~This test applies to logging objects that support intrinsic notification and to Event Enrollment objects with an Event_Type of BUFFER_READY.~~

Test Concept: The test concept corresponds to 8.4.7.

Configuration Requirements: The configuration requirements are identical to those in 8.4.7, except that the 'Issue Confirmed Notifications' parameter shall have a value of FALSE.

~~The test concept, configuration requirements and test steps are identical to those in 8.4.7, except that the 'Issue Confirmed Notifications' parameter shall have a value of FALSE, UnconfirmedEventNotification requests are issued instead of ConfirmedEventNotification requests and the TD does not acknowledge receiving the notifications.~~

Test Steps: The test steps for this test case are identical to the test steps in 8.4.7, except that the ConfirmedEventNotification requests are UnconfirmedEventNotification requests and the TD does not acknowledge receiving the notifications.

Notes to Tester: The passing results for this test case are identical to the ones in 8.4.7, except that the event notifications shall be conveyed using an UnconfirmedEventNotification service request. The MAC address used for these messages shall be either a broadcast that reaches the local network of the TD or the MAC address of the TD.

[Change **Clause 8.5.8**, p. 202]

8.5.8 CHANGE_OF_LIFE_SAFETY Tests *(UnconfirmedEventNotification)*

8.5.8.1 NORMAL to OFFNORMAL Transition Test

~~Dependencies: ReadProperty Service Execution Tests, 9.18; WriteProperty Service Execution Tests, 9.22.~~

~~BACnet Reference Clauses: 12.12.7, 12.15, 12.16, 13.3.8, 13.13 and Figure 13-9.~~

Purpose: This test case verifies the correct operation of the CHANGE_OF_LIFE_SAFETY event algorithm for objects transitioning ~~between~~ *from* NORMAL ~~and~~*to* OFFNORMAL event states. ~~It applies to Event Enrollment objects with an Event_Type of CHANGE_OF_LIFE_SAFETY and to intrinsic event reporting for Life Safety Point and Life Safety Zone objects.~~

The test concept, configuration requirements and test steps are identical to those in 8.4.8.1, except that the ~~Issue_Confirmed_Notifications property~~*Issue Confirmed Notifications parameter* shall have a value of FALSE, UnconfirmedEventNotification requests are issued instead of ConfirmedEventNotification requests and the TD does not acknowledge receiving the notifications. The MAC address used for these messages shall be either a broadcast that reaches the local network of the TD or the MAC address of the TD.

8.5.8.2 OFFNORMAL to NORMAL Transition Test

~~Dependencies: ReadProperty Service Execution Tests, 9.18; WriteProperty Service Execution Tests, 9.22.~~

~~BACnet Reference Clauses: 12.12.7, 12.15, 12.16, 13.3.8, 13.9 and Figure 13-9.~~

Purpose: This test case verifies the correct operation of the CHANGE_OF_LIFE_SAFETY event algorithm for objects transitioning ~~between~~ *from* OFFNORMAL ~~and~~*to* NORMAL event states. ~~It applies to Event Enrollment objects with an Event_Type of CHANGE_OF_LIFE_SAFETY and to intrinsic event reporting for Life Safety Point and Life Safety Zone objects.~~

The test concept, configuration requirements and test steps are identical to those in 8.4.8.2, except that the ~~Issue_Confirmed_Notifications property~~*Issue Confirmed Notifications parameter* shall have a value of FALSE, UnconfirmedEventNotification requests are issued instead of ConfirmedEventNotification requests and the TD does not acknowledge receiving the notifications.

8.5.8.3 NORMAL to LIFE_SAFETY_ALARM Transition Test

~~Dependencies: ReadProperty Service Execution Tests, 9.18; WriteProperty Service Execution Tests, 9.22.~~

~~BACnet Reference Clauses: 12.12.7, 12.15, 12.16, 13.3.8, 13.9 and Figure 13-9.~~

Purpose: This test case verifies the correct operation of the CHANGE_OF_LIFE_SAFETY event algorithm for objects transitioning ~~between~~ *from* NORMAL ~~and~~*to* LIFE_SAFETY_ALARM event states. ~~It applies to Event Enrollment objects~~

~~with an Event_Type of CHANGE_OF_LIFE_SAFETY and to intrinsic event reporting for Life Safety Point and Life Safety Zone objects.~~

The test concept, configuration requirements and test steps are identical to those in 8.4.8.3, except that the ~~Issue_Confirmed_Notifications property~~*Issue Confirmed Notifications parameter* shall have a value of FALSE, UnconfirmedEventNotification requests are issued instead of ConfirmedEventNotification requests and the TD does not acknowledge receiving the notifications.

8.5.8.4 LIFE_SAFETY_ALARM to NORMAL Transition Test

~~Dependencies: ReadProperty Service Execution Tests, 9.18; WriteProperty Service Execution Tests, 9.22.~~

~~BACnet Reference Clauses: 12.12.7, 12.15, 12.16, 13.3.8, 13.9 and Figure 13-9.~~

Purpose: This test case verifies the correct operation of the CHANGE_OF_LIFE_SAFETY event algorithm for objects transitioning ~~between~~ *from* LIFE_SAFETY_ALARM ~~and~~*to* NORMAL event states. ~~It applies to Event Enrollment objects with an Event_Type of CHANGE_OF_LIFE_SAFETY and to intrinsic event reporting for Life Safety Point and Life Safety Zone objects.~~

The test concept, configuration requirements and test steps are identical to those in 8.4.8.4, except that the ~~Issue_Confirmed_Notifications property~~*Issue Confirmed Notifications parameter* shall have a value of FALSE, UnconfirmedEventNotification requests are issued instead of ConfirmedEventNotification requests and the TD does not acknowledge receiving the notifications.

8.5.8.5 LIFE_SAFETY_ALARM to OFFNORMAL Transition Test

~~Dependencies: ReadProperty Service Execution Tests, 9.18; WriteProperty Service Execution Tests, 9.22.~~

~~BACnet Reference Clauses: 12.12.7, 12.15, 12.16, 13.3.8, 13.9 and Figure 13-9.~~

Purpose: This test case verifies the correct operation of the CHANGE_OF_LIFE_SAFETY event algorithm for objects transitioning ~~between~~ *from* LIFE_SAFETY_ALARM ~~and~~*to* OFFNORMAL event states. ~~It applies to Event Enrollment objects with an Event_Type of CHANGE_OF_LIFE_SAFETY and to intrinsic event reporting for Life Safety Point and Life Safety Zone objects.~~

The test concept, configuration requirements and test steps are identical to those in 8.4.8.5, except that the ~~Issue_Confirmed_Notifications property~~*Issue Confirmed Notifications parameter* shall have a value of FALSE, UnconfirmedEventNotification requests are issued instead of ConfirmedEventNotification requests and the TD does not acknowledge receiving the notifications.

8.5.8.6 OFFNORMAL to LIFE_SAFETY_ALARM Transition Test

~~Dependencies: ReadProperty Service Execution Tests, 9.18; WriteProperty Service Execution Tests, 9.22.~~

~~BACnet Reference Clauses: 12.12.7, 12.15, 12.16, 13.3.8, 13.9 and Figure 13-9.~~

Purpose: This test case verifies the correct operation of the CHANGE_OF_LIFE_SAFETY event algorithm for objects transitioning ~~between~~ *from* OFFNORMAL ~~and~~*to* LIFE_SAFETY_ALARM event states. ~~It applies to Event Enrollment objects with an Event_Type of CHANGE_OF_LIFE_SAFETY and to intrinsic event reporting for Life Safety Point and Life Safety Zone objects.~~

The test concept, configuration requirements and test steps are identical to those in 8.4.8.6, except that the ~~Issue_Confirmed_Notifications property~~*Issue Confirmed Notifications parameter* shall have a value of FALSE, UnconfirmedEventNotification requests are issued instead of ConfirmedEventNotification requests and the TD does not acknowledge receiving the notifications.

8.5.8.7 Mode Transition Tests when Event State is Maintained

~~Dependencies: ReadProperty Service Execution Tests, 9.18; WriteProperty Service Execution Tests, 9.22.~~

~~BACnet Reference Clauses: 12.12.7, 12.15, 12.16, 13.3.8, 13.9 and Figure 13-9.~~

Purpose: To verify the correct operation of the CHANGE_OF_LIFE_SAFETY event algorithm for objects transitioning between the NORMAL, OFFNORMAL and LIFE_SAFETY_ALARM event states when a mode change occurs. Tests are conducted when a mode change occurs, but the event state does not change. Tests are also conducted when a mode change occurs simultaneously with an event state change. In this latter case, the test verifies that the notification is immediate rather than waiting for the time delay. ~~It applies to Event Enrollment objects with an Event_Type of CHANGE_OF_LIFE_SAFETY and to intrinsic event reporting for Life Safety Point and Life Safety Zone objects.~~

The test concept, configuration requirements and test steps are identical to those in 8.4.8.7, except that the ~~Issue_Confirmed_Notifications property~~*Issue Confirmed Notifications parameter* shall have a value of FALSE, UnconfirmedEventNotification requests are issued instead of ConfirmedEventNotification requests and the TD does not acknowledge receiving the notifications.

8.5.8.8 NORMAL to OFFNORMAL Mode Transition Test

~~Dependencies: ReadProperty Service Execution Tests, 9.18; WriteProperty Service Execution Tests, 9.22.~~

~~BACnet Reference Clauses: 12.12.7, 12.15, 12.16, 13.3.8, 13.9 and Figure 13-9.~~

Purpose: This test case verifies the correct operation of the CHANGE_OF_LIFE_SAFETY event algorithm for objects transitioning ~~between~~ *from* NORMAL ~~and~~*to* OFFNORMAL event states by changing the Mode. ~~It applies to Event Enrollment objects with an Event_Type of CHANGE_OF_LIFE_SAFETY and to intrinsic event reporting for Life Safety Point and Life Safety Zone objects.~~

The test concept, configuration requirements and test steps are identical to those in 8.4.8.8, except that the ~~Issue_Confirmed_Notifications property~~*Issue Confirmed Notifications parameter* shall have a value of FALSE, UnconfirmedEventNotification requests are issued instead of ConfirmedEventNotification requests and the TD does not acknowledge receiving the notifications.

8.5.8.9 OFFNORMAL to NORMAL Mode Transition Test

~~Dependencies: ReadProperty Service Execution Tests, 9.18; WriteProperty Service Execution Tests, 9.22.~~

~~BACnet Reference Clauses: 12.12.7, 12.15, 12.16, 13.3.8, 13.9 and Figure 13-9.~~

Purpose: This test case verifies the correct operation of the CHANGE_OF_LIFE_SAFETY event algorithm for objects transitioning ~~between~~ *from* OFFNORMAL ~~and~~*to* NORMAL event states by changing the Mode. ~~It applies to Event Enrollment objects with an Event_Type of CHANGE_OF_LIFE_SAFETY and to intrinsic event reporting for Life Safety Point and Life Safety Zone objects.~~

The test concept, configuration requirements and test steps are identical to those in 8.4.8.9, except that the ~~Issue_Confirmed_Notifications property~~*Issue Confirmed Notifications parameter* shall have a value of FALSE, UnconfirmedEventNotification requests are issued instead of ConfirmedEventNotification requests and the TD does not acknowledge receiving the notifications.

8.5.8.10 NORMAL to LIFE_SAFETY_ALARM Mode Transition Test

~~Dependencies: ReadProperty Service Execution Tests, 9.18; WriteProperty Service Execution Tests, 9.22.~~

~~BACnet Reference Clauses: 12.12.7, 12.15, 12.16, 13.3.8, 13.9 and Figure 13-9.~~

Purpose: This test case verifies the correct operation of the CHANGE_OF_LIFE_SAFETY event algorithm for objects transitioning ~~between~~ *from* NORMAL ~~and~~*to* LIFE_SAFETY_ALARM event states by changing the Mode. ~~It applies to Event Enrollment objects with an Event_Type of CHANGE_OF_LIFE_SAFETY and to intrinsic event reporting for Life Safety Point and Life Safety Zone objects.~~

The test concept, configuration requirements and test steps are identical to those in 8.4.8.10, except that the ~~Issue_Confirmed_Notifications property~~*Issue Confirmed Notifications parameter* shall have a value of FALSE, UnconfirmedEventNotification requests are issued instead of ConfirmedEventNotification requests and the TD does not acknowledge receiving the notifications.

8.5.8.11 LIFE_SAFETY_ALARM to NORMAL Mode Transition Test

~~Dependencies: ReadProperty Service Execution Tests, 9.18; WriteProperty Service Execution Tests, 9.22.~~

~~BACnet Reference Clauses: 12.12.7, 12.15, 12.16, 13.3.8, 13.9 and Figure 13-9.~~

Purpose: This test case verifies the correct operation of the CHANGE_OF_LIFE_SAFETY event algorithm for objects transitioning ~~between~~ *from* LIFE_SAFETY_ALARM ~~and~~*to* NORMAL event states by changing the Mode property. ~~It applies to Event Enrollment objects with an Event_Type of CHANGE_OF_LIFE_SAFETY and to intrinsic event reporting for Life Safety Point and Life Safety Zone objects.~~

The test concept, configuration requirements and test steps are identical to those in 8.4.8.11, except that the ~~Issue_Confirmed_Notifications property~~*Issue Confirmed Notifications parameter* shall have a value of FALSE, UnconfirmedEventNotification requests are issued instead of ConfirmedEventNotification requests and the TD does not acknowledge receiving the notifications.

8.5.8.12 LIFE_SAFETY_ALARM to OFFNORMAL Mode Transition Test

~~Dependencies: ReadProperty Service Execution Tests, 9.18; WriteProperty Service Execution Tests, 9.22.~~

~~BACnet Reference Clauses: 12.12.7, 12.15, 12.16, 13.3.8, 13.9 and Figure 13-9.~~

Purpose: This test case verifies the correct operation of the CHANGE_OF_LIFE_SAFETY event algorithm for objects transitioning ~~between~~ *from* LIFE_SAFETY_ALARM ~~and~~*to* OFFNORMAL event states. ~~It applies to Event Enrollment objects with an Event_Type of CHANGE_OF_LIFE_SAFETY and to intrinsic event reporting for Life Safety Point and Life Safety Zone objects.~~

The test concept, configuration requirements and test steps are identical to those in 8.4.8.12, except that the ~~Issue_Confirmed_Notifications property~~*Issue Confirmed Notifications parameter* shall have a value of FALSE, UnconfirmedEventNotification requests are issued instead of ConfirmedEventNotification requests and the TD does not acknowledge receiving the notifications.

8.5.8.13 OFFNORMAL to LIFE_SAFETY_ALARM Mode Transition Test

~~Dependencies: ReadProperty Service Execution Tests, 9.18; WriteProperty Service Execution Tests, 9.22.~~

~~BACnet Reference Clauses: 12.12.7, 12.15, 12.16, 13.3.8, 13.9 and Figure 13-9.~~

Purpose: This test case verifies the correct operation of the CHANGE_OF_LIFE_SAFETY event algorithm for objects transitioning from the OFFNORMAL and LIFE_SAFETY_ALARM event states by changing the Mode property. ~~It applies to Event Enrollment objects with an Event_Type of CHANGE_OF_LIFE_SAFETY and to intrinsic event reporting for Life Safety Point and Life Safety Zone objects.~~

The test concept, configuration requirements and test steps are identical to those in 8.4.8.13, except that the ~~Issue_Confirmed_Notifications property~~*Issue Confirmed Notifications parameter* shall have a value of FALSE, UnconfirmedEventNotification requests are issued instead of ConfirmedEventNotification requests and the TD does not acknowledge receiving the notifications.

[Remove **Clause 8.5.8.14**, p. 205]
[Reviewer Note: Addendum *af* to ANSI/ASHRAE 135-2010 introduced a new FAULT_LIFE_SAFETY fault algorithm which is covered by another test proposed in this addendum.]

~~**8.5.8.14 TO-FAULT Transition Tests**~~

~~Dependencies: ReadProperty Service Execution Tests, 9.18; WriteProperty Service Execution Tests, 9.22.~~

~~BACnet Reference Clauses: 12.12.7, 12.15, 12.16, 13.3.8, 13.9 and Figure 13-9.~~

~~Purpose: This test case verifies the correct operation of the CHANGE_OF_LIFE_SAFETY event algorithm for objects transitioning between the NORMAL and FAULT event states. It applies to intrinsic event reporting for Life Safety Point and Life Safety Zone objects.~~

~~The test concept, configuration requirements and test steps are identical to those in 8.4.8.14, except that the Issue_Confirmed_Notifications property shall have a value of FALSE, UnconfirmedEventNotification requests are issued instead of ConfirmedEventNotification requests and the TD does not acknowledge receiving the notifications.~~

[Add new **Clause 8.5.9**, p. 206]

8.5.9 EXTENDED Test (UnconfirmedEventNotification)

Purpose: To verify the correct generation of EXTENDED event notifications.

Test Concept: The event generating object begins the test in a NORMAL state. The event generating object is made to transition to any state by any means necessary. The resulting UnconfirmedEventNotification message is received and verified.

Configuration Requirements: The configuration requirements are identical to those in 8.4.9, except that the 'Issue Confirmed Notifications' parameter shall have a value of FALSE.

Test Steps: The test steps for this test case are identical to the test steps in 8.4.9, except that the ConfirmedEventNotification requests are UnconfirmedEventNotification requests and the TD does not acknowledge receiving the notifications.

Notes to Tester: The passing results for this test case are identical to the ones in 8.4.9, except that the event notifications shall be conveyed using an UnconfirmedEventNotification service request. The MAC address used for these messages shall be either a broadcast that reaches the local network of the TD or the MAC address of the TD.

[Add new **Clause 8.5.10**, p. 206]
[Reviewer Note: New event algorithm for Protocol_Revision 10.]

8.5.10 DOUBLE_OUT_OF_RANGE Test (UnconfirmedEventNotification)

Purpose: To verify the correct operation of the DOUBLE_OUT_OF_RANGE event algorithm.

Test Concept: This test is the same as 8.5.6, except that the Event_Type is DOUBLE_OUT_OF_RANGE instead of OUT_OF_RANGE.

[Add new **Clause 8.5.11**, p. 206]
[Reviewer Note: New event algorithm for Protocol_Revision 10.]

8.5.11 SIGNED_OUT_OF_RANGE Test (UnconfirmedEventNotification)

Purpose: To verify the correct operation of the SIGNED_OUT_OF_RANGE event algorithm.

Test Concept: This test is the same as 8.5.6, except that the Event_Type is SIGNED_OUT_OF_RANGE instead of OUT_OF_RANGE.

[Add new **Clause 8.5.12**, p. 206]
[Reviewer Note: New event algorithm for Protocol_Revision 10.]

8.5.12 UNSIGNED_OUT_OF_RANGE Test (UnconfirmedEventNotification)

Purpose: To verify the correct operation of the UNSIGNED_OUT_OF_RANGE event algorithm.

Test Concept: This test is the same as 8.5.6, except that the Event_Type is UNSIGNED_OUT_OF_RANGE instead of OUT_OF_RANGE.

[Add new **Clause 8.5.13**, p. 206]
[Reviewer Note: New event algorithm for Protocol_Revision 10.]

8.5.13 CHANGE_OF_CHARACTERSTRING Test (UnconfirmedEventNotification)

Purpose: To verify the correct operation of the CHANGE_OF_CHARACTERSTRING event algorithm.

Test Concept: The test concept corresponds to 8.4.13.

Configuration Requirements: The configuration requirements are identical to those in 8.4.13, except that the 'Issue Confirmed Notifications' parameter shall have a value of FALSE.

Test Steps: The test steps for this test case are identical to the test steps in 8.4.13, except that the event notification requests are UnconfirmedEventNotification requests and the TD does not acknowledge receiving the notifications.

Notes to Tester: The passing results for this test case are identical to the ones in 8.4.13, except that the event notifications shall be conveyed using an UnconfirmedEventNotification service request. The MAC address used for these messages shall be either a broadcast that reaches the local network of the TD or the MAC address of the TD.

[Add new **Clause 8.5.14**, p. 206]

8.5.14 UNSIGNED_RANGE Test (UnconfirmedEventNotification)

Purpose: To verify the correct operation of the UNSIGNED_RANGE event algorithm.

Test Concept: The test concept corresponds to 8.4.14.

Configuration Requirements: The configuration requirements are identical to those in 8.4.14, except that the 'Issue Confirmed Notifications' parameter shall have a value of FALSE.

Test Steps: The test steps for this test case are identical to the test steps in 8.4.14 except that the ConfirmedEventNotification requests are UnconfirmedEventNotification requests and the TD does not acknowledge receiving the notifications.

Notes to Tester: The passing results for this test case are identical to the ones in 8.4.14 except that the event notifications shall be conveyed using an UnconfirmedEventNotification service request. The MAC address used for these messages shall be either a broadcast that reaches the local network of the TD or the MAC address of the TD.

[Add new **Clause 8.5.15**, p. 206]
[Reviewer Note: New event algorithm for Protocol_Revision 11.]

8.5.15 CHANGE_OF_STATUS_FLAGS Test (UnconfirmedEventNotification)

Purpose: To verify the correct operation of the CHANGE_OF_STATUS_FLAGS event algorithm.

Test Concept: The test concept corresponds to 8.4.15.

Configuration Requirements: The configuration requirements are identical to those in 8.4.15, except that the 'Issue Confirmed Notifications' parameter shall have a value of FALSE.

Test Steps: The test steps for this test case are identical to the test steps in 8.4.15 except that the ConfirmedEventNotification requests are UnconfirmedEventNotification requests and the TD does not acknowledge receiving the notifications.

Notes to Tester: The passing results for this test case are identical to the ones in 8.4.15 except that the event notifications shall be conveyed using an UnconfirmedEventNotification service request. The MAC address used for these messages shall be either a broadcast that reaches the local network of the TD or the MAC address of the TD.

[Add new **Clause 8.5.16**, p. 206]

8.5.16 Proprietary Algorithm Tests (UnconfirmedEventNotifications)

Purpose: This test verifies the correct generation of UnconfirmedEventNotification messages conveying a notification of the complex form. This applies to any non-Event Enrollment object that uses a proprietary algorithm.

Test Concept: The event generating object begins the test in a NORMAL state. The event generating object is made to transition to any state by any means necessary. The resulting UnconfirmedEventNotification message is received and verified.

Configuration Requirements: The configuration requirements are identical to those in 8.4.16, except that the 'Issue Confirmed Notifications' parameter shall have a value of FALSE.

Test Steps: The test steps for this test case are identical to the test steps in 8.4.16, except that the ConfirmedEventNotification requests are UnconfirmedEventNotification requests and the TD does not acknowledge receiving the notifications.

Notes to Tester: The passing results for this test case are identical to the ones in 8.4.16, except that the event notifications shall be conveyed using an UnconfirmedEventNotification service request. The MAC address used for these messages shall be either a broadcast that reaches the local network of the TD or the MAC address of the TD.

[Add new **Clause 8.5.17**, p. 206]

8.5.17 CHANGE_OF_RELIABILITY Tests

The CHANGE_OF_RELIABILITY tests apply to devices that claim conformance to Protocol_Revision 13 or greater only.

8.5.17.1 CHANGE_OF_RELIABILITY with the NONE fault Algorithm (UnconfirmedEventNotifications)

Purpose: To verify the correct operation of the NONE fault algorithm.

Test Concept: Select an object, O1, capable of generating fault using the NONE fault algorithm. Ensure that no other fault conditions exist for the object. Create a fault condition. Verify the transition to fault is generated with Reliability set to R1. Remove the fault condition and verify the object transitions out of fault.

Configuration Requirements: O1 is configured to detect and report faults, to have no fault conditions present and the Event_State is NORMAL. The 'Issue Confirmed Notifications' parameter shall have a value of FALSE.

Test Steps:

1. VERIFY pCurrentReliability = NO_FAULT_DETECTED
2. VERIFY pCurrentState = NORMAL
3. MAKE (O1 enter a fault condition)
4. BEFORE **Notification Fail Time**
   RECEIVE UnconfirmedEventNotification-Request
   'Process Identifier' = (any valid process identifier),

'Initiating Device Identifier' = IUT,
'Event Object Identifier' = O1,
'Time Stamp' = (any valid time stamp),
'Notification Class' = (the notification class configured for O1),
'Priority' = (the value configured for the transition),
'Event Type' = CHANGE_OF_RELIABILITY,
'Message Text' = (optional, any valid message text),
'Notify Type' = ALARM | EVENT,
'AckRequired' = TRUE | FALSE,
'From State' = NORMAL,
'To State' = FAULT,
'Event Values' = (R1 any valid BACnetReliability,
(?, T, ?, ?),
(A list of valid values for properties required to be reported
for O1, and 0 or more other properties of O1)
)

5. VERIFY pCurrentReliability = R1
6. VERIFY pCurrentState = FAULT
7. MAKE (O1clear the fault condition)
8. BEFORE **Notification Fail Time**
   RECEIVE UnconfirmedEventNotification-Request
   'Process Identifier' = (any valid process identifier),
   'Initiating Device Identifier' = IUT,
   'Event Object Identifier' = O1,
   'Time Stamp' = (any valid time stamp),
   'Notification Class' = (the notification class configured for O1),
   'Priority' = (the value configured for the transition),
   'Event Type' = CHANGE_OF_RELIABILITY,
   'Message Text' = (optional, any valid message text),
   'Notify Type' = ALARM | EVENT,
   'AckRequired' = TRUE | FALSE,
   'From State' = FAULT,
   'To State' = NORMAL,
   'Event Values' = (NO_FAULT_DETECTED,
   (?, F, ?, ?),
   (A list of valid values for properties required to be reported
   for O1, and 0 or more other properties of O1)
)
9. VERIFY pCurrentReliability = NO_FAULT_DETECTED
10. VERIFY pCurrentState = NORMAL

Notes to Tester: The mechanism to enter the NONE fault algorithm is a local matter.

8.5.17.2 CHANGE_OF_RELIABILITY with the FAULT_CHARACTERSTRING Algorithm (UnconfirmedEventNotifications)

Purpose: To verify the correct operation of the FAULT_CHARACTERSTRING fault algorithm.

Test Concept: Select a fault detecting object, O1, which is configured to use the FAULT_CHARACTERSTRING algorithm, and no other fault conditions exist for the object. pMonitoredValue is changed to a fault string and back to a non-fault string. It is verified that O1 generates the correct transitions.

Configuration Requirements: O1 is configured to detect and report faults, to have no fault conditions present, and to be in the NORMAL state. FVSET is the set of character strings defined as fault values for O1. ONVSET is the set of character strings defined as offnormal values for O1. FV1 contain a substring that exists in FVSET. If the empty string is included in the

FVSET, then FV1 should be the empty string. NFV1 is a string value that does not contain substrings from FVSET or ONVSET. The 'Issue Confirmed Notifications' parameter shall have a value of FALSE.

Test Steps:

1. VERIFY pCurrentReliability = NO_FAULT_DETECTED
2. VERIFY pCurrentState = NORMAL
3. IF (pMonitoredValue is writable) THEN
   WRITE pMonitoredValue = FV1
   ELSE
   MAKE (pMonitoredValue = FV1)
4. BEFORE **Notification Fail Time**
   RECEIVE UnconfirmedEventNotification-Request

'Process Identifier' =	(any valid process identifier),
'Initiating Device Identifier' =	IUT,
'Event Object Identifier' =	O1,
'Time Stamp' =	(any valid time stamp),
'Notification Class' =	(the notification class configured for O1),
'Priority' =	(the value configured for the transition),
'Event Type' =	CHANGE_OF_RELIABILITY,
'Message Text' =	(optional, any valid message text),
'Notify Type' =	ALARM | EVENT,
'AckRequired' =	TRUE | FALSE,
'From State' =	NORMAL,
'To State' =	FAULT,
'Event Values' =	(MULTI_STATE_FAULT, (T, T, ?, ?), (A list of valid values for properties required to be reported for O1, and 0 or more other properties of O1))

5. VERIFY pCurrentReliability = MULTI_STATE_FAULT
6. IF (pMonitoredValue is writable) THEN
   WRITE pMonitoredValue = NFV1
   ELSE
   MAKE (pMonitoredValue = NFV1)
7. BEFORE **Notification Fail Time**
   RECEIVE UnconfirmedEventNotification-Request

'Process Identifier' =	(any valid process identifier),
'Initiating Device Identifier' =	IUT,
'Event Object Identifier' =	O1,
'Time Stamp' =	(any valid time stamp),
'Notification Class' =	(the notification class configured for O1),
'Priority' =	(the value configured for the transition),
'Event Type' =	CHANGE_OF_RELIABILITY,
'Message Text' =	(optional, any valid message text),
'Notify Type' =	ALARM | EVENT,
'AckRequired' =	TRUE | FALSE,
'From State' =	FAULT,
'To State' =	NORMAL,
'Event Values' =	(NO_FAULT_DETECTED, (F, F, ?, ?), (A list of valid values for properties required to be reported for O1, and 0 or more other properties of O1))

8. VERIFY pCurrentReliability = NO_FAULT_DETECTED

Notes to Tester: Note that a string is considered a substring of itself. Values required and allowed for O1 are described in Standard 135 as "Properties Reported in CHANGE_OF_RELIABILITY Notifications" (Table 13-5 in 135-2016) along with supporting paragraphs.

8.5.17.3 CHANGE_OF_RELIABILITY with the FAULT_EXTENDED Algorithm (UnconfirmedEventNotifications)

Purpose: To verify the correct operation of the FAULT_EXTENDED fault algorithm.

Test Concept: Select a fault detecting object, O1, which is configured to use the FAULT_EXTENDED algorithm, and either pMonitoredValue is configured. Ensure that no other fault conditions exist for the object. In object O1, a condition is created that is detected as a fault by the FAULT_EXTENDED algorithm configured. The fault condition is then removed. It is verified that O1 generates the correct notifications.

Configuration Requirements: O1 is configured to detect and report faults, to have no fault conditions present and has an Event_State of NORMAL. The 'Issue Confirmed Notifications' parameter shall have a value of FALSE.

Test Steps:

1. VERIFY pCurrentReliability = NO_FAULT_DETECTED
2. VERIFY pCurrentState = NORMAL
3. MAKE (a fault condition exist)
4. BEFORE **Notification Fail Time**
   RECEIVE UnconfirmedEventNotification-Request
   'Process Identifier' = (any valid process identifier),
   'Initiating Device Identifier' = IUT,
   'Event Object Identifier' = O1,
   'Time Stamp' = (any valid time stamp),
   'Notification Class' = (the notification class configured for O1),
   'Priority' = (the value configured for the transition),
   'Event Type' = CHANGE_OF_RELIABILITY,
   'Message Text' = (optional, any valid message text),
   'Notify Type' = ALARM | EVENT,
   'AckRequired' = TRUE | FALSE,
   'From State' = (any valid event state),
   'To State' = FAULT,
   'Event Values' = ((R1: any valid reliability value),
   (T, T, ?, ?),
   (a vendor specified set of values)
)
5. VERIFY pCurrentReliability = R1
6. MAKE (remove the fault condition)
7. BEFORE **Notification Fail Time**
   RECEIVE UnconfirmedEventNotification-Request
   'Process Identifier' = (any valid process identifier),
   'Initiating Device Identifier' = IUT,
   'Event Object Identifier' = O1,
   'Time Stamp' = (any valid time stamp),
   'Notification Class' = (the notification class configured for O1),
   'Priority' = (the value configured for the transition),
   'Event Type' = CHANGE_OF_RELIABILITY,
   'Message Text' = (optional, any valid message text),
   'Notify Type' = ALARM | EVENT,
   'AckRequired' = TRUE | FALSE,
   'From State' = FAULT,
   'To State' = NORMAL,
   'Event Values' = (NO_FAULT_DETECTED,
   (?, F, ?, ?),
   (a vendor specified set of values)

)

8. VERIFY pCurrentReliability = NO_FAULT_DETECTED

8.5.17.4 CHANGE_OF_RELIABILITY with the FAULT_LIFE_SAFETY Algorithm (UnconfirmedEventNotifications)

Purpose: To verify the correct operation of the FAULT_LIFE_SAFETY fault algorithm.

Test Concept: Select a fault detecting object, O1, which is configured to use the FAULT_LIFE_SAFETY algorithm. Ensure that no other fault conditions exist in the object. Set pMonitoredValue to FV1, a value which indicates a FAULT_LIFE_SAFETY fault condition. Verify the correct transition is generated. The fault condition is removed by setting pMonitoredValue to NV1, a value, which indicates NO_FAULT_DETECTED and verify the correct transition is generated.

Configuration Requirements: O1 is configured to detect and report faults. O1 is initially configured to have no fault conditions present and has an Event_State of NORMAL. FV1 is a value for pMonitoredValue, which indicates a fault condition, and NV1 is a value for pMonitoredValue, which does not indicate a fault condition. . The 'Issue Confirmed Notifications' parameter shall have a value of FALSE.

Test Steps:

1. VERIFY pCurrentReliability = NO_FAULT_DETECTED
2. VERIFY pCurrentState = NORMAL
3. IF (pMonitoredValue is writable) THEN
   WRITE pMonitoredValue = FV1
   ELSE
   MAKE (pMonitoredValue = FV1)
4. BEFORE **Notification Fail Time**
   RECEIVE UnconfirmedEventNotification-Request

'Process Identifier' =	(any valid process identifier),
'Initiating Device Identifier' =	IUT,
'Event Object Identifier' =	O1,
'Time Stamp' =	(any valid time stamp),
'Notification Class' =	(the notification class configured for O1),
'Priority' =	(the value configured for the transition),
'Event Type' =	CHANGE_OF_RELIABILITY,
'Message Text' =	(optional, any valid message text),
'Notify Type' =	ALARM | EVENT,
'AckRequired' =	TRUE | FALSE,
'From State' =	NORMAL,
'To State' =	FAULT,
'Event Values' =	(MULTI_STATE_FAULT, (T, T, ?, ?), (A list of valid values for properties required to be reported for O1, and 0 or more other properties of O1))

5. VERIFY pCurrentReliability = MULTI_STATE_FAULT
6. IF (pMonitoredValue is writable) THEN
   WRITE pMonitoredValue = NV1
   ELSE
   MAKE (pMonitoredValue = NV1)
7. BEFORE **Notification Fail Time**
   RECEIVE UnconfirmedEventNotification-Request

'Process Identifier' =	(any valid process identifier),
'Initiating Device Identifier' =	IUT,
'Event Object Identifier' =	O1,
'Time Stamp' =	(any valid time stamp),
'Notification Class' =	(the notification class configured for O1),
'Priority' =	(the value configured for the transition),

'Event Type' = CHANGE_OF_RELIABILITY,
'Message Text' = (optional, any valid message text),
'Notify Type' = ALARM | EVENT,
'AckRequired' = TRUE | FALSE,
'From State' = FAULT,
'To State' = NORMAL,
'Event Values' = (NO_FAULT_DETECTED,
(F, F, ?, ?),
(A list of valid values for properties required to be reported
for O1, and 0 or more other properties of O1)
)

8. VERIFY pCurrentReliability = NO_FAULT_DETECTED

8.5.17.5 CHANGE_OF_RELIABILITY with the FAULT_STATE Algorithm (UnconfirmedEventNotifications)

Purpose: To verify the correct operation of the FAULT_STATE fault algorithm.

Test Concept: Select a fault detecting object, O1, which is configured to use the FAULT_STATE algorithm. Ensure that no other fault conditions exist in the object. Set pMonitoredValue to FV1, a value, which indicates a FAULT_STATE fault condition. Verify the correct transition is generated. The fault condition is removed by setting pMonitoredValue to NV1, a value which indicates NO_FAULT_DETECTED and verify the correct transition is generated.

Configuration Requirements: O1 is configured to detect and report faults. O1 is initially configured to have no fault conditions present, and an Event_State of NORMAL. FV1 is a value for pMonitoredValue, which indicates a fault condition, and NV1 is a value for pMonitoredValue which does not indicate a fault condition. The 'Issue Confirmed Notifications' parameter shall have a value of FALSE.

Test Steps:

1. VERIFY pCurrentReliability = NO_FAULT_DETECTED
2. VERIFY pCurrentState = NORMAL
3. IF (pMonitoredValue is writable) THEN
   WRITE pMonitoredValue = FV1
   ELSE
   MAKE (pMonitoredValue = FV1)
4. BEFORE **Notification Fail Time**
   RECEIVE UnconfirmedEventNotification-Request
   'Process Identifier' = (any valid process identifier),
   'Initiating Device Identifier' = IUT,
   'Event Object Identifier' = O1,
   'Time Stamp' = (any valid time stamp),
   'Notification Class' = (the notification class configured for O1),
   'Priority' = (the value configured for the transition),
   'Event Type' = CHANGE_OF_RELIABILITY,
   'Message Text' = (optional, any valid message text),
   'Notify Type' = ALARM | EVENT,
   'AckRequired' = TRUE | FALSE,
   'From State' = NORMAL,
   'To State' = FAULT,
   'Event Values' = (MULTI_STATE_FAULT,
   (T, T, ?, ?),
   (A list of valid values for properties required to be reported
   for O1, and 0 or more other properties of O1)
)
5. VERIFY pCurrentReliability = MULTI_STATE_FAULT
6. IF (pMonitoredValue is writable) THEN
   WRITE pMonitoredValue = NV1
   ELSE

MAKE (pMonitoredValue = NV1)

7. BEFORE **Notification Fail Time**
   RECEIVE UnconfirmedEventNotification-Request
   'Process Identifier' = (any valid process identifier),
   'Initiating Device Identifier' = IUT,
   'Event Object Identifier' = O1,
   'Time Stamp' = (any valid time stamp),
   'Notification Class' = (the notification class configured for O1),
   'Priority' = (the value configured for the transition),
   'Event Type' = CHANGE_OF_RELIABILITY,
   'Message Text' = (optional, any valid message text),
   'Notify Type' = ALARM | EVENT,
   'AckRequired' = TRUE | FALSE,
   'From State' = FAULT,
   'To State' = NORMAL,
   'Event Values' = (NO_FAULT_DETECTED,
   (F, F, ?, ?),
   (A list of valid values for properties required to be reported
   for O1, and 0 or more other properties of O1)
)
8. VERIFY pCurrentReliability = NO_FAULT_DETECTED

8.5.17.6 CHANGE_OF_RELIABILITY with the FAULT_STATUS_FLAGS Algorithm (UnconfirmedEventNotifications)

Purpose: To verify the correct operation of the FAULT_STATUS_FLAGS fault algorithm.

Test Concept: Select a fault detecting object, O1, which is configured to use the FAULT_STATUS_FLAGS algorithm. Ensure that no other fault conditions exist for the object. Set pMonitoredValue to FV1, a value which indicates a FAULT_STATUS_FLAGS fault condition. Verify the correct transition is generated. The fault condition is removed by setting pMonitoredValue to NV1, a value, which indicates NO_FAULT_DETECTED and verify the correct transition is generated.

Configuration Requirements: O1 is configured to detect and report faults. O1 is initially configured to have no fault conditions present, and Event_State is NORMAL. FV1 is a value for pMonitoredValue, which indicates a fault condition, and NV1 is a value for pMonitoredValue which does not indicate a fault condition. The 'Issue Confirmed Notifications' parameter shall have a value of FALSE.

Test Steps:

1. VERIFY pCurrentReliability = NO_FAULT_DETECTED
2. VERIFY pCurrentState = NORMAL
3. IF (pMonitoredValue is writable) THEN
   WRITE pMonitoredValue = FV1
   ELSE
   MAKE (pMonitoredValue = FV1)
4. BEFORE **Notification Fail Time**
   RECEIVE UnconfirmedEventNotification-Request
   'Process Identifier' = (any valid process identifier),
   'Initiating Device Identifier' = IUT,
   'Event Object Identifier' = O1,
   'Time Stamp' = (any valid time stamp),
   'Notification Class' = (the notification class configured for O1),
   'Priority' = (the value configured for the transition),
   'Event Type' = CHANGE_OF_RELIABILITY,
   'Message Text' = (optional, any valid message text),
   'Notify Type' = ALARM | EVENT,
   'AckRequired' = TRUE | FALSE,

'From State' = NORMAL,
'To State' = FAULT,
'Event Values' = (MEMBER_FAULT,
(T, T, ?, ?),
(A list of valid values for properties required to be reported
for O1, and 0 or more other properties of O1)
)

5. VERIFY pCurrentReliability = MEMBER_FAULT
6. IF (pMonitoredValue is writable) THEN
   WRITE pMonitoredValue = NV1
   ELSE
   MAKE (pMonitoredValue = NV1)
7. BEFORE **Notification Fail Time**
   RECEIVE UnconfirmedEventNotification-Request
   'Process Identifier' = (any valid process identifier),
   'Initiating Device Identifier' = IUT,
   'Event Object Identifier' = O1,
   'Time Stamp' = (any valid time stamp),
   'Notification Class' = (the notification class configured for O1),
   'Priority' = (the value configured for the transition),
   'Event Type' = CHANGE_OF_RELIABILITY,
   'Message Text' = (optional, any valid message text),
   'Notify Type' = ALARM | EVENT,
   'AckRequired' = TRUE | FALSE,
   'From State' = FAULT,
   'To State' = NORMAL,
   'Event Values' = (NO_FAULT_DETECTED,
   (F, F, ?, ?),
   (A list of valid values for properties required to be reported
   for O1, and 0 or more other properties of O1)
)
8. VERIFY pCurrentReliability = NO_FAULT_DETECTED

8.5.17.7 CHANGE_OF_RELIABILITY for Event Enrollment Fault Condition Precedence (UnconfirmedEventNotifications)

Purpose: To verify that the Event Enrollment object's fault detection gives precedence to faults in the Event Enrollment object, then faults in the monitored object, and finally faults detected by the configured Fault algorithm.

Test Concept: Select an Event Enrollment object, EE1, which monitors an object, O2, which can transition into FAULT. EE1 should be able to be put into a state where it has an internal fault (internal to the Event Enrollment object and unrelated to the Reliability of the monitored object). Starting with both objects in a NORMAL state, causes a condition, which results in a fault in O2. Verify that EE1 reports the fault. Make a condition exist that results in EE1 entering an internal fault. Verify that EE1 reports the new fault condition. Verify that a fault detectable by the fault algorithm does not generate an event. Clear EE1's the internal fault condition and verify that EE1 reports O2's fault. Clear the condition causing O2's fault and verify that EE1 reports fault algorithm event. Clear the condition causing the fault algorithm and verify the return to NORMAL event occurs.

Configuration Requirements: EE1 is configured to detect and report faults and contains a fault algorithm. EE1 and O2 are each initially configured to have no fault conditions present, and Event_State is NORMAL. The 'Issue Confirmed Notifications' parameter shall have a value of FALSE.

Test Steps:

1. VERIFY pCurrentState = NORMAL
2. VERIFY pCurrentReliability = NO_FAULT_DETECTED
3. MAKE (a condition exist which will cause O2 to detect a fault)

4. BEFORE **Notification Fail Time**
   RECEIVE UnconfirmedEventNotification-Request
   'Process Identifier' = (any valid process identifier),
   'Initiating Device Identifier' = IUT,
   'Event Object Identifier' = EE1,
   'Time Stamp' = (any valid time stamp),
   'Notification Class' = (the notification class configured for O1),
   'Priority' = (the value configured for the transition),
   'Event Type' = CHANGE_OF_RELIABILITY,
   'Message Text' = (optional, any valid message text),
   'Notify Type' = ALARM | EVENT,
   'AckRequired' = FALSE,
   'From State' = NORMAL,
   'To State' = FAULT,
   'Event Values' = (MONITORED_OBJECT_FAULT,
   (T, T, ?, ?),
   O2,
   (optional, the value of the monitored property),
   (optional, Reliability of O2),
   (optional, Status_Flags of O2),
   (0 or more other properties of O2)
)
5. VERIFY pCurrentReliability = MONITORED_OBJECT_FAULT
6. MAKE (a condition exist which will cause EE1 to transition into internal fault)
7. BEFORE **Notification Fail Time**
   RECEIVE UnconfirmedEventNotification-Request
   'Process Identifier' = (any valid process identifier),
   'Initiating Device Identifier' = IUT,
   'Event Object Identifier' = EE1,
   'Time Stamp' = (any valid time stamp),
   'Notification Class' = (the notification class configured for O1),
   'Priority' = (the value configured for the transition),
   'Event Type' = CHANGE_OF_RELIABILITY,
   'Message Text' = (optional, any valid message text),
   'Notify Type' = ALARM | EVENT,
   'AckRequired' = FALSE,
   'From State' = FAULT,
   'To State' = FAULT,
   'Event Values' = ((R1: any value other than
   MONITORED_OBJECT_FAULT
   and NO_FAULT_DETECTED),
   (T, T, ?, ?),
   O2,
   (optional, the value of the monitored property),
   (optional, Reliability of O2),
   (optional, Status_Flags of O2),
   (0 or more other properties of O2)
)
8. VERIFY pCurrentReliability = R1
9. MAKE (a condition that results in a fault detectable by the configured fault algorithm with a reliability value, R2, different from R1)
10. CHECK (that the IUT does not send any notifications)
11. VERIFY pCurrentReliability = R1
12. MAKE (clear the condition that caused EE1 to enter into an internal fault)
11. BEFORE **Notification Fail Time**
    RECEIVE UnconfirmedEventNotification-Request

'Process Identifier' = (any valid process identifier),
'Initiating Device Identifier' = IUT,
'Event Object Identifier' = EE1,
'Time Stamp' = (any valid time stamp),
'Notification Class' = (the notification class configured for O1),
'Priority' = (the value configured for the transition),
'Event Type' = CHANGE_OF_RELIABILITY,
'Message Text' = (optional, any valid message text),
'Notify Type' = ALARM | EVENT,
'AckRequired' = FALSE,
'From State' = FAULT,
'To State' = FAULT,
'Event Values' = (MONITORED_OBJECT_FAULT,
(T, T, ?, ?),
O2,
(optional, the value of the monitored property),
(optional, Reliability of O2),
(optional, Status_Flags of O2)
(0 or more other properties of O2)
)

12. VERIFY pCurrentReliability = MONITORED_OBJECT_FAULT
13. MAKE (clear the condition that caused O2 to enter into fault)
14. BEFORE **Notification Fail Time**
    RECEIVE UnconfirmedEventNotification-Request
    'Process Identifier' = (any valid process identifier),
    'Initiating Device Identifier' = IUT,
    'Event Object Identifier' = EE1,
    'Time Stamp' = (any valid time stamp),
    'Notification Class' = (the notification class configured for O1),
    'Priority' = (the value configured for the transition),
    'Event Type' = CHANGE_OF_RELIABILITY,
    'Message Text' = (optional, any valid message text),
    'Notify Type' = ALARM | EVENT,
    'AckRequired' = FALSE,
    'From State' = FAULT,
    'To State' = FAULT,
    'Event Values' = (R2,
    (T, T, ?, ?),
    O2,
    (optional, the value of the monitored property),
    NO_FAULT_DETECTED,
    (optional, Status_Flags of O2),
    (0 or more other properties of O2)
)
15. VERIFY pCurrentReliability = R2
16. MAKE (clear the condition for the fault algorithm)
17. BEFORE **Notification Fail Time**
    RECEIVE UnconfirmedEventNotification-Request
    'Process Identifier' = (any valid process identifier),
    'Initiating Device Identifier' = IUT,
    'Event Object Identifier' = EE1,
    'Time Stamp' = (any valid time stamp),
    'Notification Class' = (the notification class configured for O1),
    'Priority' = (the value configured for the transition),
    'Event Type' = CHANGE_OF_RELIABILITY,
    'Message Text' = (optional, any valid message text),

'Notify Type' = ALARM | EVENT,
'AckRequired' = FALSE,
'From State' = FAULT,
'To State' = NORMAL,
'Event Values' = (NO_FAULT_DETECTED,
(F, F, ?, ?),
O2,
(the value of the monitored property),
NO_FAULT_DETECTED,
(optional, Status_Flags of O2),
(0 or more other properties of O2)
)

19. VERIFY pCurrentReliability = NO_FAULT_DETECTED
20. VERIFY pCurrentState = NORMAL

Notes to Tester: If O2 is located in the IUT, then the IUT shall know and report the property values of O2 in the CHANGE_OF_RELIABILITY notifications. If O2 is not located in the IUT, then more time must be allowed between making or clearing a fault condition in O2 and the IUT detecting the change in O2's Reliability (the Notification Fail Time allowance does not start until after the IUT has acquired the information from O2).

8.5.17.8 CHANGE_OF_RELIABILITY of Event Enrollment Object, Monitored Object Fault (UnconfirmedEventNotifications)

Purpose: To verify the proper operation of the Event Enrollment object's fault detection when the monitored object enters the fault state.

Test Concept: Select an Event Enrollment object, EE1, that monitors an object, M1, which can transition into FAULT. Starting with both objects in a NORMAL state, causes a condition, which results in a fault in M1. Verify EE1 reports the fault. Clear the condition and verify EE1 reports the return to NORMAL.

Configuration Requirements: EE1 is configured to process faults in M1 and to report those. EE1 and M1 are each initially configured to have no fault conditions present, and Event_State is NORMAL. The 'Issue Confirmed Notifications' parameter shall have a value of FALSE.

Test Steps:

1. VERIFY pCurrentReliability = NO_FAULT_DETECTED
2. VERIFY pCurrentState = NORMAL
3. MAKE (M1 enter any fault state)
4. BEFORE **Notification Fail Time**
   RECEIVE UnconfirmedEventNotification-Request
   'Process Identifier' = (any valid process identifier),
   'Initiating Device Identifier' = IUT,
   'Event Object Identifier' = EE1,
   'Time Stamp' = (any valid time stamp),
   'Notification Class' = (the notification class configured for EE1),
   'Priority' = (the value configured for the transition),
   'Event Type' = CHANGE_OF_RELIABILITY,
   'Message Text' = (optional, any valid message text),
   'Notify Type' = ALARM | EVENT,
   'AckRequired' = TRUE | FALSE,
   'From State' = NORMAL,
   'To State' = FAULT,
   'Event Values' = (MONITORED_OBJECT_FAULT,
   (T, T, ?, ?),
   M1,
   (optional, property value of M1),

(optional, M1 Status_Flags, (?, T, ?, ?)),
(0 or more other properties of M1)
)

5. VERIFY pCurrentReliability = MONITORED_OBJECT_FAULT
6. VERIFY pCurrentState = FAULT
7. MAKE (M1 clear fault state)
8. BEFORE **Notification Fail Time**
   RECEIVE UnconfirmedEventNotification-Request
   'Process Identifier' = (any valid process identifier),
   'Initiating Device Identifier' = IUT,
   'Event Object Identifier' = EE1,
   'Time Stamp' = (any valid time stamp),
   'Notification Class' = (the notification class configured for EE1),
   'Priority' = (the value configured for the transition),
   'Event Type' = CHANGE_OF_RELIABILITY,
   'Message Text' = (optional, any valid message text),
   'Notify Type' = ALARM | EVENT,
   'AckRequired' = TRUE | FALSE,
   'From State' = FAULT,
   'To State' = NORMAL,
   'Event Values' = (NO_FAULT_DETECTED,
   (F, F, ?, ?),
   M1,
   (optional, property value of M1),
   (optional, M1 Status_Flags, (?, F, ?, ?)),
   (0 or more other properties of M1)
)
9. VERIFY pCurrentReliability = NO_FAULT_DETECTED

8.5.17.9 CHANGE_OF_RELIABILITY of Event Enrollment Object Fault (UnconfirmedEventNotifications)

Purpose: To verify the Event Enrollment object generates a fault event when the object enters into fault due to an internal unreliable operation.

Test Concept: Select an Event Enrollment object, EE1, which can be made to enter into fault due to an internal unreliable operation. Starting EE1 in a NORMAL state, causes a condition, which results in a fault. Verify that EE1 reports the fault. Clear the condition and verify that EE1 reports the return to NORMAL.

Configuration Requirements: EE1 is configured to be able to enter a fault state and to report those. EE1 is initially configured to have no fault conditions present, and Event_State is NORMAL. The 'Issue Confirmed Notifications' parameter shall have a value of FALSE.

Test Steps:

1. VERIFY pCurrentReliability = NO_FAULT_DETECTED
2. VERIFY pCurrentState = NORMAL
3. MAKE (EE1 enter any fault state)
4. BEFORE **Notification Fail Time**
   RECEIVE UnconfirmedEventNotification-Request
   'Process Identifier' = (any valid process identifier),
   'Initiating Device Identifier' = IUT,
   'Event Object Identifier' = EE1,
   'Time Stamp' = (any valid time stamp),
   'Notification Class' = (the notification class configured for EE1),
   'Priority' = (the value configured for the transition),
   'Event Type' = CHANGE_OF_RELIABILITY,
   'Message Text' = (optional, any valid message text),

'Notify Type' = ALARM | EVENT,
'AckRequired' = TRUE | FALSE,
'From State' = NORMAL,
'To State' = FAULT,
'Event Values' = ((R1: any value other than
MONITORED_OBJECT_FAULT
and NO_FAULT_DETECTED),
(T, T, ?, ?),
(M1, any valid monitored object),
(optional, property value of M1),
(optional, M1 Status_Flags, (?, F, ?, ?)),
(0 or more other properties of M1)
)

5. VERIFY pCurrentReliability = R1
6. VERIFY pCurrentState = FAULT
7. MAKE (EE1 clear fault state)
8. BEFORE **Notification Fail Time**
   RECEIVE UnconfirmedEventNotification-Request
   'Process Identifier' = (any valid process identifier),
   'Initiating Device Identifier' = IUT,
   'Event Object Identifier' = EE1,
   'Time Stamp' = (any valid time stamp),
   'Notification Class' = (the notification class configured for EE1),
   'Priority' = (the value configured for the transition),
   'Event Type' = CHANGE_OF_RELIABILITY,
   'Message Text' = (optional, any valid message text),
   'Notify Type' = ALARM | EVENT,
   'AckRequired' = TRUE | FALSE,
   'From State' = FAULT,
   'To State' = NORMAL,
   'Event Values' = (NO_FAULT_DETECTED,
   (F, F, ?, ?),
   M1,
   (optional, property value of M1),
   (optional, M1 Status_Flags, (?, F, ?, ?)),
   (0 or more other properties of M1)
)
9. VERIFY pCurrentReliability = NO_FAULT_DETECTED

8.5.17.10 After FAULT-to-NORMAL, Re-Notification of OFFNORMAL (UnconfirmedEventNotifications)

Purpose: To verify that objects go to the NORMAL state after leaving the FAULT state, then transition to OFFNORMAL if the condition still exists.

Test Concept: Select a fault detecting object, O1, which is able to detect OFFNORMAL conditions. Make O1 transition to an OFFNORMAL state and then transition to FAULT. Remove the condition causing the FAULT and verify O1 transitions from FAULT to NORMAL, then verify that the object transitions from NORMAL to the original OFFNORMAL state.

Configuration Requirements: O1 is configured to detect and report faults. O1 is configured to have no fault conditions present, and Event_State is *NORMAL*. The 'Issue Confirmed Notifications' parameter shall have a value of FALSE.

Test Steps:

1. VERIFY pCurrentReliability = NO_FAULT_DETECTED
2. VERIFY pCurrentState = NORMAL
3. MAKE (O1transition to an *offnormal* state)

4. BEFORE **Notification Fail Time**
   RECEIVE UnconfirmedEventNotification-Request
   'Process Identifier' = (any valid process identifier),
   'Initiating Device Identifier' = IUT,
   'Event Object Identifier' = O1,
   'Time Stamp' = (any valid time stamp),
   'Notification Class' = (the notification class configured for O1),
   'Priority' = (the value configured for the transition),
   'Event Type' = (ET1, any valid off normal event type),
   'Message Text' = (optional, any valid message text),
   'Notify Type' = ALARM | EVENT,
   'AckRequired' = TRUE | FALSE,
   'From State' = NORMAL,
   'To State' = OFFNORMAL,
   'Event Values' = (property-values appropriate for O1)
5. VERIFY pCurrentState = OFFNORMAL
5. MAKE (O1 enter a fault state)
6. BEFORE **Notification Fail Time**
   RECEIVE UnconfirmedEventNotification-Request
   'Process Identifier' = (any valid process identifier),
   'Initiating Device Identifier' = IUT,
   'Event Object Identifier' = O1,
   'Time Stamp' = (any valid time stamp),
   'Notification Class' = (the notification class configured for O1),
   'Priority' = (the value configured for the transition),
   'Event Type' = CHANGE_OF_RELIABILITY,
   'Message Text' = (optional, any valid message text),
   'Notify Type' = ALARM | EVENT,
   'AckRequired' = TRUE | FALSE,
   'From State' = OFFNORMAL,
   'To State' = FAULT,
   'Event Values' = ((R1 any valid BACnetReliability),
   (?, T, ?, ?),
   (A list of valid values for properties required to be reported
   for O1, and 0 or more other properties of O1)
)
7. MAKE (O1 clear the fault condition)
8. BEFORE **Notification Fail Time**
   RECEIVE UnconfirmedEventNotification-Request
   'Process Identifier' = (any valid process identifier),
   'Initiating Device Identifier' = IUT,
   'Event Object Identifier' = O1,
   'Time Stamp' = (any valid time stamp),
   'Notification Class' = (the notification class configured for O1),
   'Priority' = (the value configured for the transition),
   'Event Type' = CHANGE_OF_RELIABILITY,
   'Message Text' = (optional, any valid message text),
   'Notify Type' = ALARM | EVENT,
   'AckRequired' = TRUE | FALSE,
   'From State' = FAULT,
   'To State' = NORMAL,
   'Event Values' = (NO_FAULT_DETECTED,
   (F, F, ?, ?),
   (A list of valid values for properties required to be reported
   for O1, and 0 or more other properties of O1)
)

9. VERIFY pCurrentReliability = NO_FAULT_DETECTED
10. BEFORE **Notification Fail Time**
    RECEIVE UnconfirmedEventNotification-Request
    'Process Identifier' = (any valid process identifier),
    'Initiating Device Identifier' = IUT,
    'Event Object Identifier' = O1,
    'Time Stamp' = (any valid time stamp),
    'Notification Class' = (the notification class configured for O1),
    'Priority' = (the value configured for the transition),
    'Event Type' = ET1,
    'Message Text' = (optional, any valid message text),
    'Notify Type' = ALARM | EVENT,
    'AckRequired' = TRUE | FALSE,
    'From State' = NORMAL,
    'To State' = OFFNORMAL,
    'Event Values' = (property-values appropriate for O1)

8.5.17.11 CHANGE_OF_RELIABILITY with Internal Object Fault (UnconfirmedEventNotifications)

Purpose: To verify that fault conditions, unrelated to fault algorithms, are detected and reported.

Test Concept: An object in the IUT, O1, which can detect at least one internal fault is selected. One of O1's detectable internal faults, R1, is selected for the test. O1 begins the test in the NORMAL state with pCurrentReliability equal to NO_FAULT_DETECTED. The internal fault condition, R1, is made to exist, and it is verified that the pCurrentReliability changes to R1. It is verified that O1 generates the appropriate event notification. The fault condition is removed, and it is verified that the pCurrentReliability returns to NO_FAULT_DETECTED and the appropriate event notification message is generated.

Configuration Requirements: O1 is configured to detect and report faults. O1 is initially configured to have no fault conditions present, and Event_State is NORMAL. The 'Issue Confirmed Notifications' parameter shall have a value of FALSE.

Test Steps:

1. VERIFY pCurrentReliability = NO_FAULT_DETECTED
2. VERIFY pCurrentState = NORMAL
3. MAKE (pCurrentReliability = R1)
4. BEFORE **Notification Fail Time**
   RECEIVE UnconfirmedEventNotification-Request,
   'Process Identifier' = (any valid process ID),
   'Initiating Device Identifier' = IUT,
   'Event Object Identifier' = O1,
   'Time Stamp' = (any valid time stamp),
   'Notification Class' = (the notification class configured for O1),
   'Priority' = (the value configured for the transition),
   'Event Type' = CHANGE_OF_RELIABILITY,
   'Message Text' = (optional, any valid message text),
   'Notify Type' = EVENT | ALARM,
   'AckRequired' = TRUE | FALSE,
   'From State' = NORMAL,
   'To State' = FAULT,
   'Event Values' = (R1,
   (?, T, ?, ?),
   (A list of valid values for properties required to be reported
   for O1, and 0 or more other properties of O1)
)
5. VERIFY pCurrentReliability = R1
6. VERIFY pCurrentState = FAULT

7. MAKE (pCurrentReliability = NO_FAULT_DETECTED)
8. BEFORE **Notification Fail Time**
   RECEIVE UnconfirmedEventNotification-Request,

'Process Identifier' =	(any valid process ID),
'Initiating Device Identifier' =	IUT,
'Event Object Identifier' =	O1,
'Time Stamp' =	(any valid time stamp),
'Notification Class' =	(the notification class configured for O1),
'Priority' =	(the value configured for the transition),
'Event Type' =	CHANGE_OF_RELIABILITY,
'Message Text' =	(optional, any valid message text),
'Notify Type' =	EVENT | ALARM,
'AckRequired' =	TRUE | FALSE,
'From State' =	FAULT,
'To State' =	NORMAL,
'Event Values' =	(NO_FAULT_DETECTED, (?, F, ?, ?), (A list of valid values for properties required to be reported for O1, and 0 or more other properties of O1))

9. VERIFY pCurrentReliability = NO_FAULT_DETECTED
10. VERIFY pCurrentState = NORMAL

[Change **Clause 8.6.1**, p. 206]

8.6.1 Basic GetAlarmSummary Service Initiation

Purpose: To verify that the IUT can initiate GetAlarmSummary service requests.

~~Dependencies: None.~~

~~BACnet Reference Clause: 13.10.~~

Test Steps:

1. RECEIVE GetAlarmSummary-Request
2. TRANSMIT BACnet-ComplexACK-PDU,
   'List of Alarm Summaries' = (an empty list)

[Change **Clause 8.6.2**, p. 206]

8.6.2 Updating Alarm Summary Information with GetAlarmSummary

Purpose: This test case verifies that the IUT is capable of updating or presenting alarm summary information using the GetAlarmSummary service.

Configuration *Requirements*: The TD shall be configured to not support execution of GetEventInformation nor GetEnrollmentSummary. The TD shall be configured with a set of event-generating objects in a variety of event states, which the IUT can interrogate during the execution of the test and some of which will be reported to the IUT via GetAlarmSummary.

Test Steps:

1. MAKE (the IUT present or update the alarm summary presented to the user)
2. RECEIVE GetAlarmSummary-Request
3. TRANSMIT GetAlarmSummary-Ack
   '~~-~~List of Alarm Summaries'~~-~~ = (any valid list)
4. CHECK (that the IUT presents or updates the presentation of the alarm summary information and
   that the presentation is consistent with the information received in step 3)

Notes to Tester: The value presented by the IUT may differ from the value transmitted on the wire due to rounding, truncation, formatting, language conversion, etc.

Notes to Tester: If the IUT has not already determined that the TD does not support execution of GetEventInformation and/or GetEnrollmentSummary, the IUT may initiate a GetEventInformation and/or GetEnrollmentSummary. If this occurs, the IUT shall only pass the test if it automatically falls back to using GetAlarmSummary upon receipt of the correct BACnetError-PDU from the TD, indicating that the alternate service is not supported.

[Change **Clause 8.8.3**, p. 209]

8.8.3 Updating Alarm Summary Information with GetEventInformation Without Chaining

Purpose: This test case verifies that the IUT is capable of updating or presenting alarm summary information using the GetEventInformation service.

Configuration *Requirements*: The TD shall be configured to not support execution of GetAlarmSummary nor GetEnrollmentSummary. The TD shall be configured with a set of event-generating objects in a variety of event states, which the IUT can interrogate during the execution of the test and some of which will be reported to the IUT via GetAlarmSummary. The number of objects that will be reported via GetEventInformation shall not exceed the number that can be reported in a single GetEventInformation-Ack response.

Test Steps:

1. MAKE (the IUT present or update the alarm summary presented to the user)
2. RECEIVE GetEventInformation-Request
3. TRANSMIT GetEventInformation-Ack,
   '~~-~~List of Event Summaries'~~-~~ = (any valid list),
   '~~-~~More Events'~~-~~ = FALSE
4. CHECK (that the IUT presents or updates the alarm summary information presented to the user and that the presentation is consistent with the information received in step 3)

Notes to Tester: The value accepted by the IUT may differ from the value transmitted on the wire due to rounding, truncation, formatting, language conversion, etc.

~~Notes to Tester:~~ If the IUT has not already determined that the TD does not support execution of GetAlarmSummary and/or GetEnrollmentSummary, the IUT may initiate a GetAlarmSummary and/or GetEnrollmentSummary. If this occurs, the IUT shall pass the test only if it automatically falls back to using GetEventInformation upon receipt of the correct BACnetError-PDU from the TD, indicating that alternate service is not supported.

[Change **Clause 8.8.4**, p. 209]

8.8.4 Updating Alarm Summary Information with GetEventInformation With Chaining

Purpose: This test case verifies that the IUT is capable of updating or presenting alarm summary information using the GetEventInformation service.

Configuration *Requirements*: The TD shall be configured to not support execution of GetAlarmSummary nor GetEnrollmentSummary. The TD shall be configured with a set of event-generating objects in a variety of event states, which the IUT can interrogate during the execution of the test and some of which will be reported to the IUT via GetAlarmSummary. The number of objects that will be reported via GetEventInformation shall exceed the number that can be reported in a single GetEventInformation-Ack response.

Test Steps:

1. MAKE (the IUT present or update the alarm summary presented to the user)
2. RECEIVE GetEventInformation-Request
3. TRANSMIT GetEventInformation-Ack,
   '~~-~~List of Event Summaries'~~-~~ = (any valid list that represents the state of event generating objects in the TD),
   '~~-~~More Events'~~-~~ = TRUE
4. RECEIVE GetEventInformation-Request,
   '~~-~~Last Received Object Identifier'~~-~~ = (last object identifier sent in step 3)
5. TRANSMIT GetEventInformation-Ack,
   '~~-~~List of Event Summaries'~~-~~ = (any valid list that represents the state of event generating objects in the TD),

'More Events' = FALSE

6. CHECK (that the IUT presents or updates the alarm summary information presented to the user and that the presentation is consistent with the information received in steps 3 and 5)

Notes to Tester: The value accepted by the IUT may differ from the value transmitted on the wire due to rounding, truncation, formatting, language conversion, etc.

~~Notes to Tester:~~ If the IUT has not already determined that the TD does not support execution of GetAlarmSummary and/or GetEnrollmentSummary, the IUT may initiate a GetAlarmSummary and/or GetEnrollmentSummary. If this occurs, the IUT shall pass the test only if it automatically falls back to using GetEventInformation upon receipt of the correct BACnetError-PDU from the TD, indicating that alternate service is not supported.

[Change **Clause 9.1.1.1**, p. 246]
[Reviewer Note: Made changes to allow cases where only one Recipient_List entry is supported.]

9.1.1.1 Successful Alarm Acknowledgment of Confirmed Event Notifications Using the Time Form of the 'Time of Acknowledgment' Parameter

Purpose: To verify the successful acknowledgment of an alarm signaled by a ConfirmedEventNotification, including notification of other workstations and updating of the Acked_Transitions status. The Time form of the 'Time of Acknowledgment' parameter is used.

Test Concept: An alarm is triggered that causes the IUT to notify the TD and ~~at least~~ one other device. The TD acknowledges the alarm and verifies that the acknowledgment is properly noted by the IUT. The IUT notifies all other recipients that the alarm has been acknowledged.

Configuration Requirements: The IUT shall be configured with ~~at least~~ one object that can detect alarm conditions and send confirmed notifications. The Acked_Transitions property shall have the value B'111' indicating that all transitions have been acknowledged. The TD and ~~at least~~ one other BACnet device, *if the IUT supports multiple recipients,* shall be recipients of the alarm notification.

Test Steps:

1. MAKE (a change that triggers the detection of an alarm event in the IUT)
2. WAIT (~~Time_Delay~~*pTimeDelay*)
3. BEFORE **Notification Fail Time**
   RECEIVE ConfirmedEventNotification-Request,
   'Process Identifier' = (the process identifier configured for this event),
   'Initiating Device Identifier' = IUT,
   'Event Object Identifier' = (the object detecting the alarm),
   'Time Stamp' = (~~the current time or sequence number~~*any valid time stamp*),
   'Notification Class' = (the notification class configured for this event),
   'Priority' = (the priority configured for this event),
   'Event Type' = (any valid event type),
   'Message Text' = *(optional, any valid message text),*
   'Notify Type' = (the notify type configured for this event),
   'AckRequired' = TRUE,
   'From State' = NORMAL,
   'To State' = (any appropriate non-normal event state),
   'Event Values' = (the values appropriate to the event type)
4. TRANSMIT BACnet-SimpleACK-PDU
5. RECEIVE
   DESTINATION = (~~at least one~~ *a* device other than the TD),
   SOURCE = IUT,
   ConfirmedEventNotification-Request,
   'Process Identifier' = (the process identifier configured for this event),
   'Initiating Device Identifier' = IUT,
   'Event Object Identifier' = (the object detecting the alarm),
   'Time Stamp' = (the timestamp or sequence number received in step 3),
   'Notification Class' = (the notification class configured for this event),
   'Priority' = (the priority configured for this event),
   'Event Type' = (any valid event type),
   'Message Text' = *(optional, any valid message text),*
   'Notify Type' = (the notify type configured for this event),
   'AckRequired' = TRUE,
   'From State' = NORMAL,
   'To State' = (any appropriate non-normal event state),
   'Event Values' = (the values appropriate to the event type)

6. TRANSMIT BACnet-SimpleACK-PDU
7. VERIFY (the 'Event Object Identifier' from the event notification), Acked_Transitions = (FALSE, TRUE, TRUE)
8. TRANSMIT AcknowledgeAlarm-Request,
   'Acknowledging Process Identifier' = (the value of the 'Process Identifier' parameter in the event notification),
   'Event Object Identifier' = (the 'Event Object Identifier' from the event notification),
   'Event State Acknowledged' = (the state specified in the 'To State' parameter of the notification),
   'Time Stamp' = (the time stamp conveyed in the notification),
   '*Acknowledgement Source*' = *(any valid value),*
   'Time of Acknowledgment' = (the TD's current time using a Time format)
9. RECEIVE BACnet-~~Simple-ACK~~*SimpleACK*-PDU
10. IF (Protocol_Revision is present and Protocol_Revision ≥ 1) THEN
    BEFORE **Notification Fail Time**
    RECEIVE ConfirmedEventNotification-Request,
    'Process Identifier' = (the process identifier configured for this event),
    'Initiating Device Identifier' = IUT,
    'Event Object Identifier' = (the object detecting the alarm),
    'Time Stamp' = (the current time or sequence number),
    'Notification Class' = (the notification class configured for this event),
    'Priority' = (the priority configured for this event),
    'Event Type' = (the event type included in step 3),
    'Message Text' = *(optional, any valid message text),*
    'Notify Type' = ACK_NOTIFICATION,
    'To State' = (the 'To State' used in step 3)

    ELSE
    BEFORE **Notification Fail Time**
    RECEIVE ConfirmedEventNotification-Request,
    'Process Identifier' = (the process identifier configured for this event),
    'Initiating Device Identifier' = IUT,
    'Event Object Identifier' = (the object detecting the alarm),
    'Time Stamp' = (the current time or sequence number),
    'Notification Class' = (the notification class configured for this event),
    'Priority' = (the priority configured for this event),
    'Event Type' = (the event type included in step 3),
    'Message Text' = *(optional, any valid message text),*
    'Notify Type' = ACK_NOTIFICATION,
    ~~'To State' = (the 'To State' used in step 3)~~
11. TRANSMIT BACnet-SimpleACK-PDU
12. IF (Protocol_Revision is present and Protocol_Revision ≥ 1) THEN
    BEFORE **Notification Fail Time**
    RECEIVE
    DESTINATION = (~~at least one~~ *a* device other than the TD),
    SOURCE = IUT,
    ConfirmedEventNotification-Request,
    'Process Identifier' = (the process identifier configured for this event),
    'Initiating Device Identifier' = IUT,
    'Event Object Identifier' = (the object detecting the alarm),
    'Time Stamp' = (the timestamp or sequence number received in step 10),
    'Notification Class' = (the notification class configured for this event),
    'Priority' = (the priority configured for this event),
    'Event Type' = (the event type included in step 3),
    'Message Text' = *(optional, any valid message text),*
    'Notify Type' = ACK_NOTIFICATION,
    'To State' = (the 'To State' used in step 3)

    ELSE
    BEFORE **Notification Fail Time**
    RECEIVE

DESTINATION = (~~at least one~~ *a* device other than the TD),
SOURCE = IUT,
ConfirmedEventNotification-Request,
'Process Identifier' = (the process identifier configured for this event),
'Initiating Device Identifier' = IUT,
'Event Object Identifier' = (the object detecting the alarm),
'Time Stamp' = (the timestamp or sequence number received in step 10),
'Notification Class' = (the notification class configured for this event),
'Priority' = (the priority configured for this event),
'Event Type' = (the event type included in step 3),
'Message Text' = *(optional, any valid message text),*
'Notify Type' = ACK_NOTIFICATION

13. TRANSMIT BACnet-SimpleACK-PDU
14. VERIFY (the 'Event Object Identifier' from the event notification), Acked_Transitions = (TRUE,TRUE,TRUE)

Notes to Tester: The destination address used for the acknowledgment notification in step 12 shall be the same address used in step 5. Inclusion of the 'To State' parameter in acknowledgement notifications was added in protocol version 1, protocol revision 1. Implementations that precede this version will not include this parameter. When multiple event notifications are expected for a specific event, the order that the IUT transmits them in is irrelevant. *If the IUT can only be configured with one recipient in the Recipient_List property of the issuing Notification_class object, skip all steps related to receipt of the second notification.*

[Change **Clause 9.1.1.2**, p. 249]
[Reviewer Note: Made changes to allow cases where only one Recipient_List entry is supported.]

9.1.1.2 Successful Alarm Acknowledgment of Confirmed Event Notifications using the Sequence Number Form of the 'Time of Acknowledgment' Parameter

Purpose: To verify the successful acknowledgment of an alarm signaled by a ConfirmedEventNotification, including notification of other workstations and updating of the Acked_Transitions status. The Sequence Number form of the 'Time of Acknowledgment' parameter is used.

Test Concept: An alarm is triggered that causes the IUT to notify the TD and ~~at least one other device~~ *one other device*. The TD acknowledges the alarm and verifies that the acknowledgment is properly noted by the IUT. The IUT notifies all other recipients that the alarm has been acknowledged.

Configuration Requirements: The IUT shall be configured with at least one object that can detect alarm conditions and send confirmed notifications. The Acked_Transitions property shall have the value B'111' indicating that all transitions have been acknowledged. The TD and ~~at least~~ one other BACnet device, *if the IUT supports multiple recipients,* shall be recipients of the alarm notification.

Test Steps: The test steps defined in 9.1.1.1 shall be followed except that the 'Time of Acknowledgment' parameter of the AcknowledgeAlarm service request shall be a sequence number.

Notes to Tester: A passing result is the same message sequence described as the passing result in 9.1.1.1. *If the IUT can only be configured with one recipient in the Recipient_List property of the issuing Notification_Class object, skip all steps related to receipt of the second notification.*

[Change **Clause 9.1.1.3**, p. 249]
[Reviewer Note: Made changes to allow cases where only one Recipient_List entry is supported.]

9.1.1.3 Successful Alarm Acknowledgment of Confirmed Event Notifications Using the Date Time Form of the 'Time of Acknowledgment' Parameter

Purpose: To verify the successful acknowledgment of an alarm signaled by a ConfirmedEventNotification, including notification of other workstations and updating of the Acked_Transitions status. The Date Time form of the 'Time of Acknowledgment' parameter is used.

Test Concept: An alarm is triggered that causes the IUT to notify the TD and ~~at least~~ one other device. The TD acknowledges the alarm and verifies that the acknowledgment is properly noted by the IUT. The IUT notifies all other recipients that the alarm has been acknowledged.

Configuration Requirements: The IUT shall be configured with at least one object that can detect alarm conditions and send confirmed notifications. The Acked_Transitions property shall have the value B'111' indicating that all transitions have been acknowledged. The TD and ~~at least~~ one other BACnet*, if the IUT supports multiple recipients,* device shall be recipients of the alarm notification.

Test Steps: The test steps defined in 9.1.1.1 shall be followed except that the 'Time of Acknowledgment' parameter of the AcknowledgeAlarm service request shall convey the current time using a BACnetDateTime format.

Notes to Tester: A passing result is the same message sequence described as the passing result in 9.1.1.1. *If the IUT can only be configured with one recipient in the Recipient_List property of the issuing Notification_class object, skip all steps related to receipt of the second notification.*

[Change **Clause 9.1.1.4**, p. 249]
[Reviewer Note: Made changes to allow cases where only one Recipient_List entry is supported.]

9.1.1.4 Successful Alarm Acknowledgment of Unconfirmed Event Notifications Using the Time Form of the 'Time of Acknowledgment' Parameter

Purpose: To verify the successful acknowledgment of an alarm signaled by an UnconfirmedEventNotification, including notification of other workstations and updating of the Acked_Transitions status. The Time form of the 'Time of Acknowledgment' parameter is used.

Test Concept: An alarm is triggered that causes the IUT to notify the TD and ~~at least~~ one other device. The TD acknowledges the alarm and verifies that the acknowledgment is properly noted by the IUT. The IUT notifies all other recipients that the alarm has been acknowledged.

Configuration Requirements: The IUT shall be configured with at least one object that can detect alarm conditions and send unconfirmed notifications. The Acked_Transitions property shall have the value B'111' indicating that all transitions have been acknowledged. The TD and ~~at least~~ one other BACnet device, *if the IUT supports multiple recipients,* shall be recipients of the alarm notification.

Test Steps:

1. MAKE (a change that triggers the detection of an alarm event in the IUT)
2. WAIT (~~Time_Delay~~*pTimeDelay*)
3. BEFORE **Notification Fail Time**
   RECEIVE UnconfirmedEventNotification-Request,

DESTINATION =	*LOCAL BROADCAST* | *GLOBAL BROADCAST* | *TD,*
SOURCE =	*IUT,*
'Process Identifier' =	(the process identifier configured for this event),
'Initiating Device Identifier' =	IUT,
'Event Object Identifier' =	(the object detecting the alarm),
'Time Stamp' =	(~~the current time or sequence number~~*any valid time stamp*),
'Notification Class' =	(the notification class configured for this event),
'Priority' =	(the priority configured for this event type),

'Event Type' = (any valid event type),
'Message Text' = *(optional, any valid message text),*
'Notify Type' = (the notify type configured for this event),
'AckRequired' = TRUE,
'From State' = NORMAL,
'To State' = (any appropriate non-normal event state),
'Event Values' = (the values appropriate to the event type)

4. IF (the notification in step 3 was not a broadcast) THEN
   RECEIVE
   DESTINATION = (~~at least one~~ *a* device other than the TD),
   SOURCE = IUT,
   UnconfirmedEventNotification-Request,
   'Process Identifier' = (the process identifier configured for this event),
   'Initiating Device Identifier' = IUT,
   'Event Object Identifier' = (the object detecting the alarm),
   'Time Stamp' = (the timestamp or sequence number received in step 3),
   'Notification Class' = (the notification class configured for this event),
   'Priority' = (the priority configured for this event type),
   'Event Type' = (any valid event type),
   'Message Text' = *(optional, any valid message text),*
   'Notify Type' = (the notify type configured for this event),
   'AckRequired' = TRUE,
   'From State' = NORMAL,
   'To State' = (any appropriate non-normal event state),
   'Event Values' = (the values appropriate to the event type)
5. VERIFY (the 'Event Object Identifier' from the event notification), Acked_Transitions = (FALSE,TRUE,TRUE)
6. TRANSMIT AcknowledgeAlarm-Request,
   'Acknowledging Process Identifier' = (the value of the 'Process Identifier' parameter in the event notification),
   'Event Object Identifier' = (the 'Event Object Identifier' from the event notification),
   'Event State Acknowledged' = (the state specified in the 'To State' parameter of the notification),
   'Time Stamp' = (~~the time stamp conveyed in the notification~~*any valid time stamp*),
   '*Acknowledgement Source*' = *(any valid value),*
   'Time of Acknowledgment' = (the TD's current time using a Time format)
7. RECEIVE BACnet-~~Simple ACK~~*SimpleACK*-PDU
8. IF (Protocol_Revision is present and Protocol_Revision ≥ 1) THEN
   BEFORE **Notification Fail Time**
   RECEIVE
   DESTINATION = LOCAL BROADCAST | GLOBAL BROADCAST | TD,
   SOURCE = IUT,
   UnconfirmedEventNotification-Request,
   'Process Identifier' = (the process identifier configured for this event),
   'Initiating Device Identifier' = IUT,
   'Event Object Identifier' = (the object detecting the alarm),
   'Time Stamp' = (~~the current time or sequence number~~*any valid time stamp*),
   'Notification Class' = (the notification class configured for this event),
   'Priority' = (the priority configured for this event type),
   'Event Type' = (any valid event type),
   'Message Text' = *(optional, any valid message text),*
   'Notify Type' = ACK_NOTIFICATION,
   'To State' = (the 'To State' used in step 3 or 4)

   ELSE
   BEFORE **Notification Fail Time**
   RECEIVE
   DESTINATION = LOCAL BROADCAST | GLOBAL BROADCAST | TD,
   SOURCE = IUT,
   UnconfirmedEventNotification-Request,

'Process Identifier' = (the process identifier configured for this event),
'Initiating Device Identifier' = IUT,
'Event Object Identifier' = (the object detecting the alarm),
'Time Stamp' = (~~the current time or sequence number~~*any valid time stamp*),
'Notification Class' = (the notification class configured for this event),
'Priority' = (the priority configured for this event type),
'Event Type' = (any valid event type),
'Message Text' = *(optional, any valid message text),*
'Notify Type' = ACK_NOTIFICATION

9. IF (the notification in step 8 was not broadcast) THEN
   IF (Protocol_Revision is present and Protocol_Revision ≥ 1) THEN
      RECEIVE
         DESTINATION = (~~at least one~~ *a* device other than the TD),
         SOURCE = IUT,
         UnconfirmedEventNotification-Request,
            'Process Identifier' = (the process identifier configured for this event),
            'Initiating Device Identifier' = IUT,
            'Event Object Identifier' = (the object detecting the alarm),
            'Time Stamp' = (the timestamp or sequence number from the notification in step 8),
            'Notification Class' = (the notification class configured for this event),
            'Priority' = (the priority configured for this event type),
            'Event Type' = (any valid event type),
            'Message Text' = *(optional, any valid message text),*
            'Notify Type' = ACK_NOTIFICATION,
            'To State' = (the 'To State' used in step 3 or 4)
   ELSE
      RECEIVE
         DESTINATION = (at least one device other than the TD),
         SOURCE = IUT,
         UnconfirmedEventNotification-Request,
            'Process Identifier' = (the process identifier configured for this event),
            'Initiating Device Identifier' = IUT,
            'Event Object Identifier' = (the object detecting the alarm),
            'Time Stamp' = (the timestamp or sequence number from the notification in step 8),
            'Notification Class' = (the notification class configured for this event),
            'Priority' = (the priority configured for this event type),
            'Event Type' = (any valid event type),
            'Message Text' = *(optional, any valid message text),*
            'Notify Type' = ACK_NOTIFICATION,
10. VERIFY (the 'Event Object Identifier' from the event notification), Acked_Transitions = (TRUE, TRUE, TRUE)

Notes to Tester: The destination address used for the acknowledgment notification in step 8 shall be the same address used in step 3. The destination address used for the acknowledgment notification in step 9 shall be the same address used in step 4. Inclusion of the 'To State' parameter in acknowledgement notifications was added in protocol version 1, protocol revision 1. Implementations that precede this version will not include this parameter. When multiple event notifications are expected for a specific event, the order that the IUT transmits them in is irrelevant. *If the IUT can only be configured with one recipient in the Recipient_List property of the issuing Notification_class object, omit steps 4 and 9.*

[Change **Clause 9.1.1.5**, p. 251]
[Reviewer Note: Made changes to allow cases where only one Recipient_List entry is supported.]

9.1.1.5 Successful Alarm Acknowledgment of Unconfirmed Event Notifications Using the Sequence Number Form of the 'Time of Acknowledgment' Parameter

Purpose: To verify the successful acknowledgment of an alarm signaled by an UnconfirmedEventNotification, including notification of other workstations and updating of the Acked_Transitions status. The Sequence Number form of the 'Time of Acknowledgment' parameter is used.

Test Concept: An alarm is triggered that causes the IUT to notify the TD and ~~at least~~ one other device. The TD acknowledges the alarm and verifies that the acknowledgment is properly noted by the IUT. The IUT notifies all other recipients that the alarm has been acknowledged.

Configuration Requirements: The IUT shall be configured with at least one object that can detect alarm conditions and send unconfirmed notifications. The Acked_Transitions property shall have the value B'111' indicating that all transitions have been acknowledged. The TD and ~~at least~~ one other BACnet device*, if the IUT supports multiple recipients,* shall be recipients of the alarm notification.

Test Steps: The test steps defined in 9.1.1.4 shall be followed except that the 'Time of Acknowledgment' parameter of the AcknowledgeAlarm service request shall be a sequence number.

Notes to Tester: A passing result is the same message sequence described as the passing result in 9.1.1.4. *If the IUT can only be configured with one recipient in the Recipient_List property of the issuing Notification_class object, omit steps 4 and 9.*

[Change **Clause 9.1.1.6**, p. 252]
[Reviewer Note: Made changes to allow cases where only one Recipient_List entry is supported.]

9.1.1.6 Successful Alarm Acknowledgment of Unconfirmed Event Notifications Using the Date Time Form of the 'Time of Acknowledgment' Parameter

Purpose: To verify the successful acknowledgment of an alarm signaled by an UnconfirmedEventNotification, including notification of other workstations and updating of the Acked_Transitions status. The Date Time form of the 'Time of Acknowledgment' parameter is used.

Test Concept: An alarm is triggered that causes the IUT to notify the TD and ~~at least~~ one other device. The TD acknowledges the alarm and verifies that the acknowledgment is properly noted by the IUT. The IUT notifies all other recipients that the alarm has been acknowledged.

Configuration Requirements: The IUT shall be configured with at least one object that can detect alarm conditions and send unconfirmed notifications. The Acked_Transitions property shall have the value B'111' indicating that all transitions have been acknowledged. The TD and ~~at least~~ one other BACnet device*, if the IUT supports multiple recipients,* shall be recipients of the alarm notification.

Test Steps: The test steps defined in 9.1.1.4 shall be followed except that the 'Time of Acknowledgment' parameter of the AcknowledgeAlarm service request shall convey the current time using a BACnetDateTime format.

Notes to Tester: A passing result is the same message sequence described as the passing result in 9.1.1.4. *If the IUT can only be configured with one recipient in the Recipient_List property of the issuing Notification_class object, omit steps 4 and 9.*

[Change **Clause 9.1.1.7**, p. 252]

9.1.1.7 Successful Alarm Acknowledgment of any "Offnormal" Transitions Using an "Offnormal" 'To State'

Purpose: To verify the successful acknowledgment of an alarm that indicated an "offnormal" 'To State' other than ~~offnormal~~*OFFNORMAL*.

Test Concept: An "offnormal" alarm is triggered in the IUT where the "offnormal" state is represented by an event-state other than ~~offnormal~~*OFFNORMAL* (such as ~~high-limit~~*HIGH_LIMIT* or ~~low-limit~~*LOW_LIMIT*). The TD acknowledges the alarm

with an 'Event State Acknowledged' of ~~offnormal~~*OFFNORMAL* and verifies that the acknowledgment is accepted by the IUT.

Configuration Requirements: The IUT shall be configured with at least one object that can detect alarm conditions and can enter "offnormal" states other than ~~offnormal~~*OFFNORMAL*. The TD shall be a recipient of the alarm notification and the IUT shall be configured to send it unconfirmed. ~~If the IUT cannot be configured to generate such a notification, then this test shall be skipped.~~

Test Steps:

1. MAKE (a change that triggers the detection of an ~~alarm~~*"offnormal"* event *other than OFFNORMAL* in the IUT)
2. *WAIT pTimeDelay*
3. RECEIVE ~~UnConfirmed~~UnconfirmedEventNotification-Request,
   'Process Identifier' = (the process identifier configured for this event),
   'Initiating Device Identifier' = IUT,
   'Event Object Identifier' = (the object detecting the alarm),
   'Time Stamp' = (~~the current time or sequence number~~*any valid time stamp*),
   'Notification Class' = (the notification class configured for this event),
   'Priority' = (the priority configured for this event),
   'Event Type' = (any valid event type),
   'Message Text' = *(optional, any valid message text),*
   'Notify Type' = (any valid notify type),
   'AckRequired' = TRUE,
   'From State' = (any valid event-state),
   'To State' = (any "offnormal" event state other than ~~offnormal~~*OFFNORMAL* itself),
   'Event Values' = (the values appropriate to the event type)
4. VERIFY (the 'Event Object Identifier' from the event notification), Acked_Transitions = {0,?,?}
5. TRANSMIT AcknowledgeAlarm-Request,
   'Acknowledging Process Identifier' = (the value of the 'Process Identifier' parameter in the event notification),
   'Event Object Identifier' = (the 'Event Object Identifier' from the event notification),
   'Event State Acknowledged' = offnormal,
   'Time Stamp' = (~~the timestamp conveyed in the notification~~*any valid time stamp*),
   '*Acknowledgement Source*' = *(any valid value),*
   'Time of Acknowledgment' = (any valid timestamp)
6. RECEIVE ~~BACnet-Simple-ACK-PDU~~*BACnet-SimpleACK-PDU*
7. IF (Protocol_Revision is present and Protocol_Revision ≥ 1) THEN
   RECEIVE UnconfirmedEventNotification-Request,
   'Process Identifier' = (the process identifier configured for this event),
   'Initiating Device Identifier' = IUT,
   'Event Object Identifier' = (the object detecting the alarm),
   'Time Stamp' = (~~the current time or sequence number~~*any valid time stamp*),
   'Notification Class' = (the notification class configured for this event),
   'Priority' = (the priority configured for this event),
   'Event Type' = (the event type included in step 2),
   'Message Text' = *(optional, any valid message text),*
   'Notify Type' = ACK_NOTIFICATION,
   'To State' = (~~offnormal~~*OFFNORMAL* or the 'To State' from step 2)
   ELSE
   RECEIVE UnconfirmedEventNotification-Request,
   'Process Identifier' = (the process identifier configured for this event),
   'Initiating Device Identifier' = IUT,
   'Event Object Identifier' = (the object detecting the alarm),
   'Time Stamp' = (~~the current time or sequence number~~*any valid time stamp*),
   'Notification Class' = (the notification class configured for this event),
   'Priority' = (the priority configured for this event),
   'Event Type' = (the event type included in step 2),

'Message Text' = *(optional, any valid message text),*
'Notify Type' = ACK_NOTIFICATION,

~~7~~*8*. VERIFY (the 'Event Object Identifier' from the event notification), Acked_Transitions = {1,?,?}

[Change **Clause 9.1.1.8**, p. 253]
[Reviewer Note: Added 'Notes to Tester' to clarify what to do if the TD only supports one recipient. Modified 'Configuration Requirements' to allow for only one recipient.]

9.1.1.8 Successful Alarm Acknowledgment of Confirmed Event Notifications Using an Unknown 'Acknowledging Process Identifier' Parameter

Purpose: To verify the successful acknowledgment of an alarm signaled by a ConfirmedEventNotification, when the acknowledgement contains a mismatched or unmatched 'Acknowledging Process Identifier' parameter.

Test Concept: An alarm is triggered that causes the IUT to notify the TD and at least one other device. The TD acknowledges the alarm with a mismatched 'Acknowledging Process Identifier' (the Process Identifier associated with another recipient), or an unknown 'Acknowledging Process Identifier' (a Process Identifier not associated with any recipient), and verifies that the acknowledgment is properly noted by the IUT. This test should be performed twice, once with a mismatched Process Identifier and once with an unknown Process Identifier.

Configuration Requirements: The IUT shall be configured with at least one object, Object1, *which*~~that~~ can detect alarm conditions and send confirmed notifications. The Acked_Transitions property shall have the value (TRUE,TRUE,TRUE), indicating that all transitions have been acknowledged. The TD and ~~at least~~ one other BACnet device*, if the IUT supports multiple recipients,* shall be recipients of the alarm notification, and shall use different Process Identifiers. *D1 is either the pTimeDelay or pTimeDelayNormal parameter or 0 (for transitions to and from FAULT state) depending on the event transition.*

This test is recommended for all BACnet devices that execute AcknowledgeAlarm but is required only for those that claim conformance to Protocol_Revision 5 or greater.

Test Steps:

1. VERIFY (Object1), Acked_Transitions = (TRUE,TRUE,TRUE)
2. MAKE (a change that triggers the detection of an alarm event in the IUT)
3. *WAIT D1*
~~3~~*4*. —BEFORE **Notification Fail Time**
RECEIVE ConfirmedEventNotification-Request,
'Process Identifier' = (any Process ID),
'Initiating Device Identifier' = IUT,
'Event Object Identifier' = Object1,
'Time Stamp' = (~~the current time or sequence number~~*any valid time stamp*),
'Notification Class' = (the Notification Class configured for this event),
'Priority' = (the priority configured for this event),
'Event Type' = (any valid event type),
'Message Text' = *(optional, any valid message text),*
'Notify Type' = ALARM or EVENT,
'AckRequired' = TRUE,
'From State' = (any appropriate event state),
'To State' = (any appropriate event state),
'Event Values' = (values appropriate to the event type)
~~4~~*5*. TRANSMIT BACnet-SimpleACK-PDU
~~5~~*6*. RECEIVE
DESTINATION = (~~at least one~~ *a* device other than the TD),
SOURCE = IUT,
ConfirmedEventNotification-Request,
'Process Identifier' = (any Process ID),
'Initiating Device Identifier' = IUT,

'Event Object Identifier' = Object1,
'Time Stamp' = (~~the current time or sequence number~~*any valid time stamp*),
'Notification Class' = (the notification class configured for this event),
'Priority' = (the priority configured for this event),
'Event Type' = (any valid event type),
'Message Text' = *(optional, any valid message text),*
'Notify Type' = ALARM | EVENT,
'AckRequired' = TRUE,
'From State' = (any appropriate event state),
'To State' = (any appropriate event state),
'Event Values' = (values appropriate to the event type)

~~6~~7. TRANSMIT
DESTINATION = IUT,
SOURCE = (DESTINATION in step 5),
BACnet-SimpleACK-PDU

~~7~~8. VERIFY (Object1), Acked_Transitions = (one bit FALSE, the others TRUE)

~~8~~9. TRANSMIT AcknowledgeAlarm-Request,
'Acknowledging Process Identifier' = (Any mismatched or unknown value),
'Event Object Identifier' = Object1,
'Event State Acknowledged' = (the state specified in the 'To State' parameter of the notification),
'Time Stamp' = (~~the timestamp conveyed in the notification~~*any valid time stamp*),
'*Acknowledgement Source*' = *(any valid value),*
'Time of Acknowledgment' = (the current timestamp)

~~9~~*10.*RECEIVE BACnet-SimpleACK-PDU

~~10~~*11.IF (Protocol_Revision is present and Protocol_Revision ≥ 1) THEN*
BEFORE **Notification Fail Time**
RECEIVE ConfirmedEventNotification-Request,
'Process Identifier' = (any Process ID),
'Initiating Device Identifier' = IUT,
'Event Object Identifier' = Object1,
'Time Stamp' = (~~the current time or sequence number~~*any valid time stamp*),
'Notification Class' = (the Notification Class configured for this event),
'Priority' = (the priority configured for this event),
'Event Type' = (any valid event type),
'Message Text' = *(optional, any valid message text),*
'Notify Type' = ACK_NOTIFICATION,
'To State' = (any appropriate event state)
ELSE
BEFORE **Notification Fail Time**
RECEIVE ConfirmedEventNotification-Request,
'Process Identifier' = (any Process ID),
'Initiating Device Identifier' = IUT,
'Event Object Identifier' = Object1,
'Time Stamp' = (~~the current time or sequence number~~*any valid time stamp*),
'Notification Class' = (the Notification Class configured for this event),
'Priority' = (the priority configured for this event),
'Event Type' = (any valid event type),
'Message Text' = *(optional, any valid message text),*
'Notify Type' = ACK_NOTIFICATION,

~~11~~*12*. TRANSMIT BACnet-SimpleACK-PDU

~~12~~*13*. *IF (Protocol_Revision is present and Protocol_Revision ≥ 1) THEN*
RECEIVE
DESTINATION = (~~at least one~~ *a* device other than the TD),
SOURCE = IUT,
ConfirmedEventNotification-Request,
'Process Identifier' = (any Process ID),

'Initiating Device Identifier' = IUT,
'Event Object Identifier' = Object1,
'Time Stamp' = (~~the current time or sequence number~~*any valid time stamp*),
'Notification Class' = (the notification class configured for this event),
'Priority' = (the priority configured for this event),
'Event Type' = (any valid event type),
'Message Text' = (optional, any valid message text),
'Notify Type' = ACK_NOTIFICATION,
'To State' = (any appropriate event state)

ELSE
RECEIVE
DESTINATION = (a device other than the TD),
SOURCE = IUT,
ConfirmedEventNotification-Request,
'Process Identifier' = (any Process ID),
'Initiating Device Identifier' = IUT,
'Event Object Identifier' = Object1,
'Time Stamp' = (any valid time stamp),
'Notification Class' = (the notification class configured for this event),
'Priority' = (the priority configured for this event),
'Event Type' = (any valid event type),
'Message Text' = (optional, any valid message text),
'Notify Type' = ACK_NOTIFICATION

~~13~~*14*. TRANSMIT
DESTINATION = IUT,
SOURCE = (DESTINATION in step 5),
BACnet-SimpleACK-PDU

~~14~~*15*. VERIFY (Object1), Acked_Transitions = (TRUE,TRUE,TRUE)

Notes to Tester: The ConfirmedEventNotification-Request messages can be received in either order. *If the IUT can only be configured with one recipient in the Recipient_List property of the issuing Notification_class object, omit steps 5 and 6.*

[Change **Clause 9.1.1.9**, p. 255]
[Reviewer Note: Added 'Notes to Tester' to handle cases with only one recipient. Updated 'Test Concept' to handle cases with only one recipient.]

9.1.1.9 Successful Alarm Acknowledgment of Unconfirmed Event Notifications Using an Unknown 'Acknowledging Process Identifier' Parameter

Purpose: To verify the successful acknowledgment of an alarm signaled by an UnconfirmedEventNotification, when the acknowledgement contains a mismatched or unmatched 'Acknowledging Process Identifier' parameter.

Test Concept: An alarm is triggered that causes the IUT to notify the TD and at least one other device. The TD acknowledges the alarm with a mismatched 'Acknowledging Process Identifier' (the Process Identifier associated with another recipient), or unknown (a Process Identifier not associated with any recipient), and verifies that the acknowledgment is properly noted by the IUT. This test should be performed twice, once with a mismatched Process Identifier and once with an unknown Process Identifier.

Configuration Requirements: The IUT shall be configured with at least one object, Object1, that can detect alarm conditions and send unconfirmed notifications. The Acked_Transitions property shall have the value (TRUE,TRUE,TRUE), indicating that all transitions have been acknowledged. The TD and at least one other BACnet device, *if the IUT supports multiple recipients,* shall be recipients of the alarm notification, configured to receive different Process Identifiers. *D1 is either the pTimeDelay or pTimeDelayNormal parameter or 0 (for transitions to and from FAULT state) depending on the event transition.*

This test is recommended for all BACnet devices that execute AcknowledgeAlarm but is required only for those that claim conformance to Protocol_Revision 5 or greater.

Test Steps:

1. VERIFY (Object1), Acked_Transitions = (TRUE,TRUE,TRUE)
2. MAKE (a change that triggers the detection of an alarm event in the IUT)
3. *WAIT D1*
~~3~~4. BEFORE **Notification Fail Time**
RECEIVE UnconfirmedEventNotification-Request,
'Process Identifier' = (any Process ID),
'Initiating Device Identifier' = IUT,
'Event Object Identifier' = Object1,
'Time Stamp' = (~~the current time or sequence number~~*any valid time stamp*),
'Notification Class' = (the Notification Class configured for this event),
'Priority' = (the priority configured for this event),
'Event Type' = (any valid event type),
'Message Text' = *(optional, any valid message text),*
'Notify Type' = ALARM or EVENT,
'AckRequired' = TRUE,
'From State' = (any appropriate event state),
'To State' = (any appropriate event state),
'Event Values' = (values appropriate to the event type)
~~4~~5. RECEIVE
DESTINATION = (at least one device other than the TD),
SOURCE = IUT,
UnconfirmedEventNotification-Request,
'Process Identifier' = (any Process ID),
'Initiating Device Identifier' = IUT,
'Event Object Identifier' = (the object detecting the alarm),
'Time Stamp' = (~~the current time or sequence number~~*any valid time stamp*),
'Notification Class' = (the notification class configured for this event),
'Priority' = (the priority configured for this event),
'Event Type' = (any valid event type),
'Message Text' = *(optional, any valid message text),*
'Notify Type' = ALARM | EVENT,
'AckRequired' = TRUE,
'From State' = (any appropriate event state),
'To State' = (any appropriate event state),
'Event Values' = (values appropriate to the event type)
~~5~~6. VERIFY (Object1), Acked_Transitions = (one bit FALSE, the others TRUE)
~~6~~7. TRANSMIT AcknowledgeAlarm-Request,
'Acknowledging Process Identifier' = (Any mismatched or unknown value),
'Event Object Identifier' = Object1,
'Event State Acknowledged' = (the state specified in the 'To State' parameter of the notification),
'Time Stamp' = (~~the timestamp conveyed in the notification~~*any valid time stamp*),
'*Acknowledgement Source*' = *(any valid value),*
'Time of Acknowledgment' = (the current timestamp)
~~7~~8. RECEIVE BACnet-SimpleACK-PDU
~~8~~9. ~~—~~BEFORE **Notification Fail Time**
RECEIVE UnconfirmedEventNotification-Request,
'Process Identifier' = (any Process ID),
'Initiating Device Identifier' = IUT,
'Event Object Identifier' = Object1,
'Time Stamp' = (~~the current time or sequence number~~*any valid time stamp*),
'Notification Class' = (the Notification Class configured for this event),
'Priority' = (the priority configured for this event),
'Event Type' = (any valid event type),

'*Message Text*' = *(optional, any valid message text),*
'Notify Type' = ACK_NOTIFICATION,
'To State' = (any appropriate event state)

~~9~~*10*.RECEIVE
DESTINATION = ~~-~~(at least one device other than the TD),
SOURCE = IUT,
UnconfirmedEventNotification-Request,
'Process Identifier' = (any Process ID),
'Initiating Device Identifier' = IUT,
'Event Object Identifier' = Object1,
'Time Stamp' = (~~the current time or sequence number~~*any valid time stamp*),
'Notification Class' = (the notification class configured for this event),
'Priority' = (the priority configured for this event),
'Event Type' = (any valid event type),
'*Message Text*' = *(optional, any valid message text),*
'Notify Type' = ACK_NOTIFICATION,
'To State' = (any appropriate event state)

~~10~~*11*. VERIFY (Object1), Acked_Transitions = (TRUE,TRUE,TRUE)

Note to Tester: The UnconfirmedEventNotification-Request messages can be received in either order. *If the IUT can only be configured with one recipient in the Recipient_List property of the issuing Notification_class object, omit step 4.*

[Change **Clause 9.1.1.10**, p. 257]
[Reviewer Note: Made changes to allow cases where only one Recipient_List entry is supported.]

9.1.1.10 Successful Alarm Re-Acknowledgment of Confirmed Event Notifications

Purpose: To verify the successful re-acknowledgment of an event signaled by a ConfirmedEventNotification, including notification of other workstations and updating of the Acked_Transitions status.

Test Concept: An event is triggered and~~, after the specified Time_Delay of the event-generating object,~~ the IUT notifies the TD and ~~at least~~ one other device. The TD acknowledges the event and verifies that the acknowledgment is properly noted by the IUT. The IUT notifies all recipients that the event has been acknowledged. The TD then acknowledges the event again, and the IUT again notifies all recipients. This behavior was not defined before Protocol_Revision 7 and so this test shall only be performed if Protocol_Revision is present (i.e., Protocol_Revision e 7).

Configuration Requirements: The IUT shall be configured with at least one event-initiating object that generates offnormal transitions and sends confirmed notifications. The Acked_Transitions property shall have the value B'111', indicating that all transitions have been acknowledged. The TD and ~~at least~~ one other BACnet device, *if the IUT supports multiple recipients,* shall be recipients of the event notification.

Test Steps:

1. MAKE (a change that triggers the detection of an event in the IUT)
2. WAIT (~~Time_Delay~~*pTimeDelay*)
3. BEFORE **Notification Fail Time**
   RECEIVE ConfirmedEventNotification-Request,
   'Process Identifier' = (the process identifier configured for this event),
   'Initiating Device Identifier' = IUT,
   'Event Object Identifier' = (the event-initiating object),
   'Time Stamp' = (~~the current time or sequence number~~*any valid time stamp*),
   'Notification Class' = (the notification class configured for this event),
   'Priority' = (the priority configured for this event),
   'Event Type' = (any valid event type),
   '*Message Text*' = *(optional, any valid message text),*

'Notify Type' = (the notify type configured for this event),
'AckRequired' = TRUE,
'From State' = NORMAL,
'To State' = (any appropriate offnormal event state),
'Event Values' = (the values appropriate to the event type)

4. TRANSMIT BACnet-SimpleACK-PDU
5. RECEIVE
   DESTINATION = (~~at least one~~ *a* device other than the TD),
   SOURCE = IUT,
   ConfirmedEventNotification-Request,
   'Process Identifier' = (the process identifier configured for this event),
   'Initiating Device Identifier' = IUT,
   'Event Object Identifier' = (the event-initiating object),
   'Time Stamp' = (the timestamp or sequence number received in step 3),
   'Notification Class' = (the notification class configured for this event),
   'Priority' = (the priority configured for this event),
   'Event Type' = (any valid event type),
   'Message Text' = *(optional, any valid message text),*
   'Notify Type' = (the notify type configured for this event),
   'AckRequired' = TRUE,
   'From State' = NORMAL,
   'To State' = (any appropriate offnormal event state),
   'Event Values' = (the values appropriate to the event type)
6. TRANSMIT BACnet-SimpleACK-PDU
7. VERIFY (the 'Event Object Identifier' from the event notification), Acked_Transitions = ~~B'011'~~*(FALSE, TRUE, TRUE)*
8. TRANSMIT AcknowledgeAlarm-Request,
   'Acknowledging Process Identifier' = (the value of the 'Process Identifier' parameter in the event notification),
   'Event Object Identifier' = (the 'Event Object Identifier' from the event notification),
   'Event State Acknowledged' = (the state specified in the 'To State' parameter of the notification),
   'Time Stamp' = (~~the time stamp conveyed in the notification~~*any valid time stamp*),
   'Acknowledgment Source' = ~~(a character string)~~*(any valid value),*
   'Time of Acknowledgment' = (any of the forms specified for this parameter)
9. RECEIVE BACnet-~~Simple ACK~~*SimpleACK*-PDU
10. BEFORE **Notification Fail Time**
    RECEIVE ConfirmedEventNotification-Request,
    'Process Identifier' = (the process identifier configured for this event),
    'Initiating Device Identifier' = IUT,
    'Event Object Identifier' = (the event-initiating object),
    'Time Stamp' = (~~the current time or sequence number~~*any valid time stamp*),
    'Notification Class' = (the notification class configured for this event),
    'Priority' = (the priority configured for this event),
    'Event Type' = (the event type included in step 3),
    'Message Text' = *(optional, any valid message text),*
    'Notify Type' = ACK_NOTIFICATION,
    'To State' = (the 'To State' used in step 3)
11. TRANSMIT BACnet-SimpleACK-PDU
12. RECEIVE
    DESTINATION = (~~at least one~~ *a* device other than the TD),
    SOURCE = IUT,
    ConfirmedEventNotification-Request,
    'Process Identifier' = (the process identifier configured for this event),
    'Initiating Device Identifier' = IUT,
    'Event Object Identifier' = (the event-initiating object),
    'Time Stamp' = (the timestamp or sequence number received in step 10),
    'Notification Class' = (the notification class configured for this event),
    'Priority' = (the priority configured for this event),

'Event Type' = (the event type included in step 3),
'Message Text' = *(optional, any valid message text),*
'Notify Type' = ACK_NOTIFICATION,
'To State' = (the 'To State' used in step 3)

13. TRANSMIT BACnet-SimpleACK-PDU
14. VERIFY (the 'Event Object Identifier' from the event notification), Acked_Transitions = ~~B'111'~~*(TRUE, TRUE, TRUE)*
15. TRANSMIT AcknowledgeAlarm-Request,
    'Acknowledging Process Identifier' = (the value of the 'Process Identifier' parameter in the event notification),
    'Event Object Identifier' = (the 'Event Object Identifier' from the event notification),
    'Event State Acknowledged' = (the state specified in the 'To State' parameter of the notification),
    'Time Stamp' = (~~the time stamp conveyed in the notification~~*any valid time stamp*),
    'Acknowledgment Source' = (a character string)
    'Time of Acknowledgment' = (any of the forms specified for this parameter)
16. RECEIVE BACnet-SimpleACK-PDU
17. BEFORE **Notification Fail Time**
    RECEIVE ConfirmedEventNotification-Request,
    'Process Identifier' = (the process identifier configured for this event),
    'Initiating Device Identifier' = IUT,
    'Event Object Identifier' = (the event-initiating object),
    'Time Stamp' = (~~the current time or sequence number~~*any valid time stamp*),
    'Notification Class' = (the notification class configured for this event),
    'Priority' = (the priority configured for this event),
    'Event Type' = (the event type included in step 3),
    'Message Text' = *(optional, any valid message text),*
    'Notify Type' = ACK_NOTIFICATION,
    'To State' = (the 'To State' used in step 3)
18. TRANSMIT BACnet-SimpleACK-PDU
19. RECEIVE
    DESTINATION = (~~at least one~~ *a* device other than the TD),
    SOURCE = IUT,
    ConfirmedEventNotification-Request,
    'Process Identifier' = (the process identifier configured for this event),
    'Initiating Device Identifier' = IUT,
    'Event Object Identifier' = (the event-initiating object),
    'Time Stamp' = (the timestamp or sequence number received in step 17),
    'Notification Class' = (the notification class configured for this event),
    'Priority' = (the priority configured for this event),
    'Event Type' = (the event type included in step 3),
    'Message Text' = *(optional, any valid message text),*
    'Notify Type' = ACK_NOTIFICATION,
    'To State' = (the 'To State' used in step 3)
20. TRANSMIT BACnet-SimpleACK-PDU
21. VERIFY (the 'Event Object Identifier' from the event notification), Acked_Transitions = ~~B'111'~~*(TRUE, TRUE, TRUE)*

Notes to Tester: The destination address used for the acknowledgment notification in steps 12 and 19 shall be the same address used in step 4. When multiple event notifications are expected for a specific event, the order that the IUT transmits them in is irrelevant. *If the IUT can only be configured with one recipient in the Recipient_List property of the issuing Notification_class object, omit steps 5, 6, 12, 13, 19, and 20.*

[Change **Clause 9.1.1.11**, p. 259]
[Reviewer Note: Made changes to allow cases where only one Recipient_List entry is supported.]

9.1.1.11 Successful Alarm Re-Acknowledgment of Unconfirmed Event Notifications

Purpose: To verify the successful re-acknowledgment of an event signaled by an UnconfirmedEventNotification, including notification of other workstations and updating of the Acked_Transitions status.

Test Concept: An event is triggered and~~, after the specified Time_Delay of the event-generating object,~~ the IUT notifies the TD and ~~at least~~ one other device. The TD acknowledges the event and verifies that the acknowledgment is properly noted by the IUT. The IUT notifies all recipients that the event has been acknowledged. The TD then acknowledges the event again, and the IUT again notifies all recipients. This behavior was not defined before Protocol_Revision 7 and so this test shall only be performed if Protocol_Revision is present (i.e., Protocol_Revision ≥ 7).

Configuration Requirements: The IUT shall be configured with at least one event-initiating object that generates offnormal transitions and sends unconfirmed notifications. The Acked_Transitions property shall have the value B'111', indicating that all transitions have been acknowledged. The TD and ~~at least~~ one other BACnet device, *if the IUT supports multiple recipients,* shall be recipients of the event notification.

Test Steps:

1. MAKE (a change that triggers the detection of an offnormal event in the IUT)
2. WAIT (~~Time_Delay~~*pTimeDelay*)
3. BEFORE **Notification Fail Time**
   RECEIVE UnconfirmedEventNotification-Request,
   'Process Identifier' = (the process identifier configured for this event),
   'Initiating Device Identifier' = IUT,
   'Event Object Identifier' = (the event-initiating object),
   'Time Stamp' = (~~the current time or sequence number~~*any valid time stamp*),
   'Notification Class' = (the notification class configured for this event),
   'Priority' = (the priority configured for this event),
   'Event Type' = (any valid event type),
   'Message Text' = *(optional, any valid message text),*
   'Notify Type' = (the notify type configured for this event),
   'AckRequired' = TRUE,
   'From State' = NORMAL,
   'To State' = (any appropriate offnormal event state),
   'Event Values' = (the values appropriate to the event type)
4. RECEIVE
   DESTINATION = (at least one device other than the TD),
   SOURCE = IUT,
   UnconfirmedEventNotification-Request,
   'Process Identifier' = (the process identifier configured for this event),
   'Initiating Device Identifier' = IUT,
   'Event Object Identifier' = (the event-initiating object)
   'Time Stamp' = (the timestamp or sequence number received in step 3),
   'Notification Class' = (the notification class configured for this event),
   'Priority' = (the priority configured for this event),
   'Event Type' = (any valid event type),
   'Message Text' = *(optional, any valid message text),*
   'Notify Type' = (the notify type configured for this event),
   'AckRequired' = TRUE,
   'From State' = NORMAL,
   'To State' = (any appropriate offnormal event state),
   'Event Values' = (the values appropriate to the event type)
5. VERIFY (the 'Event Object Identifier' from the event notification), Acked_Transitions = ~~B'011'~~*(FALSE, TRUE, TRUE)*
6. TRANSMIT AcknowledgeAlarm-Request,
   'Acknowledging Process Identifier' = (the value of the 'Process Identifier' parameter in the event notification),
   'Event Object Identifier' = (the 'Event Object Identifier' from the event notification),
   'Event State Acknowledged' = (the state specified in the 'To State' parameter of the notification),
   'Time Stamp' = (~~the time stamp conveyed in the notification~~*any vaid time stamp*),
   'Acknowledgment Source' = ~~(a character string)~~*(any valid value),*
   'Time of Acknowledgment' = (any of the forms specified for this parameter)

7. RECEIVE BACnet-SimpleACK-PDU
8. BEFORE **Notification Fail Time**
   RECEIVE UnconfirmedEventNotification-Request,
   'Process Identifier' = (the process identifier configured for this event),
   'Initiating Device Identifier' = IUT,
   'Event Object Identifier' = (the event-initiating object),
   'Time Stamp' = (~~the current time or sequence number~~*any valid time stamp*),
   'Notification Class' = (the notification class configured for this event),
   'Priority' = (the priority configured for this event),
   'Event Type' = (the event type included in step 3),
   'Message Text' = *(optional, any valid message text),*
   'Notify Type' = ACK_NOTIFICATION,
   'To State' = (the 'To State' used in step 3)
9. RECEIVE
   DESTINATION = (at least one device other than the TD),
   SOURCE = IUT,
   UnconfirmedEventNotification-Request,
   'Process Identifier' = (the process identifier configured for this event),
   'Initiating Device Identifier' = IUT,
   'Event Object Identifier' = (the event-initiating object),
   'Time Stamp' = (the timestamp or sequence number received in step 8),
   'Notification Class' = (the notification class configured for this event),
   'Priority' = (the priority configured for this event),
   'Event Type' = (the event type included in step 3),
   'Message Text' = *(optional, any valid message text),*
   'Notify Type' = ACK_NOTIFICATION,
   'To State' = (the 'To State' used in step 3)
10. VERIFY (the 'Event Object Identifier' from the event notification), Acked_Transitions = ~~B'111'~~*(TRUE, TRUE, TRUE)*
11. TRANSMIT AcknowledgeAlarm-Request,
    'Acknowledging Process Identifier' = (the value of the 'Process Identifier' parameter in the event notification),
    'Event Object Identifier' = (the 'Event Object Identifier' from the event notification),
    'Event State Acknowledged' = (the state specified in the 'To State' parameter of the notification),
    'Time Stamp' = (~~the time stamp conveyed in the notification~~*any valid time stamp*),
    'Acknowledgment Source' = (a character string)
    'Time of Acknowledgment' = (any of the forms specified for this parameter)
12. RECEIVE BACnet-SimpleACK-PDU
13. BEFORE **Notification Fail Time**
    RECEIVE UnconfirmedEventNotification-Request,
    'Process Identifier' = (the process identifier configured for this event),
    'Initiating Device Identifier' = IUT,
    'Event Object Identifier' = (the event-initiating object),
    'Time Stamp' = (~~the current time or sequence number~~*any valid time stamp*),
    'Notification Class' = (the notification class configured for this event),
    'Priority' = (the priority configured for this event),
    'Event Type' = (the event type included in step 3),
    'Message Text' = *(optional, any valid message text),*
    'Notify Type' = ACK_NOTIFICATION,
    'To State' = (the 'To State' used in step 3)
14. RECEIVE
    DESTINATION = (at least one device other than the TD),
    SOURCE = IUT,
    UnconfirmedEventNotification-Request,
    'Process Identifier' = (the process identifier configured for this event),
    'Initiating Device Identifier' = IUT,
    'Event Object Identifier' = (the event-initiating object),
    'Time Stamp' = (the timestamp or sequence number received in step 13),

'Notification Class' =	(the notification class configured for this event),
'Priority' =	(the priority configured for this event),
'Event Type' =	(the event type included in step 3),
'Message Text' =	*(optional, any valid message text),*
'Notify Type' =	ACK_NOTIFICATION,
'To State' =	(the 'To State' used in step 3)

15. VERIFY (the 'Event Object Identifier' from the event notification), Acked_Transitions = ~~B'111'~~*(TRUE, TRUE, TRUE)*

Notes to Tester: The destination address used for the acknowledgment notification in steps 9 and 14 shall be the same address used in step 4. When multiple event notifications are expected for a specific event, the order that the IUT transmits them in is irrelevant. *If the IUT can only be configured with one recipient in the Recipient_List property of the issuing Notification_class object, omit steps 4, 9, and 14.*

[Add new **Clause 9.1.1.14**, p. 261]

9.1.1.14 Successful Alarm Acknowledgment of Confirmed Event Notifications when 'To State' is either High-Limit or Low-Limit, Revision 5 and higher only

Purpose: To verify the successful acknowledgment of an alarm signaled by a ConfirmedEventNotification, including notification of other workstations and updating of the Acked_Transitions status when the 'To State' parameter is either High-Limit or Low-Limit and the 'Event State Acknowledged' parameter is Off-Normal.

Test Concept: An alarm is triggered that causes the IUT to notify the TD and one other device with a~~n~~ 'To State' event of either High-Limit or Low-Limit. The TD acknowledges the alarm using all of the correct parameters and using an 'Event State Acknowledged' parameter of 'Off-Normal' -and verifies that the acknowledgment is properly noted by the IUT. The IUT notifies all other recipients that the alarm has been acknowledged.

Configuration Requirements: The IUT shall be configured with at least one object that can detect alarm conditions and send confirmed notifications. The Acked_Transitions property shall have the value B'111' indicating that all transitions have been acknowledged. The TD and one other BACnet device, if the IUT supports multiple recipients, shall be recipients of the alarm notification.

Test Steps: The test steps defined in 9.1.1.1 shall be followed except that the 'To State' parameter shall be either High-Limit or Low-Limit. When acknowledging the alarm, the TD shall use an 'Event State Acknowledged' parameter of Off-Normal.

Notes to Tester: A passing result is the same message sequence described as the passing result in 9.1.1.1. If the IUT can only be configured with one recipient in the Recipient_List property of the issuing Notification_class object, skip all steps related to receipt of the second notification.

[Add new **Clause 9.1.1.15**, p. 261]
[Reviewer Note: Addendum 135-2010af added language to ensure that notifications are not ignored due to unsupported character sets.]

9.1.1.15 Unsupported Message Text Character Set AcknowledgeAlarm Test

Purpose: To verify that the IUT does not fail to process an AcknowledgeAlarm request because the Acknowledgment Source parameter is of a character set that the IUT does not support.

Test Concept: Cause an event-initiating object, O1, in the IUT to transition to Event_State ES1. Acknowledge the transition and, in the AcknowledgeAlarm service, provide an ‘Acknowledgment Source’ parameter, AS1, which has a character set that the IUT does not support. Verify that the IUT processes the request even if the ‘Acknowledgment Source’ uses a character set that the IUT does not support, and that the IUT accepts and applies that Acknowledgment request, irrespective of the ‘Acknowledgment Source’.

Configuration Requirements: Configure an event-initiating object, O1, which references a Notification Class object, N1. Configure O1 such that it needs an acknowledgment when it transitions out of its current state. D1 is either the pTimeDelay, or pTimeDelayNormal parameter, or 0 (for transitions to and from FAULT state) depending on the event transition. AS1 shall be a character string short enough for the IUT to receive and encoded in a character set that the IUT does not support. If the IUT supports all character sets, this test shall be skipped.

Test Steps:

1. MAKE (a condition exist that will cause O1 to create a transition)
2. WAIT D1
3. BEFORE **Notification Fail Time**
   RECEIVE UnconfirmedEventNotification-Request
   'Process Identifier' = (any valid process identifier),
   'Initiating Device Identifier' = IUT,
   'Event Object Identifier' = O1,
   'Time Stamp' = (TS1: any valid timestamp),
   Notification Class' = (N1: the Notification_Class configured in O1),
   'Priority' = (any valid priority),
   'Event Type' = (any standard event type),
   'Message Text' = (optional, any valid message text),
   'Notify Type' = ALARM | EVENT,
   'AckRequired' = TRUE,
   'From State' = (any valid event state),
   'To State' = (ES1: any valid event state),
   'Event Values' = (any values appropriate to the event type)
4. IF (ES1 = NORMAL) THEN
   VERIFY Acked_Transitions = (?,?,F)
   ELSE IF (ES1 = FAULT) THEN
   VERIFY Acked_Transitions = (?,F,?)
   ELSE
   VERIFY Acked_Transitions = (F,?,?)
5. TRANSMIT AcknowledgeAlarm-Request
   'Acknowledging Process Identifier' = (any valid value),
   'Event Object Identifier' = O1,
   'Event State Acknowledged' = ES1,
   'Time Stamp' = TS1,
   'Acknowledgment Source' = AS1,
   'Time of Acknowledgment' = (any valid timestamp)
6. RECEIVE BACnet-SimpleACK-PDU
7. BEFORE **Notification Fail Time**
   RECEIVE UnconfirmedEventNotification-Request
   'Process Identifier' = (any valid process identifier),
   'Initiating Device Identifier' = IUT,
   'Event Object Identifier' = O1,
   'Time Stamp' = TS1
   Notification Class' = (N1: the Notification_Class configured in O1),
   'Priority' = (any valid priority),
   'Event Type' = (any standard event type),
   'Message Text' = (optional, any valid message text),
   'Notify Type' = ACK_NOTIFICATION,
   'To State' = ES1
8. IF (ES1 = NORMAL) THEN
   VERIFY Acked_Transitions = (?,?,T)
   ELSE IF (ES1 = FAULT) THEN
   VERIFY Acked_Transitions = (?,T,?)
   ELSE

VERIFY Acked_Transitions = (T,?,?)

Notes to Tester: The use of UnconfirmedEventNotification is specified in this test, solely to simplify the expression of the test. The behavior being tested applies to the ConfirmedEventNotification service as well.

[Change **Clause 9.1.2.1**, p. 262]
[Reviewer Note: Made changes to allow cases where only one Recipient_List entry is supported.]

9.1.2.1 Unsuccessful Alarm Acknowledgment of Confirmed Event Notifications Because the 'Time Stamp' is Too Old

Purpose: To verify that an alarm remains unacknowledged if the time stamp in the acknowledgment does not match the most recent transition to the current alarm state.

Test Concept: An alarm is triggered that causes the IUT to notify the TD and ~~at least~~ one other device. The TD acknowledges the alarm using an old time stamp and verifies that the acknowledgment is not accepted by the IUT and that the IUT does not notify other devices that the alarm was acknowledged. The TD then acknowledges the alarm using the proper time stamp and verifies that the acknowledgment is properly noted by the IUT. The IUT notifies all other recipients that the alarm was acknowledged.

Configuration Requirements: The IUT shall be configured with at least one object that can detect alarm conditions and send confirmed notifications. The Acked_Transitions property shall have the value B'111' indicating that all transitions have been acknowledged. The TD and ~~at least~~ one other BACnet device, *if the IUT supports multiple recipients,* shall be recipients of the alarm notification.

Test Steps:

1. MAKE (a change that triggers the detection of an alarm event in the IUT)
2. WAIT (~~Time_Delay~~*pTimeDelay*)
3. BEFORE **Notification Fail Time**
   RECEIVE ConfirmedEventNotification-Request,
   'Process Identifier' = (the process identifier configured for this event),
   'Initiating Device Identifier' = IUT,
   'Event Object Identifier' = (the object detecting the alarm),
   'Time Stamp' = (~~the current time or sequence number~~*any valid time stamp*),
   'Notification Class' = (the notification class configured for this event),
   'Priority' = (the priority configured for this event type),
   'Event Type' = (any valid event type),
   'Message Text' = *(optional, any valid message text),*
   'Notify Type' = (the notify type configured for the event),
   'AckRequired' = TRUE,
   'From State' = NORMAL,
   'To State' = (any appropriate non-normal event state),
   'Event Values' = (the values appropriate to the event type)
4. TRANSMIT BACnet-SimpleACK-PDU
5. RECEIVE
   DESTINATION = (~~at least one~~ *a* device other than the TD),
   SOURCE = IUT,
   ConfirmedEventNotification-Request,
   'Process Identifier' = (the process identifier configured for this event),
   'Initiating Device Identifier' = IUT,
   'Event Object Identifier' = (the object detecting the alarm),
   'Time Stamp' = (the timestamp or sequence number received in step 3),
   'Notification Class' = (the notification class configured for this event),
   'Priority' = (the priority configured for this event type),
   'Event Type' = (any valid event type),

'*Message Text*' = *(optional, any valid message text),*
'Notify Type' = (the notify type configured for the event),
'AckRequired' = TRUE,
'From State' = NORMAL,
'To State' = (any appropriate non-normal event state),
'Event Values' = (the values appropriate to the event type)

6. TRANSMIT BACnet-SimpleACK-PDU
7. VERIFY (the 'Event Object Identifier' from the event notification), Acked_Transitions = (FALSE,TRUE,TRUE)
8. TRANSMIT AcknowledgeAlarm-Request,
   'Acknowledging Process Identifier' = (the value of the 'Process Identifier' parameter in the event notification),
   'Event Object Identifier' = (the 'Event Object Identifier' from the event notification),
   'Event State Acknowledged' = (the state specified in the 'To State' parameter of the notification),
   'Time Stamp' = (~~a time stamp older than the one conveyed in the notification~~*any valid time stamp*),
   '*Acknowledgement Source*' = *(any valid value),*
   'Time of Acknowledgment' = (the current time using a Time format)
9. RECEIVE BACnet-Error-PDU
   Error Class = SERVICES,
   Error Code = INVALID_TIME_STAMP
10. VERIFY (the 'Event Object Identifier' from the event notification), Acked_Transitions = (FALSE,TRUE,TRUE)
11. TRANSMIT AcknowledgeAlarm-Request,
   'Acknowledging Process Identifier' = (the process identifier configured for this event),
   'Event Object Identifier' = (the 'Event Object Identifier' from the event notification),
   'Event State Acknowledged' = (the state specified in the 'To State' parameter of the notification),
   'Time Stamp' = (~~the time stamp conveyed in the notification~~*any valid time stamp*),
   '*Acknowledgement Source*' = *(any valid value),*
   'Time of Acknowledgment' = (the current time using a Time format)
12. RECEIVE BACnet-~~Simple-ACK~~*SimpleACK*-PDU
13. IF (Protocol_Revision is present and Protocol_Revision ≥ 1) THEN
    BEFORE **Notification Fail Time**
    RECEIVE
    ConfirmedEventNotification-Request,
    'Process Identifier' = (the process identifier configured for this event),
    'Initiating Device Identifier' = IUT,
    'Event Object Identifier' = (the object detecting the alarm),
    'Time Stamp' = (~~the current time or sequence number~~*any valid time stamp*),
    'Notification Class' = (the notification class configured for this event),
    'Priority' = (the priority configured for this event type),
    'Event Type' = (any valid event type),
    'Message Text' = *(optional, any valid message text),*
    'Notify Type' = ACK_NOTIFICATION,
    'To State' = (the 'To State' used in step 3 or 5)

    ELSE
    BEFORE **Notification Fail Time**
    RECEIVE
    ConfirmedEventNotification-Request,
    'Process Identifier' = (the process identifier configured for this event),
    'Initiating Device Identifier' = IUT,
    'Event Object Identifier' = (the object detecting the alarm),
    'Time Stamp' = (~~the current time or sequence number~~*any valid time stamp*),
    'Notification Class' = (the notification class configured for this event),
    'Priority' = (the priority configured for this event type),
    'Event Type' = (any valid event type),
    'Message Text' = *(optional, any valid message text),*
    'Notify Type' = ACK_NOTIFICATION
14. TRANSMIT BACnet-SimpleACK-PDU

15. IF (Protocol_Revision is present and Protocol_Revision ≥ 1) THEN
    RECEIVE
        DESTINATION = *(~~at least one~~ a device other than the TD),*
        SOURCE = *IUT,*
        ConfirmedEventNotification-Request,
            'Process Identifier' = (the process identifier configured for this event),
            'Initiating Device Identifier' = IUT,
            'Event Object Identifier' = (the object detecting the alarm),
            'Time Stamp' = (the timestamp or sequence number from the notification in step 13),
            'Notification Class' = (the notification class configured for this event),
            'Priority' = (the priority configured for this event type),
            'Event Type' = (any valid event type),
            'Message Text' = (optional, any valid message text),
            'Notify Type' = ACK_NOTIFICATION,
            'To State' = (the 'To State' used in step 3 or 5)
    ELSE
    RECEIVE
        DESTINATION = (~~at least one~~ *a* device other than the TD),
        SOURCE = IUT,
        ConfirmedEventNotification-Request,
            'Process Identifier' = (the process identifier configured for this event),
            'Initiating Device Identifier' = IUT,
            'Event Object Identifier' = (the object detecting the alarm),
            'Time Stamp' = (the timestamp or sequence number from the notification in step 13),
            'Notification Class' = (the notification class configured for this event),
            'Priority' = (the priority configured for this event type),
            'Event Type' = *(*any valid event type),
            'Message Text' = (optional, any valid message text),
            'Notify Type' = ACK_NOTIFICATION
16. TRANSMIT BACnet-SimpleACK-PDU
17. VERIFY (the 'Event Object Identifier' from the event notification), Acked_Transitions = (TRUE,TRUE,TRUE)

Notes to Tester: The destination address used for the acknowledgment notification in step 11 shall be the same address used in step 3. *If the IUT can only be configured with one recipient in the Recipient_List property of the issuing Notification_class object, omit steps 5, 6, 15, and 16.*

[Change **Clause 9.1.2.3**, p. 264]
[Reviewer Note: This test was updated to account for revision 5 specifications. There is no new SSPC proposal.]

9.1.2.3 Unsuccessful Alarm Acknowledgment of Confirmed Event Notifications Because the ~~Referenced Object Does Not Exist~~*'Event Object Identifier' is Invalid*

Purpose: To verify that an alarm remains unacknowledged if the 'Event Object Identifier' represents an object that does not exist *or is not consistent with the other parameters that define the alarm being acknowledged*.

Test Concept: An alarm is triggered that causes the IUT to notify the TD and ~~at least~~ one other device. The TD acknowledges the alarm using an improper 'Event Object Identifier' and verifies that the acknowledgment is not accepted by the IUT and that the IUT does not notify other devices that the alarm was acknowledged. The TD then acknowledges the alarm using the proper 'Event Object Identifier' and verifies that the acknowledgment is properly noted by the IUT. The IUT notifies all other recipients that the alarm was acknowledged.

Configuration Requirements: The IUT shall be configured with at least one object that can detect alarm conditions and send confirmed notifications. The Acked_Transitions property shall have the value B'111' indicating that all transitions have been acknowledged. The TD and ~~at least~~ one other BACnet device, *if the IUT supports multiple recipients,* shall be recipients of the alarm notification.

Test Steps: The test steps defined in 9.1.2.1 shall be followed except that in the first AcknowledgeAlarm request the 'Time Stamp' shall have the same value as the 'Time Stamp' from the event notification and the 'Event Object Identifier' shall ~~have a value that is different from the 'Event Object Identifier' in the event notification and for which no object exists in the IUT~~ *specify an object that does not support or is not configured for alarming, or which does not exist.*

Notes to Tester: A passing result is the same message sequence described in 9.1.2.1 except that the ~~Error Class in step 7 shall be OBJECT and the Error Code in step 7 shall be UNKNOWN_OBJECT~~*Error Class and Error Code in step 9 shall be OBJECT and UNKNOWN_OBJECT if the object referenced by 'Event Object Identifier' does not exist or OBJECT and NO_ALARM_CONFIGURED if the object exists but does not support or is not configured for alarming.* ~~For devices that claim a Protocol_Revision of 5 or prior, an Error Class of SERVICES with an Error Code of INCONSISTENT_PARAMETERS shall also be accepted.~~*For devices claiming a Protocol Revision less than 5, an Error Class and Error Code of SERVICES and* INCONSISTENT_PARAMETERS *or Error Class of OBJECT and Error Code of OTHER shall also be allowed. If the IUT can only be configured with one recipient in the Recipient_List property of the issuing Notification_class object, omit steps 5, 6, 15, and 16.*

[Change **Clause 9.1.2.4**, p. 265]
[Reviewer Note: This test was updated to account for revision 5 specifications. There is no new SSPC proposal.]

9.1.2.4 Unsuccessful Alarm Acknowledgment of Confirmed Event Notifications Because the 'Event State Acknowledged' is Invalid

Purpose: To verify that an alarm remains unacknowledged if the 'Event State Acknowledged' is inconsistent with the ~~other parameters~~ *Event_State* that ~~define~~ *was provided in the notification which is*~~the alarm~~ being acknowledged.

Test Concept: An alarm is triggered that causes the IUT to notify the TD and ~~at least~~ one other device. The TD acknowledges the alarm using an invalid event state and verifies that the acknowledgment is not accepted by the IUT and that the IUT does not notify other devices that the alarm was acknowledged. The TD then acknowledges the alarm using the proper event state and verifies that the acknowledgment is properly noted by the IUT. The IUT notifies all other recipients that the alarm was acknowledged.

Configuration Requirements: The IUT shall be configured with at least one object that can detect alarm conditions and send confirmed notifications. The Acked_Transitions property shall have the value B'111' indicating that all transitions have been acknowledged. The TD and ~~at least~~ one other BACnet device, *if the IUT supports multiple recipients,* shall be recipients of the alarm notification.

Test Steps: The test steps defined in 9.1.2.1 shall be followed except that in the first AcknowledgeAlarm request the 'Time Stamp' shall have the same value as the 'Time Stamp' from the event notification*, the 'To State' in the notification shall be any offnormal transition* and the 'Event State Acknowledged' shall have an ~~off normal value other than OFFNORMAL and other than the value of the 'To State' parameter in the event notification.~~*offnormal value that is different from the 'To State' in the event notification and shall not be OFFNORMAL.*

Notes to Tester: A passing result is the same message sequence described *as the passing result* in 9.1.2.1 except that the Error Code ~~in step 7~~ *in step 9* shall be *INVALID_EVENT_STATE. For devices claiming a Protocol Revision less than 5, an Error Code of* INCONSISTENT_PARAMETERS *shall also be allowed. If the IUT can only be configured with one recipient in the Recipient_List property of the issuing Notification_class object, omit steps 5, 6, 15, and 16.*

[Change **Clause 9.1.2.5**, p. 265]
[Reviewer Note: Made changes to allow cases where only one Recipient_List entry is supported.]

9.1.2.5 Unsuccessful Alarm Acknowledgment of Unconfirmed Event Notifications Because the 'Time Stamp' is Too Old

Purpose: To verify that an alarm remains unacknowledged if the time stamp in the acknowledgment does not match the most recent transition to the current alarm state.

Test Concept: An alarm is triggered that causes the IUT to notify the TD and ~~at least~~ one other device. The TD acknowledges the alarm using an old time stamp and verifies that the acknowledgment is not accepted by the IUT and that the IUT does not notify other devices that the alarm was acknowledged. The TD then acknowledges the alarm using the proper time stamp and verifies that the acknowledgment is properly noted by the IUT. The IUT notifies all other recipients that the alarm was acknowledged.

Configuration Requirements: The IUT shall be configured with at least one object that can detect alarm conditions and send unconfirmed notifications. The Acked_Transitions property shall have the value B'111' indicating that all transitions have been acknowledged. The TD and ~~at least~~ one other BACnet device, *if the IUT supports multiple recipients,* shall be recipients of the alarm notification.

Test Steps:

1. MAKE (a change that triggers the detection of an alarm event in the IUT)
2. WAIT ~~Time_Delay~~*(pTimeDelay)*
3. BEFORE **Notification Fail Time**
   RECEIVE UnconfirmedEventNotification-Request,
   'Process Identifier' = (the process identifier configured for this event),
   'Initiating Device Identifier' = IUT,
   'Event Object Identifier' = (the object detecting the alarm),
   'Time Stamp' = (~~the current time or sequence number~~*any valid time stamp*),
   'Notification Class' = (the notification class configured for this event),
   'Priority' = (the priority configured for this event type),
   'Event Type' = (any valid event type),
   'Message Text' = *(optional, any valid message text),*
   'Notify Type' = (the notify type configured for the event),
   'AckRequired' = TRUE,
   'From State' = NORMAL,
   'To State' = (any appropriate non-normal event state),
   'Event Values' = (the values appropriate to the event type)
4. IF (the notification in step 3 was not a broadcast) THEN
   RECEIVE
   DESTINATION = (at least one device other than the TD),
   SOURCE = IUT,
   UnconfirmedEventNotification-Request,
   'Process Identifier' = (the process identifier configured for this event),
   'Initiating Device Identifier' = IUT,
   'Event Object Identifier' = (the object detecting the alarm),
   'Time Stamp' = (the timestamp or sequence number from step 3),
   'Notification Class' = (the notification class configured for this event),
   'Priority' = (the priority configured for this event type),
   'Event Type' = (any valid event type),
   'Message Text' = *(optional, any valid message text),*
   'Notify Type' = (the notify type configured for the event),
   'AckRequired' = TRUE,
   'From State' = NORMAL,
   'To State' = (any appropriate non-normal event state),
   'Event Values' = (the values appropriate to the event type)
5. VERIFY (the 'Event Object Identifier' from the event notification), Acked_Transitions = (FALSE,TRUE,TRUE)
6. TRANSMIT AcknowledgeAlarm-Request,
   'Acknowledging Process Identifier' = (the value of the 'Process Identifier' parameter in the event notification),
   'Event Object Identifier' = (the 'Event Object Identifier' from the event notification),
   'Event State Acknowledged' = (the state specified in the 'To State' parameter of the notification),
   'Time Stamp' = (a time stamp older than the one conveyed in the notification),
   '*Acknowledgement Source*' = *(any valid value),*

'Time of Acknowledgment' = (the TD's current time using a Time format)

7. RECEIVE BACnet-Error-PDU
   Error Class = SERVICES,
   Error Code = INVALID_TIME_STAMP
8. VERIFY (the 'Event Object Identifier' from the event notification), Acked_Transitions = (FALSE,TRUE,TRUE)
9. TRANSMIT AcknowledgeAlarm-Request,
   'Acknowledging Process Identifier' = (the process identifier configured for this event),
   'Event Object Identifier' = (the 'Event Object Identifier' from the event notification),
   'Event State Acknowledged' = (the state specified in the 'To State' parameter of the notification),
   'Time Stamp' = (the time stamp conveyed in the notification),
   '*Acknowledgement Source*' = *(any valid value),*
   'Time of Acknowledgment' = (the TD's current time using a Time format)
10. RECEIVE BACnet-~~Simple-ACK~~*SimpleACK*-PDU
11. IF (Protocol_Revision is present and Protocol_Revision ≥ 1) THEN
    BEFORE **Notification Fail Time**
    RECEIVE
    DESTINATION = LOCAL BROADCAST | GLOBAL BROADCAST | TD,
    SOURCE = IUT,
    UnconfirmedEventNotification-Request,
    'Process Identifier' = (the process identifier configured for this event),
    'Initiating Device Identifier' = IUT,
    'Event Object Identifier' = (the object detecting the alarm),
    'Time Stamp' = (~~the current time or sequence number~~*any valid time stamp*),
    'Notification Class' = (the notification class configured for this event),
    'Priority' = (the priority configured for this event type),
    'Event Type' = (any valid event type),
    'Message Text' = *(optional, any valid message text),*
    'Notify Type' = ACK_NOTIFICATION,
    'To State' = (the 'To State' used in step 3 or 4)
    ELSE
    BEFORE **Notification Fail Time**
    RECEIVE
    DESTINATION = LOCAL BROADCAST | GLOBAL BROADCAST | TD,
    SOURCE = IUT,
    UnconfirmedEventNotification-Request,
    'Process Identifier' = (the process identifier configured for this event),
    'Initiating Device Identifier' = IUT,
    'Event Object Identifier' = (the object detecting the alarm),
    'Time Stamp' = (~~the current time or sequence number~~*any valid time stamp*),
    'Notification Class' = (the notification class configured for this event),
    'Priority' = (the priority configured for this event type),
    'Event Type' = (any valid event type),
    'Message Text' = *(optional, any valid message text),*
    'Notify Type' = ACK_NOTIFICATION
12. IF (the notification in step 11 was not broadcast) THEN
    IF (Protocol_Revision is present and Protocol_Revision ≥ 1) THEN
    RECEIVE
    DESTINATION = (at least one device other than the TD),
    SOURCE = IUT,
    UnconfirmedEventNotification-Request,
    'Process Identifier' = (the process identifier configured for this event),
    'Initiating Device Identifier' = IUT,
    'Event Object Identifier' = (the object detecting the alarm),
    'Time Stamp' = (the timestamp or sequence number from the notification in step 11),
    'Notification Class' = (the notification class configured for this event),
    'Priority' = (the priority configured for this event type),

'Event Type' = (any valid event type),
'Message Text' = *(optional, any valid message text),*
'Notify Type' = ACK_NOTIFICATION,
'To State' = (the 'To State' used in step 3 or 4)

ELSE
RECEIVE
DESTINATION = (at least one device other than the TD),
SOURCE = IUT,
UnconfirmedEventNotification-Request,
'Process Identifier' = (the process identifier configured for this event),
'Initiating Device Identifier' = IUT,
'Event Object Identifier' = (the object detecting the alarm),
'Time Stamp' = (the timestamp or sequence number from the notification in step 11),
'Notification Class' = (the notification class configured for this event),
'Priority' = (the priority configured for this event type),
'Event Type' = (any valid event type),
'Message Text' = *(optional, any valid message text),*
'Notify Type' = ACK_NOTIFICATION

13. VERIFY (the 'Event Object Identifier' from the event notification), Acked_Transitions = (TRUE,TRUE,TRUE)

Notes to Tester: The destination address used for the acknowledgment notification in step 11 shall be the same address used in step 3. The destination address used for the acknowledgment notification in step 12 shall be the same address used in step 4. When multiple event notifications are expected for a specific event, the order that the IUT transmits them in is irrelevant. *If the IUT can only be configured with one recipient in the Recipient_List property of the issuing Notification_class object, omit steps 4 and 12.*

[Change **Clause 9.1.2.6**, p. 267]
[Reviewer Note: Made changes to allow cases where only one Recipient_List entry is supported.]

9.1.2.6 Unsuccessful Alarm Acknowledgment of Unconfirmed Event Notifications Because the ~~Referenced Object Does Not Exist~~*'Event Object Identifier' is Invalid*

Purpose: To verify that an alarm remains unacknowledged if the 'Event Object Identifier' represents an object that does not exist *or is not consistent with the other parameters that define the alarm being acknowledged*.

Test Concept: An alarm is triggered that causes the IUT to notify the TD and ~~at least~~ one other device. The TD acknowledges the alarm using an ~~invalid~~*improper* event object identifier and verifies that the acknowledgment is not accepted by the IUT and that the IUT does not notify other devices that the alarm was acknowledged. The TD then acknowledges the alarm using the proper event object identifier and verifies that the acknowledgment is properly noted by the IUT. The IUT notifies all other recipients that the alarm was acknowledged.

Configuration Requirements: The IUT shall be configured with at least one object that can detect alarm conditions and send unconfirmed notifications. The Acked_Transitions property shall have the value B'111' indicating that all transitions have been acknowledged. The TD and ~~at least~~ one other BACnet device, *if the IUT supports multiple recipients,* shall be recipients of the alarm notification.

Test Steps: The test steps defined in 9.1.2.5 shall be followed except that in the first AcknowledgeAlarm request the ~~-~~'Time Stamp' shall have the same value as the 'Time Stamp' from the event notification and the 'Event Object Identifier' ~~have a value that is different from the 'Event Object Identifier' in the event notification and for which no object exists in the IUT~~ *specify an object that does not support or is not configured for alarming, or which does not exist.*

Notes to Tester: A passing result is the same message sequence described as the passing result in 9.1.2.5 except that the ~~Error Class in step 7 shall be OBJECT and the Error Code in step 7 shall be UNKNOWN_OBJECT~~*Error Class and Error Code in step 7 shall be OBJECT and UNKNOWN_OBJECT if the object referenced by 'Event Object Identifier' does not exist or OBJECT and NO_ALARM_CONFIGURED if the object exists but does not support or is not configured for alarming.* ~~For~~

~~devices that claim a Protocol_Revision of 5 or prior, an Error Class of SERVICES with an Error Code of INCONSISTENT_PARAMETERS shall also be accepted.~~*For devices claiming a Protocol Revision less than 5, an Error Class and Error Code of SERVICES and* INCONSISTENT_PARAMETERS *or Error Class of OBJECT and Error Code of OTHER shall also be allowed. If the IUT can only be configured with one recipient in the Recipient_List property of the issuing Notification_class object, omit steps 4 and 12.*

[Change **Clause 9.1.2.7**, p. 268]
[Reviewer Note: This test was updated to account for revision 5 specifications. There is no new SSPC proposal. Made changes to allow cases where only one Recipient_List entry is supported.]

9.1.2.7 Unsuccessful Alarm Acknowledgment of Unconfirmed Event Notifications Because the 'Event State Acknowledged' is Invalid

Purpose: To verify that an alarm remains unacknowledged if the 'Event State Acknowledged' is inconsistent with the ~~other parameters~~*Event_State* that ~~define~~ *was provided in the notification which is*~~the alarm~~ being acknowledged.

Test Concept: An alarm is triggered that causes the IUT to notify the TD and ~~at least~~ one other device. The TD acknowledges the alarm using an invalid 'Event State Acknowledged' and verifies that the acknowledgment is not accepted by the IUT and that the IUT does not notify other devices that the alarm was acknowledged. The TD then acknowledges the alarm using the proper 'Event State Acknowledged' and verifies that the acknowledgment is properly noted by the IUT. The IUT notifies all other recipients that the alarm was acknowledged.

Configuration Requirements: The IUT shall be configured with at least one object that can detect alarm conditions and send unconfirmed notifications. The Acked_Transitions property shall have the value B'111' indicating that all transitions have been acknowledged. The TD and ~~at least~~ one other BACnet device, *if the IUT supports multiple recipients,* shall be recipients of the alarm notification.

Test Steps: The test steps defined in ~~9.1.2.1~~*9.1.2.5* shall be followed except that in the first AcknowledgeAlarm request the 'Time Stamp' shall have the same value as the 'Time Stamp' from the event notification*, the 'To State' in the notification shall be any offnormal transition* and the 'Event State Acknowledged' shall have an ~~off-normal value other than OFFNORMAL and other than the value of the 'To State' parameter in the event notification.~~*offnormal value that is different from the 'To State' in the event notification and shall not be OFFNORMAL.*

Notes to Tester: A passing result is the same message sequence described as the passing result in 9.1.2.5 except that the Error Code in step 7 shall be *INVALID_EVENT_STATE. For devices claiming a Protocol Revision less than 5, an Error Code of* INCONSISTENT_PARAMETERS *shall also be allowed. If the IUT can only be configured with one recipient in the Recipient_List property of the issuing Notification_class object, omit steps 4 and 12.*

[Change **Clause 9.4**, p. 275]

9.4 ConfirmedEventNotification Service Execution Tests

~~Dependencies: None.~~

~~BACnet Reference Clause: 13.8.~~

Purpose: To verify that the IUT can execute the ConfirmedEventNotification service request.

Test Concept: BACnet does not define any action to be taken upon receipt of a ConfirmedEventNotification except to return an acknowledgement. Although returning an acknowledgment is sufficient to conform to the standard, a vendor may specify additional actions that can be observed. Any vendor-defined actions that do not occur during the tests shall be noted in the test report. No negative tests are included.

9.4.1 ConfirmedEventNotification Using the Time Form of the 'Timestamp' Parameter and Conveying a ~~Text~~ Message *Text*

Test Steps:

1. TRANSMIT ConfirmedEventNotification-Request,
   'Process Identifier' = (any valid process identifier),
   'Initiating Device Identifier' = TD,
   'Event Object Identifier' = (any Event Enrollment object),
   'Time Stamp' = (current time using the Time format),
   'Notification Class' = (any valid notification class),
   'Priority' = (any valid priority),
   'Event Type' = (any standard event type),
   'Message Text' = (any character string),
   'Notify Type' = ALARM | EVENT,
   'AckRequired' = FALSE,
   'From State' = NORMAL,
   'To State' = (any non-normal state appropriate to the event type),
   'Event Values' = (any values appropriate to the event type)
2. RECEIVE BACnet-SimpleACK-PDU
3. CHECK (for any vendor-defined observable actions)

9.4.2 ConfirmedEventNotification Using the DateTime Form of the 'Timestamp' Parameter and no ~~Text~~ Message *Text*

Test Steps:

1. TRANSMIT ConfirmedEventNotification-Request,
   'Process Identifier' = (any valid process identifier),
   'Initiating Device Identifier' = TD,
   'Event Object Identifier' = (any Event Enrollment object),
   'Time Stamp' = (current time using the DateTime format),
   'Notification Class' = (any valid notification class),
   'Priority' = (any valid priority),
   'Event Type' = (any standard event type),
   'Notify Type' = ALARM | EVENT,
   'AckRequired' = FALSE,
   'From State' = NORMAL,
   'To State' = (any non-normal state appropriate to the event type),
   'Event Values' = (any values appropriate to the event type)
2. RECEIVE BACnet-SimpleACK-PDU
3. CHECK (for any vendor-defined observable actions)

9.4.3 ConfirmedEventNotification Using the Sequence Number Form of the 'Timestamp' Parameter and no ~~Text~~ Message *Text*

Test Steps:

1. TRANSMIT ConfirmedEventNotification-Request,
   'Process Identifier' = (any valid process identifier),
   'Initiating Device Identifier' = TD,
   'Event Object Identifier' = (any Event Enrollment object),
   'Time Stamp' = (current sequence number),
   'Notification Class' = (any valid notification class),
   'Priority' = (any valid priority),
   'Event Type' = (any standard event type),
   'Notify Type' = ALARM | EVENT,
   'AckRequired' = FALSE,
   'From State' = NORMAL,
   'To State' = (any non-normal state appropriate to the event type),
   'Event Values' = (any values appropriate to the event type)
2. RECEIVE BACnet-SimpleACK-PDU
3. CHECK (for any vendor-defined observable actions)

9.4.4 . . .

[Reviewer Note: Addendum 135-2010af added language to ensure that notifications are not ignored due to unsupported character sets.]

9.4.5 ConfirmedEventNotification Simple Presentation

Purpose: This test case verifies that the IUT is capable of minimally displaying ConfirmedEventNotifications.

Configuration *Requirements*: For this test, the tester shall choose one event-generating object, O1.

Test Steps:

1. TRANSMIT ConfirmedEventNotification-Request,
   'Process Identifier' = (a valid process identifier specified by the IUT vendor),
   'Initiating Device Identifier' = TD,
   'Event Object Identifier' = O1,
   'Time Stamp' = (~~current time in any format~~*any valid time stamp*),
   'Notification Class' = (any valid notification class),
   'Priority' = (any valid priority),
   'Event Type' = (any standard event type),
   'Message Text' = (any character string),
   'Notify Type' = ALARM | EVENT,
   'AckRequired' = TRUE | FALSE,
   'From State' = (state S1, any valid state for this event type),
   'To State' = (state S2, any valid state for this event type that can follow S1),
   'Event Values' = (any values appropriate to the event type)
2. RECEIVE BACnet-SimpleACK-PDU
3. CHECK (that the IUT indicates the notification to the operator and that the indication includes identification of the event generating object or the monitored object, the event timestamp, and the event Message Text)
4. CHECK (that all information indicated to the user is consistent with the information provided in step 1)

~~Passing Result~~*Notes to Tester*: The IUT shall truncate the message text if it is longer than the maximum length displayable by the IUT. The IUT is allowed to include characters in the displayed text that indicate the message has been truncated, even if the truncated message is then shorter than 32 characters. The IUT shall not truncate Message Text that is less than or equal to 32 characters in length. *A device shall not fail to process an EventNotification service request containing a 'Message Text'*

parameter in an unsupported character set. It is a local matter whether the parameter is used as provided or whether a character string, in a supported character set, of length 0 is used in its place.

[Reviewer Note: Addendum 135-2010af added language to ensure that notifications are not ignored due to unsupported character sets.]

9.4.6 ConfirmedEventNotification Full Presentation

Purpose: This test case verifies that the IUT is capable of displaying ConfirmedEventNotifications.

Configuration *Requirements*: For this test, the tester shall choose one event generating object, O1.

Test Steps:

1. TRANSMIT ConfirmedEventNotification-Request,
   'Process Identifier' = (a valid process identifier specified by the IUT vendor),
   'Initiating Device Identifier' = TD,
   'Event Object Identifier' = O1,
   'Time Stamp' = (~~current time in any format~~*any valid time stamp*),
   'Notification Class' = (any valid notification class),
   'Priority' = (any valid priority),
   'Event Type' = (any standard event type),
   'Message Text' = (any character string),
   'Notify Type' = ALARM | EVENT,
   'AckRequired' = TRUE | FALSE,
   'From State' = (state S1, any valid state for this event type),
   'To State' = (state S2, any valid state for this event type that can follow ~~S2~~*S1*),
   'Event Values' = (any values appropriate to the event type)
2. RECEIVE BACnet-SimpleACK-PDU
3. CHECK (that the IUT indicates the notification to the operator and that the indication includes identification of the event generating object or the monitored object, the event timestamp, the event Message Text, Notification Class, Priority, Notify Type, Ack Required, To State and Event Values)
4. CHECK (that all information indicated to the user is consistent with the information provided in step 1)

~~Passing Result~~*Notes to Tester*: The IUT shall truncate the message text if it is longer than the maximum length displayable by the IUT. The IUT is allowed to include characters in the displayed text that indicate the message has been truncated, even if the truncated message is then shorter than 255 characters. The IUT shall not truncate Message Text that is less than or equal to 255 characters in length. *A device shall not fail to process an EventNotification service request containing a 'Message Text' parameter in an unsupported character set. It is a local matter whether the parameter is used as provided or whether a character string, in a supported character set, of length 0 is used in its place.*

[Add new **Clause 9.4.X1**, p. 271]
[Reviewer Note: Addendum 135-2010af added language to ensure that notifications are not ignored due to unsupported character sets.]

9.4.X1 Unsupported Message Text Character Set ConfirmedEventNotification Test

Purpose: To verify that the IUT correctly receives and processes ConfirmedEventNotifications when the Message Text parameter is of a character set that the IUT does not support.

Test Concept: Send a notification to the IUT, from an event-initiating object, O1, which contains a Message Text parameter value, T1, which uses a character set that the IUT does not support. Verify that the IUT processes the request even if the 'Message Text' uses a character set that the IUT does not support, and that the IUT returns a Result(+) and performs the vendor specified actions.

Configuration Requirements: Configure the TD as though it has an event-initiating object, O1, which references a Notification Class object N1. Configure N1 to direct notifications to the IUT using a vendor specified Process Id, PID1. If the IUT supports all character sets, this test shall be skipped.

Test Steps:

1. TRANSMIT ConfirmedEventNotification-Request,
   'Process Identifier' = PID1,
   'Initiating Device Identifier' = TD,
   'Event Object Identifier' = O1,
   'Time Stamp' = (any valid timestamp),
   Notification Class' = (N1: the Notification_Class configured in O1),
   'Priority' = (any valid priority),
   'Event Type' = (the standard event type associated with O1),
   'Message Text' = T1,
   'Notify Type' = ALARM | EVENT,
   'AckRequired' = FALSE,
   'From State' = (any valid event state),
   'To State' = (any valid event state),
   'Event Values' = (any values appropriate to the event type)
2. RECEIVE BACnet-SimpleACK-PDU
3. CHECK (for any vendor-defined observable actions)

[Add new **Clause 9.5.3**, p. 277]
[Reviewer Note: Addendum 135-2010af added language to ensure that notifications are not ignored due to unsupported character sets.]

9.5.3 Unsupported Message Text Character Set UnconfirmedEventNotification Test

Purpose: To verify that the IUT correctly receives and processes UnconfirmedEventNotifications when the Message Text parameter is of a character set that the IUT does not support.

Test Concept: Send a notification to the IUT, from an event-initiating object, O1, which contains a Message Text parameter value, T1, which uses a character set that the IUT does not support. Verify that the IUT processes the request even if the 'Message Text' uses a character set that the IUT does not support, and that the IUT performs the vendor specified actions.

Configuration Requirements: Configure TD to direct notifications to the IUT using a vendor specified Process Identifier, PID1. If the IUT supports all character sets, this test shall be skipped.

Test Steps: The test steps for this test case are identical to the test steps in 9.4.X1 except that the UnconfirmedEventNotification requests are used instead of ConfirmedEventNotification requests and the IUT does not acknowledge receiving the notifications.

[Change **Clause 9.6**, p. 277]

9.6 GetAlarmSummary Service Execution Tests

This clause defines the tests necessary to demonstrate support for executing GetAlarmSummary service requests.

~~BACnet Reference Clause: 13.10.~~

~~Dependencies: None.~~

[Change **Clause 9.7.1.1**, p. 278]
[Reviewer Note: It is not important what filter parameter or parameter is used to engender the return of a summary with zero summaries.]

9.7.1.1 Enrollment Summary with Zero Summaries

Purpose: To verify that the IUT can execute the GetEnrollmentSummary request when there are no enrollments to report.

Configuration Requirements: The IUT shall be configured with no enrollments to report.

Test Steps:

1. TRANSMIT GetEnrollmentSummary-Request,
   'Acknowledgment Filter' = ~~ALL~~ *NOT_ACKED*
2. RECEIVE GetEnrollmentSummary-ACK,
   'List of Enrollment Summaries' = (an empty list)

Notes to Tester: If the IUT cannot be configured with no enrollments to report, then the GetEnrollmentSummary-Request shall be transmitted with a further constrained argument so that the resulting filtered enrollment summary yields zero summaries.

[Change **Clause 9.7.2.2**, p. 280]

9.7.2.2 Event State Filter

Purpose: To verify that the IUT can execute the GetEnrollmentSummary request when the 'Event State Filter' is used.

Configuration Requirements: If possible, the IUT shall be configured so that it has one or more event-generating objects that have an Event_State property value of NORMAL, one or more with an Event_State property value of FAULT, one or more with an Event_State property value of OFFNORMAL, and one or more with an Event_State property value that is not NORMAL, OFFNORMAL, or FAULT. If only a subset of these cases can be supported as many of them as possible shall be configured.

Test Steps:

1. TRANSMIT GetEnrollmentSummary-Request,
   'Acknowledgment Filter' = ALL,
   'Event State Filter' = NORMAL
2. RECEIVE GetEnrollmentSummary-ACK,
   'List of Enrollment Summaries' = (all configured event-generating objects with ~~Event_State~~*pCurrentState* = NORMAL)
3. TRANSMIT GetEnrollmentSummary-Request,
   'Acknowledgment Filter' = ALL,
   'Event State Filter' = FAULT
4. RECEIVE GetEnrollmentSummary-ACK,
   'List of Enrollment Summaries' = (all configured event-generating objects with ~~Event_State~~*pCurrentState* = FAULT)
5. TRANSMIT GetEnrollmentSummary-Request,
   'Acknowledgment Filter' = ALL,
   'Event State Filter' = OFFNORMAL
6. RECEIVE GetEnrollmentSummary-ACK,
   'List of Enrollment Summaries' = (all configured event-generating objects with ~~Event_State~~*pCurrentState* = OFFNORMAL)
7. TRANSMIT GetEnrollmentSummary-Request,
   'Acknowledgment Filter' = ALL,

'Event State Filter' = ACTIVE

8. RECEIVE GetEnrollmentSummary-ACK,
   'List of Enrollment Summaries' = (all configured event-generating objects with ~~Event_State~~*pCurrentState* = a value other than NORMAL)
9. TRANSMIT GetEnrollmentSummary-Request,
   'Acknowledgment Filter' = ALL,
   'Event State Filter' = ALL
10. RECEIVE GetEnrollmentSummary-ACK,
    'List of Enrollment Summaries' = (the union of all of the summaries returned in steps 1 - 8)

[Change **Clause 9.7.2.3**, p. 280]
[Reviewer Note: Revise test for new Event Types.]

9.7.2.3 Event Type Filter

Purpose: To verify that the IUT can execute the GetEnrollmentSummary request when the 'Event Type Filter' is used.

~~Configuration Requirements: If possible, the IUT shall be configured so that it has one or more event-generating objects for each of the event types CHANGE_OF_BITSTRING, CHANGE_OF_STATE, CHANGE_OF_VALUE, COMMAND_FAILURE, FLOATING_LIMIT, and OUT_OF_RANGE. If only a subset of these event types are supported as many of them as possible shall be configured.~~

~~Test Steps:~~

~~1. TRANSMIT GetEnrollmentSummary-Request,~~
~~'Acknowledgment Filter' = ALL,~~
~~'Event Type Filter' = CHANGE_OF_BITSTRING~~
~~2. RECEIVE GetEnrollmentSummary-ACK,~~
~~'List of Enrollment Summaries' = (all configured event-generating objects with Event_Type = CHANGE_OF_BITSTRING)~~
~~3. TRANSMIT GetEnrollmentSummary-Request,~~
~~'Acknowledgment Filter' = ALL,~~
~~'Event Type Filter' = CHANGE_OF_STATE~~
~~4. RECEIVE GetEnrollmentSummary-ACK,~~
~~'List of Enrollment Summaries' = (all configured event-generating objects with Event_Type = CHANGE_OF_STATE)~~
~~5. TRANSMIT GetEnrollmentSummary-Request,~~
~~'Acknowledgment Filter' = ALL,~~
~~'Event Type Filter' = CHANGE_OF_VALUE~~
~~6. RECEIVE GetEnrollmentSummary-ACK,~~
~~'List of Enrollment Summaries' = (all configured event-generating objects with Event_Type = CHANGE_OF_VALUE)~~
~~7. TRANSMIT GetEnrollmentSummary-Request,~~
~~'Acknowledgment Filter' = ALL,~~
~~'Event Type Filter' = FLOATING_LIMIT~~
~~8. RECEIVE GetEnrollmentSummary-ACK,~~
~~'List of Enrollment Summaries' = (all configured event-generating objects with Event_Type = FLOATING_LIMIT)~~

Configuration Requirements: If possible, the IUT shall be configured so that it has one or more event-generating objects for each of its supported event types. If the IUT cannot be configured in such a way all at once, then the test shall be repeated so that each of its supported event types is tested. If only a subset of these event types are supported, as many of them as possible shall be configured.

Test Steps:

REPEAT Y = (All the configurations that will be tested) DO {
REPEAT X = (All the Event Types currently configured) DO {
TRANSMIT GetEnrollmentSummary-Request,
'Acknowledgment Filter' = ALL,
'Event Type Filter' = X
RECEIVE GetEnrollmentSummary-ACK,
'List of Enrollment Summaries' = (all configured event-generating objects with Event_Type = X)
}

[Change **Clause 9.8.6**, p. 283]
[Reviewer Note: Corrects the 'max=APDU-length-accepted' value to represent 128 bytes instead of 50 bytes.]

9.8.6 Chaining Test

Purpose: This test case exercises the chaining capabilities using multiple GetEventInformation messages.

Configuration Requirements: The IUT shall be configured so that there are more event states than can be conveyed in a single APDU of 128 bytes. The IUT shall be configured to contain enough events to trigger the chaining effect. If the IUT can not be configured to contain enough active events to trigger chaining, this test may be skipped.

Test Concept: In steps 1-4, the test first tests proper chaining by requesting two lists from the IUT and verifying that the second list is properly distinct from the first. In steps 5-9, to test the “fixed object processing order” as defined in BACnet 13.12.1.1.1, it requests the first list again, and then, before requesting the second list, the tester makes the last object in the first list no longer have any active event states. When the TD requests the second list using the object identifier of the now-normal device, the IUT should respond with the same second list as it did before.

Test Steps:

1. TRANSMIT GetEventInformation-Request,
   'max-APDU-length-accepted' = ~~B'0000'~~*B'0001'*,
   'segmented-response-accepted' = FALSE
2. RECEIVE GetEventInformation-ACK,
   'List of Event Summaries' = (an arbitrary list),
   'More Events' = TRUE
3. TRANSMIT GetEventInformation-Request,
   'Last Received Object Identifier' = the last object identifier of the list received in step 2)
4. RECEIVE GetEventInformation-ACK,
   'List of Event Summaries' = (a list of object identifiers not including any received in step 2)
5. TRANSMIT GetEventInformation-Request,
   'max-APDU-length-accepted' = ~~B'0000'~~*B'0001'*,
   'segmented-response-accepted' = FALSE
6. RECEIVE GetEventInformation-ACK,
   'List of Event Summaries' = (an arbitrary list),
   'More Events' = TRUE
7. MAKE (the object identified by the last object identifier in the list received in step 6 have no active event states)
8. TRANSMIT GetEventInformation-Request,
   'Last Received Object Identifier' = (the last object identifier of the list received in step 6)
9. RECEIVE GetEventInformation-ACK,
   'List of Event Summaries' = (the same list received in step 4)

[Add new **Clause 7.3.2.30**, p. 152]

7.3.2.30 Notification Forwarder Object Tests

This clause defines the tests necessary to demonstrate Notification Forwarder Object functionality.

7.3.2.30.1 Common values and configurations used in all Notification Forwarder object tests

All tests use the value names listed below for configuration and verification of functionality. Most of the tests begin with either base setup 1 configuration or base setup 2 configuration defined below. Notifications originate from the notification source object and are distributed by the IUT Notification Forwarder object to the destinations. A notification source object may be any BACnet event-initiating object.

7.3.2.30.1.1 Values used in all Notification Forwarder object tests

Values are split into two categories indicated by the first characters which indicate SRC (Source) or DEST (Destination). After the category are additional characters indicating the meaning of the parameter:

SRC (Source) category. Used by notification source and Notification Class objects sending messages directed to the IUT Notification Forwarder object(s); or in the case that Local_Forwarding_Only is TRUE for the IUT, then the objects within the IUT device that are sending messages to the IUT Notification Forwarder object(s).

SRC_NOTIF_DEV	(Device containing the notification source object)
SRC_NOTIF_OBJ	(Object used as the notification source object)
SRC_NOTIF_CLS	(Notification Class object used by the notification source object)
SRC_PROCESS_ID	(Process Identifier value used in the Recipient_List of the Notification Class object used by the notification source object)
SRC_CONF_NOTIF	(Issue Confirmed Notifications value used in the Recipient_List of the Notification Class object used by the notification source object)
SRC_NOTIF_TYP	(Notify_Type value used by the source notification object)

DEST (Destination) category. Used by the IUT Notification Forwarder object(s) when forwarding messages.

DEST_OBJ_ID	(Recipient used in the Recipient_List or Subscribed_Recipients list of the IUT Notification Forwarder object under test)
DEST_PROCESS_ID	(Process_Identifier value used in the Recipient_List or Subscribed_Recipients list of the IUT Notification Forwarder object under test)
DEST_CONF_NOTIF	(Issue Confirmed Notifications value used in the Recipient_List or Subscribed_Recipients list of the IUT Notification Forwarder object under test)

Thus SRC_CONF_NOTIF indicates the (source) confirmed notifications selection for event notifications directed to the IUT Notification Forwarder object(s) while DEST_CONF_NOTIF indicates the (destination) confirmed notification selection for event notifications forwarded by the IUT Notification Forwarder object(s) to destination devices.

Other commonly used values include:

TR	Time Remaining
SR	Subscribed Recipients

Values indicated by these name codes can be selected by the tester within the requirements of the specification, and the test but must be a single unchanging value for the duration of each test. Values may be changed between tests.

Devices commonly specified include:

DS	Device notification Source, referenced by the value of SRC_NOTIF_DEV previously defined.
D1	Destination Device 1, referenced by the value of DEST_OBJ_ID previously defined.
D2, D3…DX	Other distinct destination devices that may be specified in a test.

When multiple destination devices are required for a test, those devices shall be distinct unless the test specifically indicates otherwise. Destination devices may be connected to be reachable through any combination of available ports on the IUT device appropriate for the test.

TD shall be connected to monitor the port or ports used by the IUT to transmit to the destination devices.

7.3.2.30.1.2 Base setup 1 for Notification Forwarder object tests

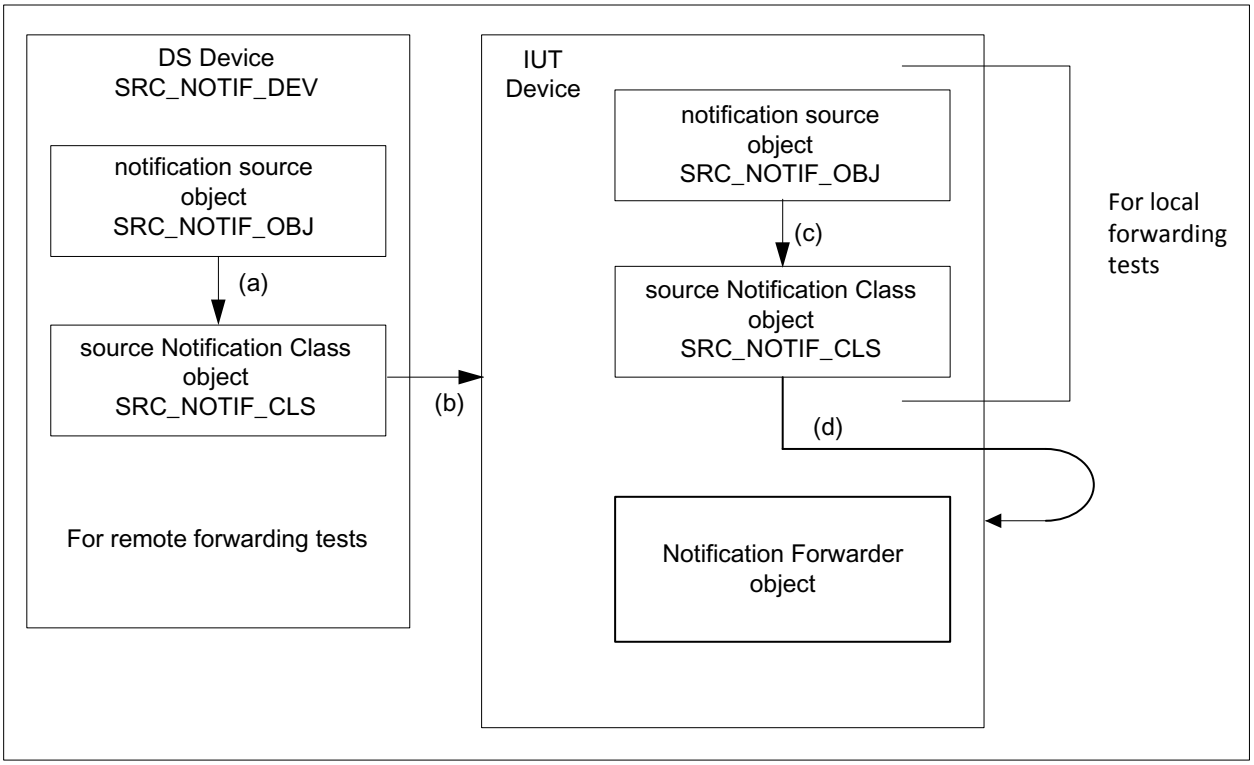

Figure X1. Logical relationship of objects required for Notification Forwarder functionality tests using base setup 1.

Base setup 1 as shown in Figure X1 requires that DS include two objects, a Notification source object (SRC_NOTIF_OBJ) and a Source notification class object (SRC_NOTIF_CLS). The DS Notification source object is an object capable of generating an event notification such as an analog value object with intrinsic alarming. The Notification source object's Notification_Class property references (a) the DS's Source notification class object which is a BACnet notification class object containing a BACnetDestination (b) referencing the IUT. The DS Notification source object is used to produce event notifications for distributing by the IUT Notification Forwarder Object unless the test indicates otherwise or unless Local_Forwarding_Only in the Notification Forwarder Object is TRUE and the test does not indicate otherwise.

Base setup 1 also requires that the IUT include three objects, a Notification source object (SRC_NOTIF_OBJ), a Source notification class object (SRC_NOTIF_CLS), and the Notification Forwarder Object being tested. The IUT Notification source object is an object capable of generating an event notification such as an analog value object with intrinsic alarming. The Notification source object's Notification_Class property references (c) the IUT's Source notification class object which is a BACnet notification class object containing a BACnetDestination (d) referencing the IUT. The IUT's Notification source object is used to produce event notifications for distributing by the IUT Notification Forwarder Object when Local_Forwarding_Only is TRUE and the test does not indicate otherwise.

Configure a Notification Forwarder object in the IUT as follows:

Out_Of_Service	= FALSE.
Recipient_List	= empty
Subscribed_Recipients list	= empty

Process_Identifier_Filter = NULL.
Enable all Port_Filter array entries (if present).

If the IUT has no Notification Forwarder object with a writable, NULL or zero Process_Identifier_Filter property, then select a Process Identifier value SRC_PROCESS_ID in the source notification class below to match the Process_Identifier_Filter value.

Configure a Notification source object in DS (or in the IUT when Local_Forwarding_Only is TRUE) as follows:

Source notification device	= SRC_NOTIF_DEV,	-- DS (or IUT if Local_Forwarding_Only is TRUE)
Source notification object	= SRC_NOTIF_OBJ,	
Notification_Class	= SRC_NOTIF_CLS	-- (a) or (c)
Event_Enable	= {T, T, T}	

Configure a Source notification class SRC_NOTIF_CLS in DS (or in the IUT when Local_Forwarding_Only is TRUE) as follows:

Notification Class	= instance number of notification class object,	
Priority	= (any valid priority)	
Ack_Required	= (any valid Ack_Required value)	
Recipient_List	= { (all),	-- Valid Days
	(all),	-- From Time, To Time
	(IUT as a recipient (b) or (d) compatible with the value of Local_Forwarding_Only in the Notification Forwarder Object under test)	-- Recipient
	}	-- One list element
Process Identifier	= SRC_PROCESS_ID	
Issue Confirmed Notifications	= SRC_CONF_NOTIF	
Transitions	= {T, T, T}	

SRC_CONF_NOTIF is assumed to be FALSE unless specified within the test. Thus UnconfirmedEventNotification-Request messages from the Notification source object are expected unless specified otherwise. Behaviors can alternately be tested using SRC_CONF_NOTIF set to TRUE resulting in ConfirmedEventNotification service messages from the Notification source object that must be acknowledged at the communication level, but it is not necessary to test both settings unless specifically directed otherwise by the test.

Local_Forwarding_Only is assumed to be FALSE unless specified within the test. Thus event notifications are assumed to originate from DS and are directed to the IUT unless specified within the test. Behaviors can alternately be tested with Local_Forwarding_Only set TRUE if no value is specified in the test and the IUT is capable of containing this value. In this case, event notifications shall originate from a Notification source object within the IUT device and are directed to the IUT object, and it is necessary for the tester to MAKE the IUT Notification source object generate an UnconfirmedEventNotification-Request with the specified parameters rather than TRANSMIT an UnconfirmedEventNotification-Request. It is not necessary to test both settings unless specifically directed by the test.

7.3.2.30.1.3 Base setup 2 for Notification Forwarder object tests

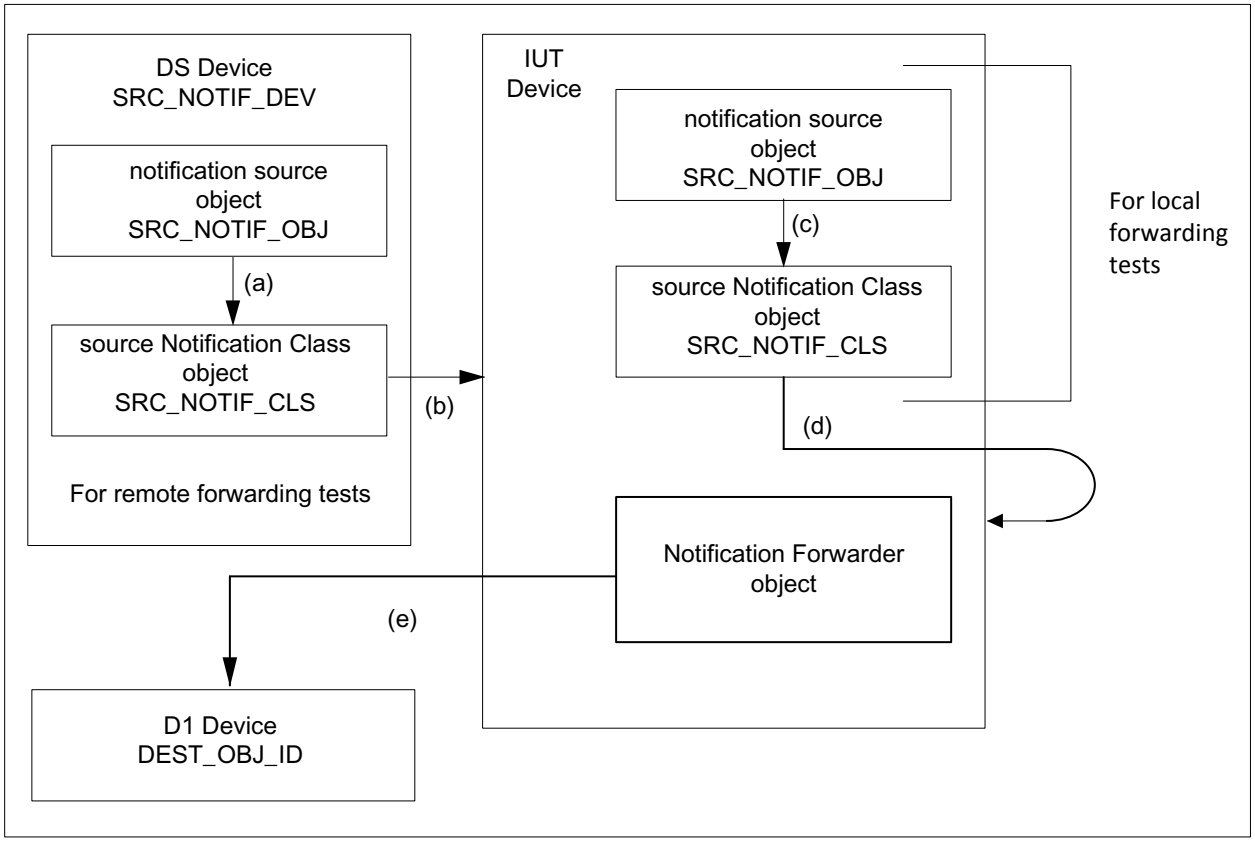

Figure X2. Logical relationship of objects required for Notification Forwarder functionality tests using base setup 2.

Base setup 2 as shown in Figure X2 is the same as base setup 1 except that the IUT's Notification Forwarder Object has a Subscribed_Recipient (e) added referencing D1 while base setup 1 has no defined recipients. Base setup 2 is used when a single subscribed recipient can be used to complete the test or is a logical starting point for the test.

This is achieved as follows:
Beginning with Base setup 1, add D1 as a subscribed recipient (e) to the IUT Notification Forwarder Object by performing the following additional steps:

1. TRANSMIT AddListElement-Request,
   'Object Identifier' = (the object being tested),
   'Property Identifier' = Subscribed_Recipients,
   List of Elements = { DEST_OBJ_ID, -- Recipient (e)
   DEST_PROCESS_ID, -- Any Process Identifier
   DEST_CONF_NOTIF, -- Issue Confirmed Notifications
   TR -- Time Remaining where TR > test duration
   } -- One list element
2. RECEIVE BACnet-SimpleACK-PDU

DEST_CONF_NOTIF is assumed to be FALSE unless specified within the test. Thus UnconfirmedEventNotification-Request messages from the Notification Forwarder object are expected unless specified otherwise. Behaviors can alternately be tested using DEST_CONF_NOTIF set to TRUE resulting in ConfirmedEventNotification service messages from the NotificationForwarder object that must be acknowledged at the communication level, but it is not necessary to test both settings unless specifically directed otherwise by the test.

7.3.2.30.2 Recipient_List Forwarding Test

Purpose: Verify that an event notification can be forwarded to a Recipient device in the Recipient_List. This test is used as the basis for several additional tests that require more specific value choices.

Test Concept: Write or configure a Recipient_List entry and then send an event notification to the IUT. Ensure that the event notification is forwarded to the recipient device with the correct parameters. Parameter values shall be selected for the test as a consistent set indicated in the test steps.

Configuration Requirements: Base setup 1 for Notification Forwarder object tests

Test Steps:

1. MAKE (Recipient_List = {(all), -- Valid Days
(all), -- From Time, To Time
DEST_OBJ_ID, -- Recipient D1
DEST_PROCESS_ID, -- Process Identifier
DEST_CONF_NOTIF, -- Issue Confirmed Notifications
{T, T, T} -- Transitions
}) -- One list element
2. IF (SRC_CONF_NOTIF is FALSE) THEN
TRANSMIT SOURCE = DS, UnconfirmedEventNotification-Request,
'Process Identifier' = SRC_PROCESS_ID,
'Initiating Device Identifier' = SRC_NOTIF_DEV,
'Event Object Identifier' = SRC_NOTIF_OBJ,
'Time Stamp' = (any valid time stamp),
'Notification Class' = SRC_NOTIF_CLS,
'Priority' = (any valid priority),
'Event Type' = (any valid event type),
'Message Text' = (optional, any valid message text),
'Notify Type' = SRC_NOTIF_TYP,
'AckRequired' = (any valid value), -- absent if Notify Type is ACK_NOTIFICATION
'From State' = (any valid From_State), -- absent if Notify Type is ACK_NOTIFICATION
'To State' = (any valid To_State),
'Event Values' = (any valid event values) -- absent if Notify Type is ACK_NOTIFICATION
ELSE
TRANSMIT SOURCE = DS, ConfirmedEventNotification-Request,
'Process Identifier' = SRC_PROCESS_ID,
'Initiating Device Identifier' = SRC_NOTIF_DEV,
'Event Object Identifier' = SRC_NOTIF_OBJ,
'Time Stamp' = (any valid time stamp),
'Notification Class' = SRC_NOTIF_CLS,
'Priority' = (any valid priority),
'Event Type' = (any valid event type),
'Message Text' = (optional, any valid message text),
'Notify Type' = SRC_NOTIF_TYP,
'AckRequired' = (any valid value), -- absent if Notify Type is ACK_NOTIFICATION
'From State' = (any valid From_State), -- absent if Notify Type is ACK_NOTIFICATION
'To State' = (any valid To_State),
'Event Values' = (any valid event values) -- absent if Notify Type is ACK_NOTIFICATION
RECEIVE DESTINATION = DS, BACnet-SimpleACK-PDU
3. BEFORE ***Notification Fail Time***
IF (DEST_CONF_NOTIF is FALSE) THEN
RECEIVE DESTINATION = D1, UnconfirmedEventNotification-Request,
'Process Identifier' = DEST_PROCESS_ID,
(other parameter values match the parameter values used in the previous step)
ELSE

RECEIVE DESTINATION = D1, ConfirmedEventNotification-Request
'Process Identifier' = DEST_PROCESS_ID,
(other parameter values match the parameter values used in the previous step)
TRANSMIT SOURCE = D1, BACnet-SimpleACK-PDU

7.3.2.30.3 Subscribed_Recipients Forwarding Test

Purpose: Verify that an event notification can be forwarded to a Recipient device in the Subscribed_Recipients. This test is used as the basis for several additional tests that require more specific value choices.

Test Concept: Add a Subscribed_Recipients list entry and then send an event notification to the IUT. Ensure that the event notification is forwarded to the recipient device with the correct parameters. Parameter values shall be selected for the test as a consistent set indicated in the test steps.

Configuration Requirements: Base setup 2 for Notification Forwarder object tests with TR lifetime sufficient for this test.

Test Steps:

1. IF (SRC_CONF_NOTIF is FALSE) THEN
   TRANSMIT SOURCE = DS, UnconfirmedEventNotification-Request,
   'Process Identifier' = SRC_PROCESS_ID,
   'Initiating Device Identifier' = SRC_NOTIF_DEV,
   'Event Object Identifier' = SRC_NOTIF_OBJ,
   'Time Stamp' = (any valid time stamp),
   'Notification Class' = SRC_NOTIF_CLS,
   'Priority' = (any valid priority),
   'Event Type' = (any valid event type),
   'Message Text' = (optional, any valid message text),
   'Notify Type' = SRC_NOTIF_TYP,
   'AckRequired' = (any valid value), -- absent if Notify Type is ACK_NOTIFICATION
   'From State' = (any valid From_State), -- absent if Notify Type is ACK_NOTIFICATION
   'To State' = (any valid To_State),
   'Event Values' = (any valid event values) -- absent if Notify Type is ACK_NOTIFICATION
   ELSE
   TRANSMIT SOURCE = DS, ConfirmedEventNotification-Request,
   'Process Identifier' = SRC_PROCESS_ID,
   'Initiating Device Identifier' = SRC_NOTIF_DEV,
   'Event Object Identifier' = SRC_NOTIF_OBJ,
   'Time Stamp' = (any valid time stamp),
   'Notification Class' = SRC_NOTIF_CLS,
   'Priority' = (any valid priority),
   'Event Type' = (any valid event type),
   'Message Text' = (optional, any valid message text),
   'Notify Type' = SRC_NOTIF_TYP,
   'AckRequired' = (any valid value), -- absent if Notify Type is ACK_NOTIFICATION
   'From State' = (any valid From_State), -- absent if Notify Type is ACK_NOTIFICATION
   'To State' = (any valid To_State),
   'Event Values' = (any valid event values) -- absent if Notify Type is ACK_NOTIFICATION
   RECEIVE DESTINATION = DS, BACnet-SimpleACK-PDU
2. BEFORE ***Notification Fail Time***
   IF (DEST_CONF_NOTIF is FALSE) THEN
   RECEIVE DESTINATION = D1, UnconfirmedEventNotification-Request
   'Process Identifier' = DEST_PROCESS_ID,
   (other parameter values match the parameter values used in the previous step)
   ELSE
   RECEIVE DESTINATION = D1, ConfirmedEventNotification-Request
   'Process Identifier' = DEST_PROCESS_ID,

(other parameter values match the parameter values used in the previous step)
TRANSMIT SOURCE = D1, BACnet-SimpleACK-PDU

7.3.2.30.4 Confirmed And Unconfirmed Forwarding Test

7.3.2.30.4.1 Confirmed Source And Confirmed Destination Forwarding Test

Purpose: Check that a confirmed source event notification is forwarded as a confirmed event notification when specified in the IUT Notification Forwarder recipient settings as confirmed.

Test Concept: Perform tests **7.3.2.30.2 Recipient_List Forwarding Test** and **7.3.2.30.3 Subscribed_Recipients Forwarding Test** for SRC_CONF_NOTIF set to TRUE (confirmed), and DEST_CONF_NOTIF set to TRUE (confirmed).

7.3.2.30.4.2 Confirmed Source And Unconfirmed Destination Forwarding Test

Purpose: Check that a confirmed source event notification is forwarded as an unconfirmed event notification when specified in the IUT Notification Forwarder recipient settings as unconfirmed.

Test Concept: Perform tests **7.3.2.30.2 Recipient_List Forwarding Test** and **7.3.2.30.3 Subscribed_Recipients Forwarding Test** for SRC_CONF_NOTIF set to TRUE (confirmed), and DEST_CONF_NOTIF set to FALSE (unconfirmed).

7.3.2.30.4.3 Unconfirmed Source And Confirmed Destination Forwarding Test

Purpose: Check that an unconfirmed source event notification is forwarded as a confirmed event notification when specified in the IUT Notification Forwarder recipient settings as confirmed.

Test Concept: Perform tests **7.3.2.30.2 Recipient_List Forwarding Test** and **7.3.2.30.3 Subscribed_Recipients Forwarding Test** for SRC_CONF_NOTIF set to FALSE (unconfirmed), and DEST_CONF_NOTIF set to TRUE (confirmed).

7.3.2.30.4.4 Unconfirmed Source And Unconfirmed Destination Forwarding Test

Purpose: Check that an unconfirmed source event notification is forwarded as an unconfirmed event notification when specified in the IUT Notification Forwarder recipient settings as unconfirmed.

Test Concept: Perform tests **7.3.2.30.2 Recipient_List Forwarding Test** and **7.3.2.30.3 Subscribed_Recipients Forwarding Test** for SRC_CONF_NOTIF set to FALSE (unconfirmed), and DEST_CONF_NOTIF set to FALSE (unconfirmed).

7.3.2.30.5 Character Encoding Test

Purpose: Verify that no event notifications are ignored due to the choice of character set in an event notification.

Test Concept: Send event notifications using character set encoding X to a Notification Forwarder. Character set encoding X should be chosen to be different than the character set encoding selected or used by the IUT device for at least one execution of this test. One or more Message Text characters shall be chosen as multi-byte characters. In the test steps, the event notification is sent twice, once testing the Notification Forwarder Subscribed_Recipients property and once testing the Notification Forwarder Recipient_List property. For each event notification sent, the forwarded event notification is examined. The test shall be performed with Local_Forwarding_Only set to FALSE. If the IUT is not capable of having Local_Forwarding_Only set to FALSE, then this test shall be omitted.

Configuration Requirements:
Base setup 1 for Notification Forwarder object tests.
Local_Forwarding_Only = FALSE.
Notification source Message Text = (any valid message text with character encoding X and not empty)

Test Steps:

1. TRANSMIT AddListElement-Request,
   'Object Identifier' = (the object being tested),
   'Property Identifier' = Subscribed_Recipients,
   'List of Elements' = { DEST_OBJ_ID, -- Recipient D1
   DEST_PROCESS_ID, -- Process Identifier
   FALSE, -- Issue Confirmed Notifications
   TR -- Time Remaining where TR > test duration until WAIT statement
   } -- One list element
2. RECEIVE BACnet-SimpleACK-PDU
3. TRANSMIT SOURCE = DS, UnconfirmedEventNotification-Request,
   'Process Identifier' = SRC_PROCESS_ID,
   'Initiating Device Identifier' = SRC_NOTIF_DEV,
   'Event Object Identifier' = SRC_NOTIF_OBJ,
   'Time Stamp' = (any valid time stamp),
   'Notification Class' = SRC_NOTIF_CLS,
   'Priority' = (any valid priority),
   'Event Type' = (any valid event type),
   'Message Text' = (required, any valid message text with character encoding X and not empty),
   'Notify Type' = SRC_NOTIF_TYP,
   'AckRequired' = (any valid value), -- absent if Notify Type is ACK_NOTIFICATION
   'From State' = (any valid From_State), -- absent if Notify Type is ACK_NOTIFICATION
   'To State' = (any valid To_State),
   'Event Values' = (any valid event values) -- absent if Notify Type is ACK_NOTIFICATION
4. BEFORE ***Notification Fail Time***
   RECEIVE DESTINATION = D1, UnconfirmedEventNotification-Request,
   'Message Text' = (message text from previous step with character encoding = X)
   (Other parameters omitted for clarity)
5. WAIT (TR+1) minutes for Time Remaining to expire
6. MAKE (Recipient_List = {(all), -- Valid Days
   (all), -- From Time, To Time
   DEST_OBJ_ID, -- Recipient D1
   DEST_PROCESS_ID, -- Process Identifier
   FALSE, -- Issue Confirmed Notifications
   {T, T, T} -- Transitions
   }) -- One list element
7. TRANSMIT SOURCE = DS, UnconfirmedEventNotification-Request,
   'Process Identifier' = SRC_PROCESS_ID,
   'Initiating Device Identifier' = SRC_NOTIF_DEV,
   'Event Object Identifier' = SRC_NOTIF_OBJ,
   'Time Stamp' = (any valid time stamp),
   'Notification Class' = SRC_NOTIF_CLS,
   'Priority' = (any valid priority),
   'Event Type' = (any valid event type),
   'Message Text' = (any valid message text with character encoding X and not empty), --required for this test
   'Notify Type' = SRC_NOTIF_TYP,
   'AckRequired' = (any valid value), -- absent if Notify Type is ACK_NOTIFICATION
   'From State' = (any valid From_State), -- absent if Notify Type is ACK_NOTIFICATION
   'To State' = (any valid To_State),
   'Event Values' = (any valid event values) -- absent if Notify Type is ACK_NOTIFICATION
8. BEFORE ***Notification Fail Time***
   RECEIVE DESTINATION = D1, UnconfirmedEventNotification-Request,
   'Message Text' = (message text from previous step with character encoding = X)
   (Other parameters omitted for clarity)

7.3.2.30.6 Out_Of_Service Property Test

Purpose: This test case verifies that event forwarding is not done while Out_Of_Service is TRUE.

Test Concept: Set up both Recipient_List and Subscribed_Recipient recipient entries with no filters specified and then send event notifications to the Notification Forwarder while the value of the Out_Of_Service property is TRUE. Subscribed_Recipients are configured as part of base setup 2 for Notification Forwarder object tests. Verify that forwarding of the event notifications is not performed.

If the Out_Of_Service property of the object under test is not writable, and if the value of the property cannot be changed by other means, then this test shall be omitted.

Configuration Requirements: Base setup 2 for Notification Forwarder object tests with TR lifetime sufficient for this test.

Test Steps:

1. MAKE (Recipient_List = {(all), -- Valid Days
   (all), -- From Time, To Time
   DEST_OBJ_ID2, -- Recipient D2
   DEST_PROCESS_ID, -- Process Identifier
   FALSE, -- Issue Confirmed Notifications
   {T, T, T} -- Transitions
   }) -- One list element
2. MAKE (Out_Of_Service = TRUE)
3. VERIFY Out_Of_Service = TRUE
4. VERIFY Status_Flags = (?, FALSE, ?, TRUE)
5. TRANSMIT SOURCE = DS, UnconfirmedEventNotification-Request,
   'Process Identifier' = SRC_PROCESS_ID,
   'Initiating Device Identifier' = SRC_NOTIF_DEV,
   'Event Object Identifier' = SRC_NOTIF_OBJ,
   'Time Stamp' = (any valid time stamp),
   'Notification Class' = SRC_NOTIF_CLS,
   'Priority' = (any valid priority),
   'Event Type' = (any valid event type),
   'Message Text' = (optional, any valid message text),
   'Notify Type' = SRC_NOTIF_TYP,
   'AckRequired' = (any valid value), -- absent if Notify Type is ACK_NOTIFICATION
   'From State' = (any valid From_State), -- absent if Notify Type is ACK_NOTIFICATION
   'To State' = (any valid To_State),
   'Event Values' = (any valid event values) -- absent if Notify Type is ACK_NOTIFICATION
6. WAIT **Notification Fail Time**
7. CHECK (the IUT did not transmit an event notification)
8. MAKE (Out_Of_Service = FALSE)
9. VERIFY Out_Of_Service = FALSE
10. VERIFY Status_Flags = (?, ?, ?, FALSE)
11. TRANSMIT SOURCE = DS, UnconfirmedEventNotification-Request,
   'Process Identifier' = SRC_PROCESS_ID,
   'Initiating Device Identifier' = SRC_NOTIF_DEV,
   'Event Object Identifier' = SRC_NOTIF_OBJ,
   'Time Stamp' = (any valid time stamp),
   'Notification Class' = SRC_NOTIF_CLS,
   'Priority' = (any valid priority),
   'Event Type' = (any valid event type),
   'Message Text' = (optional, any valid message text),
   'Notify Type' = SRC_NOTIF_TYP,
   'AckRequired' = (any valid value), -- absent if Notify Type is ACK_NOTIFICATION
   'From State' = (any valid From_State), -- absent if Notify Type is ACK_NOTIFICATION

'To State' = (any valid To_State),
'Event Values' = (any valid event values) -- absent if Notify Type is ACK_NOTIFICATION

12. BEFORE ***Notification Fail Time*** --The following can be in any order
RECEIVE DESTINATION = D1, UnconfirmedEventNotification-Request
RECEIVE DESTINATION = D2, UnconfirmedEventNotification-Request

7.3.2.30.7 Recipient_List Property Tests

7.3.2.30.7.1 Destination Date Filtering Test

Purpose: Test destination date filtering operation by checking that forwarding of event notifications obeys the recipient validDays settings.

Test Concept: Configure a Recipient_List entry in the Notification Forwarder object such that the validDays is enabled for some days and disabled for some days. Set the IUT device date to an enabled day TIME_E. Send an event notification to the Notification Forwarder object and check that the event notification is forwarded. Change the IUT device's day to a disabled day TIME_D and send another event notification. Check that the event notification is not forwarded.

TIME_E is any time within the window defined by From Time and To Time in the Recipient_List that corresponds to one of the enabled days.

TIME_D is any time within the window defined by From Time and To Time in the Recipient_List that corresponds to one of the disabled days.

Configuration Requirements: Base setup 1 for Notification Forwarder object tests.

Test Steps:

1. MAKE (Recipient_List ={(at least one day of week has a value of TRUE, at least one day of week has a value of FALSE), --Valid Days
(any range sufficient for the duration of the test), -- From Time, To Time
DEST_OBJ_ID, -- Recipient D1
DEST_PROCESS_ID, -- Process Identifier
FALSE, -- Issue Confirmed Notifications
{T, T, T} -- Transitions
}) -- One list element
2. IF (IUT supports the TimeSynchronization service) THEN
TRANSMIT TimeSynchronization-Request,
'Time' = TIME_E
ELSE IF (IUT supports the UTCTimeSynchronization service) THEN
TRANSMIT UTCTimeSynchronization-Request,
'Time' = TIME_E
ELSE
MAKE (the local date and time = TIME_E)
3. TRANSMIT SOURCE = DS, UnconfirmedEventNotification-Request,
'Process Identifier' = SRC_PROCESS_ID,
'Initiating Device Identifier' = SRC_NOTIF_DEV,
'Event Object Identifier' = SRC_NOTIF_OBJ,
'Time Stamp' = (any valid time stamp),
'Notification Class' = SRC_NOTIF_CLS,
'Priority' = (any valid priority),
'Event Type' = (any valid event type),
'Message Text' = (optional, any valid message text),
'Notify Type' = SRC_NOTIF_TYP,
'AckRequired' = (any valid value), -- absent if Notify Type is ACK_NOTIFICATION
'From State' = (any valid From_State), -- absent if Notify Type is ACK_NOTIFICATION
'To State' = (any valid To_State),

'Event Values' = (any valid event values) -- absent if Notify Type is ACK_NOTIFICATION

4. BEFORE ***Notification Fail Time***
   RECEIVE DESTINATION = D1, UnconfirmedEventNotification-Request
   'Process Identifier' = DEST_PROCESS_ID,
   (other parameter values match the parameter values used in the previous step)
5. IF (IUT supports the TimeSynchronization service) THEN
   TRANSMIT TimeSynchronization-Request,
   'Time' = TIME_D
   ELSE IF (IUT supports the UTCTimeSynchronization service) THEN
   TRANSMIT UTCTimeSynchronization-Request,
   'Time' = TIME_D
   ELSE
   MAKE (the local date and time = TIME_D)
7. TRANSMIT SOURCE = DS, UnconfirmedEventNotification-Request,
   'Process Identifier' = SRC_PROCESS_ID,
   'Initiating Device Identifier' = SRC_NOTIF_DEV,
   'Event Object Identifier' = SRC_NOTIF_OBJ,
   'Time Stamp' = (any valid time stamp),
   'Notification Class' = SRC_NOTIF_CLS,
   'Priority' = (any valid priority),
   'Event Type' = (any valid event type),
   'Message Text' = (optional, any valid message text),
   'Notify Type' = SRC_NOTIF_TYP,
   'AckRequired' = (any valid value), -- absent if Notify Type is ACK_NOTIFICATION
   'From State' = (any valid From_State), -- absent if Notify Type is ACK_NOTIFICATION
   'To State' = (any valid To_State),
   'Event Values' = (any valid event values) -- absent if Notify Type is ACK_NOTIFICATION
8. WAIT **Notification Fail Time**
9. CHECK (the IUT did not transmit an event notification)
10. IF (IUT supports the TimeSynchronization service) THEN
    TRANSMIT TimeSynchronization-Request,
    'Time' = (current date and time)
    ELSE IF (IUT supports the UTCTimeSynchronization service) THEN
    TRANSMIT UTCTimeSynchronization-Request,
    'Time' = (current date and time)
    ELSE
    MAKE (the local date and time = (current date and time))

7.3.2.30.7.2 Destination Time Filtering Test

Purpose: Test destination time filtering operation by checking that forwarding of event notifications obeys the recipient fromTime and toTime settings.

Test Concept: Configure a Recipient_List entry in the Notification_Forwarder object such that the fromTime and toTime includes some period of time. Set the IUT device time to a value TIME_E between fromTime and toTime. Send an event notification to the Notification Forwarder object and check that the event notification is forwarded. Change the IUT device's time to TIME_D so that it is outside the range of fromTime and toTime and send another event notification. Check that the event notification is not forwarded.

TIME_E is any time within the window defined by From Time and To Time in the Recipient_List that corresponds to one of the enabled days.

TIME_D is any time outside the window defined by From Time and To Time in the Recipient_List that corresponds to one of the enabled days.

Configuration Requirements: Base setup 1 for Notification Forwarder object tests.

Test Steps:

1. MAKE (Recipient_List = {(at least one day of week has a value of TRUE), -- Valid Days
   (any range sufficient for the duration of the test), -- From Time, To Time
   DEST_OBJ_ID, -- Recipient D1
   DEST_PROCESS_ID, -- Process Identifier
   FALSE, -- Issue Confirmed Notifications
   {T, T, T} -- Transitions
   }) -- One list element
2. IF (IUT supports the TimeSynchronization service) THEN
   TRANSMIT TimeSynchronization-Request,
   'Time' = TIME_E
   ELSE IF (IUT supports the UTCTimeSynchronization service) THEN
   TRANSMIT UTCTimeSynchronization-Request,
   'Time' = TIME_E
   ELSE
   MAKE (the local date and time = TIME_E)
3. TRANSMIT SOURCE = DS, UnconfirmedEventNotification-Request,
   'Process Identifier' = SRC_PROCESS_ID,
   'Initiating Device Identifier' = SRC_NOTIF_DEV,
   'Event Object Identifier' = SRC_NOTIF_OBJ,
   'Time Stamp' = (any valid time stamp),
   'Notification Class' = SRC_NOTIF_CLS,
   'Priority' = (any valid priority),
   'Event Type' = (any valid event type),
   'Message Text' = (optional, any valid message text),
   'Notify Type' = SRC_NOTIF_TYP,
   'AckRequired' = (any valid value), -- absent if Notify Type is ACK_NOTIFICATION
   'From State' = (any valid From_State), -- absent if Notify Type is ACK_NOTIFICATION
   'To State' = (any valid To_State),
   'Event Values' = (any valid event values) -- absent if Notify Type is ACK_NOTIFICATION
4. BEFORE ***Notification Fail Time***
   RECEIVE DESTINATION = D1, UnconfirmedEventNotification-Request
   'Process Identifier' = DEST_PROCESS_ID,
   (other parameter values match the parameter values used in the previous step)
5. IF (IUT supports the TimeSynchronization service) THEN
   TRANSMIT TimeSynchronization-Request,
   'Time' = TIME_D
   ELSE IF (IUT supports the UTCTimeSynchronization service) THEN
   TRANSMIT UTCTimeSynchronization-Request,
   'Time' = TIME_D
   ELSE
   MAKE (the local date and time = TIME_D)
6. TRANSMIT SOURCE = DS, UnconfirmedEventNotification-Request,
   'Process Identifier' = SRC_PROCESS_ID,
   'Initiating Device Identifier' = SRC_NOTIF_DEV,
   'Event Object Identifier' = SRC_NOTIF_OBJ,
   'Time Stamp' = (any valid time stamp),
   'Notification Class' = SRC_NOTIF_CLS,
   'Priority' = (any valid priority),
   'Event Type' = (any valid event type),
   'Message Text' = (optional, any valid message text),
   'Notify Type' = SRC_NOTIF_TYP,
   'AckRequired' = (any valid value), -- absent if Notify Type is ACK_NOTIFICATION
   'From State' = (any valid From_State), -- absent if Notify Type is ACK_NOTIFICATION

'To State' = (any valid To_State),
'Event Values' = (any valid event values) -- absent if Notify Type is ACK_NOTIFICATION

8. WAIT **Notification Fail Time**
9. CHECK (the IUT did not transmit an event notification)
10. IF (IUT supports the TimeSynchronization service) THEN
TRANSMIT TimeSynchronization-Request,
'Time' = (current date and time)
ELSE IF (IUT supports the UTCTimeSynchronization service) THEN
TRANSMIT UTCTimeSynchronization-Request,
'Time' = current date and time)
ELSE
MAKE (the local date and time = (current date and time))

7.3.2.30.7.3 Process Identifier Test

Purpose: Check that the process identifier values used in forwarded event notifications is equal to the values used in the recipient configurations contained in the IUT.

Test Concept: Perform tests **7.3.2.30.2 Recipient_List Forwarding Test** and **7.3.2.30.3 Subscribed_Recipients Forwarding Test** for different values of SRC_PROCESS_ID, and DEST_PROCESS_ID. The following test cases are required at a minimum.

Case 1 - SRC_PROCESS_ID <> DEST_PROCESS_ID and both are non-zero.

Case 2 - SRC_PROCESS_ID = DEST_PROCESS_ID and both are non-zero.

7.3.2.30.7.4 Destination Transition Filtering Test

Purpose: To verify that notification messages are forwarded only if the Recipient_List Transitions bit parameter corresponding to the event transition is set.

Test Concept: The IUT is configured such that the Transitions parameter indicates that some event transitions are to trigger an event notification and some are not. Each event transition is triggered, and the IUT is monitored to verify that notification messages are forwarded only for those transitions for which the Transitions parameter has a value of TRUE.

Configuration Requirements: Base setup 1 for Notification Forwarder object tests.

Test Steps:

1. MAKE (Recipient_List = {(all), -- Valid Days
(all), -- From Time, To Time
DEST_OBJ_ID, -- Recipient D1
DEST_PROCESS_ID, -- Process Identifier
FALSE, -- Issue Confirmed Notifications
any valid value} -- Transitions
}) - -- One list element
2. TRANSMIT SOURCE = DS, UnconfirmedEventNotification-Request,
'Process Identifier' = SRC_PROCESS_ID,
'Initiating Device Identifier' = SRC_NOTIF_DEV,
'Event Object Identifier' = SRC_NOTIF_OBJ,
'Time Stamp' = (any valid time stamp),
'Notification Class' = SRC_NOTIF_CLS,
'Priority' = (any valid priority),
'Event Type' = (any valid event type),
'Message Text' = (optional, any valid message text),
'Notify Type' = SRC_NOTIF_TYP, -- choose ALARM or EVENT
'AckRequired' = (any valid value),

'From State' = normal,
'To State' = offnormal, -- offnormal transition
'Event Values' = (any valid event values)

3. IF (offnormal transitions are enabled in the Recipient_List)
   BEFORE ***Notification Fail Time***
      RECEIVE DESTINATION = D1, UnconfirmedEventNotification-Request
         'Process Identifier' = DEST_PROCESS_ID,
         (other parameter values match the parameter values used in the previous step)
   ELSE
      WAIT **Notification Fail Time**
      CHECK (the IUT did not transmit an event notification)
4. TRANSMIT SOURCE = DS, UnconfirmedEventNotification-Request,
   'Process Identifier' = SRC_PROCESS_ID,
   'Initiating Device Identifier' = SRC_NOTIF_DEV,
   'Event Object Identifier' = SRC_NOTIF_OBJ,
   'Time Stamp' = (any valid time stamp),
   'Notification Class' = SRC_NOTIF_CLS,
   'Priority' = (any valid priority),
   'Event Type' = (any valid event type),
   'Message Text' = (optional, any valid message text),
   'Notify Type' = SRC_NOTIF_TYP, -- choose ALARM or EVENT
   'AckRequired' = (any valid value),
   'From State' = offnormal,
   'To State' = normal, -- normal transition
   'Event Values' = (any valid event values)
5. IF (normal transitions are enabled in the Recipient_List) *THEN*
   BEFORE ***Notification Fail Time***
      RECEIVE DESTINATION = D1, UnconfirmedEventNotification-Request
         'Process Identifier' = DEST_PROCESS_ID,
         (other parameter values match the parameter values used in the previous step)
   ELSE
      WAIT **Notification Fail Time**
      CHECK (the IUT did not transmit an event notification)
6. TRANSMIT SOURCE = DS, UnconfirmedEventNotification-Request,
   'Process Identifier' = SRC_PROCESS_ID,
   'Initiating Device Identifier' = SRC_NOTIF_DEV,
   'Event Object Identifier' = SRC_NOTIF_OBJ,
   'Time Stamp' = (any valid time stamp),
   'Notification Class' = SRC_NOTIF_CLS,
   'Priority' = (any valid priority),
   'Event Type' = (any valid event type),
   'Message Text' = (optional, any valid message text),
   'Notify Type' = SRC_NOTIF_TYP, -- choose ALARM or EVENT
   'AckRequired' = (any valid value),
   'From State' = normal,
   'To State' = fault, -- fault transition
   'Event Values' = (any valid event values)
7. IF (fault transitions are enabled in the Recipient_List) *THEN*
   BEFORE ***Notification Fail Time***
      RECEIVE DESTINATION = D1, UnconfirmedEventNotification-Request
         'Process Identifier' = DEST_PROCESS_ID,
         (other parameter values match the parameter values used in the previous step)
   ELSE
      WAIT **Notification Fail Time**
      CHECK (the IUT did not transmit an event notification)

7.3.2.30.8 Subscribed_Recipients Property Test

7.3.2.30.8.1 Time Count Down Test

Purpose: Verify Subscribed_Recipients entries count Time Remaining down.

Test Concept: Add a subscription to a Notification Forwarder and check that the Subscribed_Recipients Time Remaining value decreases after time has passed.

Note to tester: If the device timing resolution is significantly greater than 1 minute, the wait times called for by the test steps may be extended.

Configuration Requirements: Base setup 2 for Notification Forwarder object tests with TR lifetime sufficient for this test.

Test Steps:

1. READ SR1 = Subscribed_Recipients
2. CHECK (SR1 = {DEST_OBJ_ID, -- Recipient D1
   DEST_PROCESS_ID, -- Process Identifier
   FALSE, -- Issue Confirmed Notifications
   TR1 -- Time Remaining where (TR-1) <= TR1 <= TR
   }) -- One list element
3. WAIT greater of 2 minutes or twice timing resolution
4. READ SR2 = Subscribed_Recipients
5. CHECK (SR2 = {DEST_OBJ_ID, -- Recipient D1
   DEST_PROCESS_ID, -- Process Identifier
   FALSE, -- Issue Confirmed Notifications
   TR2 -- Time Remaining where TR2 < TR1
   }) -- One list element

7.3.2.30.8.2 Expiration Test

Purpose: Verify Subscribed_Recipients entries expire when Time Remaining reaches zero.

Test Concept: Add a subscription to a Notification Forwarder and verify that the subscription was successful. Then wait for the Subscribed_Recipients Time Remaining value to expire and check that the subscription has been removed from the Subscribed_Recipients list.

Note to tester: If the device timing resolution is significantly greater than 1 minute, the wait times called for by the test steps may be extended.

Configuration Requirements: Base setup 1 for Notification Forwarder object tests.

Test Steps:

1. TRANSMIT AddListElement-Request,
   'Object Identifier' = (the object being tested),
   'Property Identifier' = Subscribed_Recipients,
   'List of Elements' = {DEST_OBJ_ID, -- Recipient D1
   DEST_PROCESS_ID, -- Process Identifier
   FALSE, -- Issue Confirmed Notifications
   2 -- Time Remaining minutes
   } -- One list element
2. RECEIVE BACnet-SimpleACK-PDU
3. READ SR = Subscribed_Recipients
4. CHECK (SR = {DEST_OBJ_ID, -- Recipient D1
   DEST_PROCESS_ID, -- Process Identifier

```
                FALSE,              -- Issue Confirmed Notifications
                TR1                 -- Time Remaining where 1 <= TR1 <= 2
                })                  -- One list element
```

5. WAIT more than greater of 2 minutes or twice timing resolution
6. VERIFY Subscribed_Recipients = { } -- Empty list

7.3.2.30.8.3 Time Renewal Test

Purpose: Verify that a Subscribed_Recipients resubscription renews the Time Remaining value.

Test Concept: Add a Subscribed_Recipients subscription to a Notification Forwarder and wait until the Time Remaining value should have decreased. Renew the subscription and check that the Time Remaining has been restored to near the original value and that no additional list entries have been made.

Note to tester: If the device timing resolution is significantly greater than 1 minute, the wait times called for by the test steps may be extended.

Configuration Requirements: Base setup 2 for Notification Forwarder object tests with TR lifetime sufficient for this test.

Test Steps:

1. WAIT greater of 2 minutes or twice timing resolution
2. TRANSMIT AddListElement-Request,

```
    'Object Identifier' =   (the object being tested),
    'Property Identifier' = Subscribed_Recipients,
    'List of Elements' =    {DEST_OBJ_ID,          -- Recipient D1
                             DEST_PROCESS_ID,      -- Process Identifier
                             FALSE,                -- Issue Confirmed Notifications
                             TR                    -- Time Remaining
                             }                     -- One list element
```

3. RECEIVE BACnet-SimpleACK-PDU
4. READ SR = Subscribed_Recipients
5. CHECK (SR =

```
                {DEST_OBJ_ID,          -- Recipient D1
                 DEST_PROCESS_ID,      -- Process Identifier
                 FALSE,                -- Issue Confirmed Notifications
                 TR1                   -- Time Remaining where (TR-1) <= TR1 <= TR
                 })                    -- One list element
```

7.3.2.30.8.4 Resubscription Update Test

Purpose: Verify that a Subscribed_Recipients resubscription replaces Time Remaining and Confirmed Notification values.

Test Concept: Add a Subscribed_Recipients subscription to a Notification Forwarder. Then renew the subscription with a different Time Remaining value and Confirmed Notification value and check that the Time Remaining and Confirmed Notification values have been replaced and no additional list entries have been made.

Configuration Requirements: Base setup 2 for Notification Forwarder object tests with TR lifetime sufficient for this test.

Test Steps:

1. TRANSMIT AddListElement-Request,

```
    'Object Identifier' =   (the object being tested),
    'Property Identifier' = Subscribed_Recipients,
    'List of Elements' =    {DEST_OBJ_ID,          -- Recipient D1
                             DEST_PROCESS_ID,      -- Process Identifier
                             DEST_CONF_NOTIF1,     -- Issue Confirmed Notifications<>DEST_CONF_NOTIF
                             TR2                   -- Time Remaining where TR2 > test duration and TR2 <> TR
```

```
                        }                       -- One list element
2.  RECEIVE BACnet-SimpleACK-PDU
3.  READ SR = Subscribed_Recipients
4.  CHECK (SR =     {DEST_OBJ_ID,               -- Recipient D1
                    DEST_PROCESS_ID,            -- Process Identifier
                    DEST_CONF_NOTIF1,           -- Issue Confirmed Notifications
                    TR3                         -- Time Remaining where (TR2-1) <= TR3 <= TR2
                    })                          -- One list element
```

7.3.2.30.8.5 Delete Test

Purpose: Verify RemoveListElement can delete a Subscribed_Recipients list entry.

Test Concept: Add a Subscribed_Recipients subscription to a Notification Forwarder. Then delete the subscription and check that the subscription has been removed.

Configuration Requirements: Base setup 2 for Notification Forwarder object tests with TR lifetime sufficient for this test.

Test Steps:

```
1.  TRANSMIT RemoveListElement-Request,
    'Object Identifier' =       (the object being tested),
        'Property Identifier' = Subscribed_Recipients,
        'List of Elements' =    {DEST_OBJ_ID,           -- Recipient D1
                                DEST_PROCESS_ID         -- Process Identifier
                                }                       -- One list element
2.  RECEIVE BACnet-SimpleACK-PDU
3.  VERIFY Subscribed_Recipients = {  }                 -- Empty list
```

7.3.2.30.8.6 Subscription Of Similar Entries Test

Purpose: Verify multiple Subscribed_Recipients subscriptions with minimal differences can be added as separate entries.

Test Concept: Add a Subscribed_Recipients subscription, then add it again with a different Recipient, then add it again with a different Process Identifier. Verify that the Subscribed_Recipients property list size is three and that all three subscriptions were added.

Configuration Requirements: Base setup 2 for Notification Forwarder object tests with TR lifetime sufficient for this test.

Test Steps:

```
1.  TRANSMIT AddListElement-Request,
    'Object Identifier' =       (the object being tested),
        'Property Identifier' = Subscribed_Recipients,
        'List of Elements' =    {DEST_OBJ_ID2,          -- Recipient D2
                                DEST_PROCESS_ID,        -- Process Identifier
                                FALSE,                  -- Issue Confirmed Notifications
                                TR                      -- Time Remaining where TR > test duration
                                }                       -- One list element
2.  RECEIVE BACnet-SimpleACK-PDU
3.  TRANSMIT AddListElement-Request,
        'Object Identifier' =   (the object being tested),
        'Property Identifier' = Subscribed_Recipients,
        'List of Elements' =    {DEST_OBJ_ID,-- Recipient D1
                                B,          -- Process Identifier where B <> DEST_PROCESS_ID
                                FALSE,      -- Issue Confirmed Notifications
                                TR          -- Time Remaining where TR > test duration
```

```
                                }                   -- One list element
4.  RECEIVE BACnet-SimpleACK-PDU
5.  READ SR = Subscribed_Recipients
6.  CHECK (SR =     {{DEST_OBJ_ID,          -- Recipient D1
                    DEST_PROCESS_ID,        -- Process Identifier
                    FALSE,                  -- Issue Confirmed Notifications
                    TR1                     -- Time Remaining where TR1 <= TR
                    },                      -- List element in any order
                    {DEST_OBJ_ID2,          -- Recipient D2
                    DEST_PROCESS_ID,        -- Process Identifier
                    FALSE,                  -- Issue Confirmed Notifications
                    TR2                     -- Time Remaining where TR2 <= TR
                    },                      -- List element in any order
                    {DEST_OBJ_ID,           -- Recipient D1
                    B,                      -- Process Identifier
                    FALSE,                  -- Issue Confirmed Notifications
                    TR3                     -- Time Remaining where TR3 <= TR
                    }                       -- List element in any order
                    })                      -- Three list elements in any order
```

7.3.2.30.9 Process_Identifier_Filter Property Test

7.3.2.30.9.1 NULL And Unsigned32 Choice Test

Purpose: Ensure writable Process_Identifier_Filter properties allow a choice of both NULL and Unsigned32 values.

Test Concept: Write a NULL Process_Identifier_Filter property value and verify that it was saved. Then write an Unsigned32 value = X Process_Identifier_Filter property value and verify that it was saved where X is any non-null legal process identifier value.

If no Notification Forwarder object in the IUT has a writable Process_Identifier_Filter property, then omit this test.

Configuration Requirements: Base setup 1 for Notification Forwarder object tests.

Test Steps:

1. WRITE Process_Identifier_Filter = NULL
2 VERIFY Process_Identifier_Filter = NULL
3 WRITE Process_Identifier_Filter = X
4 VERIFY Process_Identifier_Filter = X

7.3.2.30.9.2 NULL Unfiltered Process Identifier Test

Purpose: Ensure a NULL value Process_Identifier_Filter property allows forwarding of all event notification process identifier values.

Test Concept: Perform tests **7.3.2.30.2 Recipient_List Forwarding Test** and **7.3.2.30.3 Subscribed_Recipients Forwarding Test** for non-zero values of SRC_PROCESS_ID. Select a non-zero value of SRC_PROCESS_ID within the BACnet allowable range and perform each test listed.

If no Notification Forwarder object in the IUT has a Process_Identifier_Filter value that can be made NULL, then omit this test.

7.3.2.30.9.3 Zero Unfiltered Process Identifier Test

Purpose: Ensure a zero value Process_Identifier_Filter property allows forwarding of all event notification process identifier values.

Test Concept: Define a Subscribed_Recipient entry and set the Process_Identifier_Filter to zero. Then send an event notification with a non-zero process identifier value to the notification forwarder. Verify that forwarding of the event notification is performed.

If no Notification Forwarder object in the IUT has a Process_Identifier_Filter value that can be made zero, then omit this test.

Configuration Requirements: Base setup 2 for Notification Forwarder object tests with TR lifetime sufficient for this test.

Test Steps:

1. MAKE (Process_Identifier_Filter = zero)
2. TRANSMIT SOURCE = DS, UnconfirmedEventNotification-Request,
   'Process Identifier' = SRC_PROCESS_ID,
   'Initiating Device Identifier' = SRC_NOTIF_DEV,
   'Event Object Identifier' = SRC_NOTIF_OBJ,
   'Time Stamp' = (any valid time stamp),
   'Notification Class' = SRC_NOTIF_CLS,
   'Priority' = (any valid priority),
   'Event Type' = (any valid event type),
   'Message Text' = (optional, any valid message text),
   'Notify Type' = SRC_NOTIF_TYP, -- choose ALARM or EVENT
   'AckRequired' = (any valid value),
   'From State' = (any valid From_State),
   'To State' = (any valid To_State),
   'Event Values' = (any valid event values)
3. BEFORE ***Notification Fail Time***
   RECEIVE DESTINATION = D1, UnconfirmedEventNotification-Request
   'Process Identifier' = DEST_PROCESS_ID,
   (other parameter values match the parameter values used in the previous step)

7.3.2.30.9.4 Specific Value Process Identifier Test

Purpose: Ensure a non-NULL and non-zero value Process_Identifier_Filter property allows forwarding of only event notifications with process identifier values matching the Process_Identifier_Filter value.

Test Concept: Define a Subscribed_Recipient entry and set the Process_Identifier_Filter to a non-zero value. Then send an event notification with a non-zero process identifier value = X to the notification forwarder. Verify that forwarding of the event notification is only performed if the IUT Process_Identifier_Filter value matches the event notification Process_Identifier value.

If no Notification Forwarder object in the IUT has a Process_Identifier_Filter value that can be made non-NULL and non-zero, then omit this test.

Configuration Requirements: Base setup 2 for Notification Forwarder object tests with TR lifetime sufficient for this test.

Test Steps:

1. MAKE (Process_Identifier_Filter = X where X is not NULL and not zero)
2. TRANSMIT SOURCE = DS, UnconfirmedEventNotification-Request,
   'Process Identifier' = SRC_PROCESS_ID,
   'Initiating Device Identifier' = SRC_NOTIF_DEV,
   'Event Object Identifier' = SRC_NOTIF_OBJ,
   'Time Stamp' = (any valid time stamp),
   'Notification Class' = SRC_NOTIF_CLS,
   'Priority' = (any valid priority),
   'Event Type' = (any valid event type),

'Message Text' = (optional, any valid message text),
'Notify Type' = SRC_NOTIF_TYP, -- choose ALARM or EVENT
'AckRequired' = (any valid value),
'From State' = (any valid From_State),
'To State' = (any valid To_State),
'Event Values' = (any valid event values)

3. IF (SRC_PROCESS_ID = X) THEN
   BEFORE ***Notification Fail Time***
   RECEIVE DESTINATION = D1, UnconfirmedEventNotification-Request
   'Process Identifier' = DEST_PROCESS_ID,
   (other parameter values match the parameter values used in the previous step)
   ELSE
   WAIT **Notification Fail Time**
   CHECK (the IUT did not transmit an event notification)

7.3.2.30.9.5 Fixed Process_Identifier_Filter Test

Purpose: Test required configurability of property Process_Identifier_Filter.

Test Concept: If the Process_Identifier_Filter is not configurable or writable, check that at least one Notification Forwarder instance in the device under test has a NULL or zero Process_Identifier_Filter property value.

If Process_Identifier_Filter is configurable or writable, then omit this test.

Configuration Requirements: Base setup 1 for Notification Forwarder object tests.

Test Steps:

1. RESULT = 0
2. REPEAT ObjectX = (use each Notification Forwarder object in the IUT) DO
   {READ PID = ObjectX, Process_Identifier_Filter
   IF (PID = zero or NULL) THEN
   RESULT = 1
   }
3. CHECK (RESULT = 1)

7.3.2.30.10 Port_Filter Test

Purpose: Test that Notification Forwarder object Port_Filter properties in routers are writable and that event notifications are only forwarded if the Port_Filter property is set to allow the receiving port to forward the event.

Test Concept: Port P has a Port_ID value P_ID and is the router port where an event notification can be received by the IUT when sent from DS. Define D1 as a recipient in both the Recipient_List and the Subscribed_Recipients properties. D1 can be reachable through any port on the IUT device. Read the existing Port_Filter attribute and its array size and search the attribute array Port_ID entries for the entry matching P_ID indicating this is the correct port for a source event notification to reach the IUT from DS. Enable all ports except this one and send an event notification from DS to be forwarded. The IUT should not forward the event since the receiving port is disabled. Disable all ports except port P with Port_ID value P_ID and send an event notification to be forwarded. The IUT should now forward the event since the receiving port is enabled.

If the IUT device is not a router, then omit this test. If the IUT device cannot contain a Notification Forwarder object with a Local_Forwarding_Only property that has a value of FALSE, then omit this test.

Configuration Requirements: Base setup 1 for Notification Forwarder object tests.

Test Steps:

1. MAKE (Local_Forwarding_Only = FALSE)
2. MAKE (Recipient_List = {(all), -- Valid Days
(all), -- From Time, To Time
DEST_OBJ_ID, -- Recipient D1
DEST_PROCESS_ID, -- Process Identifier
FALSE, -- Issue Confirmed Notifications
{T, T, T} -- Transitions
}) -- One list element
3. TRANSMIT AddListElement-Request,
'Object Identifier' = (the object being tested),
'Property Identifier' = Subscribed_Recipients,
'List of Elements' = {DEST_OBJ_ID, -- Recipient D1
DEST_PROCESS_ID, -- Process Identifier
FALSE, -- Issue Confirmed Notifications
TR -- Time Remaining where TR > test duration
} -- One list element
4. RECEIVE BACnet-SimpleACK-PDU
5. READ PF1 = Port_Filter
6. READ PF1_SIZE = Port_Filter, ARRAY_INDEX = 0
7. WHILE (PF1 index X Port_ID <> P_ID) DO
(Next index X)
8. WHILE (Z = 1 to PF1_SIZE) DO
IF (Z <> X) THEN
WRITE Port_Filter = (Port_ID from PF1 index Z, Enabled TRUE), ARRAY_INDEX = Z
ELSE
WRITE Port_Filter = (Port_ID from PF1 index Z, Enabled FALSE), ARRAY_INDEX = Z
9. TRANSMIT SOURCE = DS, UnconfirmedEventNotification-Request,
'Process Identifier' = SRC_PROCESS_ID,
'Initiating Device Identifier' = SRC_NOTIF_DEV,
'Event Object Identifier' = SRC_NOTIF_OBJ,
'Time Stamp' = (any valid time stamp),
'Notification Class' = SRC_NOTIF_CLS,
'Priority' = (any valid priority),
'Event Type' = (any valid event type),
'Message Text' = (optional, any valid message text),
'Notify Type' = SRC_NOTIF_TYP,
'AckRequired' = (any valid value), -- absent if Notify Type is ACK_NOTIFICATION
'From State' = (any valid From_State), -- absent if Notify Type is ACK_NOTIFICATION
'To State' = (any valid To_State),
'Event Values' = (any valid event values) -- absent if Notify Type is ACK_NOTIFICATION
10. WAIT **Notification Fail Time**
11. CHECK (the IUT did not transmit an event notification)
12. WHILE (Z = 1 to PF1_SIZE) DO
IF (Z <> X) THEN
WRITE Port_Filter = (Port_ID from PF1 index Z, Enabled FALSE), ARRAY_INDEX = Z
ELSE
WRITE Port_Filter = (Port_ID from PF1 index Z, Enabled TRUE), ARRAY_INDEX = Z
13. TRANSMIT SOURCE = DS, UnconfirmedEventNotification-Request,
'Process Identifier' = SRC_PROCESS_ID,
'Initiating Device Identifier' = SRC_NOTIF_DEV,
'Event Object Identifier' = SRC_NOTIF_OBJ,
'Time Stamp' = (any valid time stamp),
'Notification Class' = SRC_NOTIF_CLS,
'Priority' = (any valid priority),
'Event Type' = (any valid event type),
'Message Text' = (optional, any valid message text),

'Notify Type' = SRC_NOTIF_TYP,
'AckRequired' = (any valid value), -- absent if Notify Type is ACK_NOTIFICATION
'From State' = (any valid From_State), -- absent if Notify Type is ACK_NOTIFICATION
'To State' = (any valid To_State),
'Event Values' = (any valid event values) -- absent if Notify Type is ACK_NOTIFICATION

14. BEFORE ***Notification Fail Time***
RECEIVE DESTINATION = D1, UnconfirmedEventNotification-Request
'Process Identifier' = DEST_PROCESS_ID,
(other parameter values match the parameter values used in the previous step)

7.3.2.30.11 Local_Forwarding_Only Property Tests

7.3.2.30.11.1 Only Forwards Locally When True

Purpose: Verify a Notification Forwarder object with Local_Forwarding_Only set TRUE will only forward locally generated events.

Test Concept: Set the Local_Forwarding_Only property in the IUT to TRUE and add a Notification Forwarder subscription with D1 as the recipient. Send an Event Notification from DS to the IUT and check that the event is not forwarded. Send a different Event Notification from a source within the IUT and check that the event was forwarded.

If the IUT device cannot contain a Notification Forwarder object with a Local_Forwarding_Only property that has a value of TRUE, then omit this test.

Configuration Requirements:

Begin with base setup 1 for Notification Forwarder object tests, perform the following additional steps:

Notification source object in DS configured as follows:
Source notification device = SRC_NOTIF_DEV1,
Source notification object = SRC_NOTIF_OBJ1,
Notification_Class = SRC_NOTIF_CLS1
Event_Enable = {T, T, T}

Notification source object in IUT configured as follows:
Source notification device = IUT,
Source notification object = SRC_NOTIF_OBJ2,
Notification_Class = SRC_NOTIF_CLS2
Event_Enable = {T, T, T}

Source notification class object SRC_NOTIF_CLS1 in DS configured as follows:
Notification Class = (instance number of notification class object),
Priority = (any valid priority)
Ack_Required = (any valid Ack_Required value)
Recipient_List = { (all), -- Valid Days
(all), -- From Time, To Time
IUT -- Recipient
} --One list element
Process Identifier = SRC_PROCESS_ID
Issue Confirmed Notifications = FALSE
Transitions = {T, T, T}

Source notification class object SRC_NOTIF_CLS2 in IUT configured as follows:
Notification Class = (instance number of notification class object),
Priority = (any valid priority)
Ack_Required = (any valid Ack_Required value)
Recipient_List = { (all), -- Valid Days
(all), -- From Time, To Time

```
                                    IUT                 -- Recipient
                                  }  -- One list element
        Process Identifier          = SRC_PROCESS_ID
        Issue Confirmed Notifications = FALSE
        Transitions                 = {T, T, T}
```

Note to Tester: Issue Confirmed Notifications is specified as FALSE within the test. Thus UnconfirmedEventNotification-Request messages from the Notification source object are expected. Behaviors can alternately be tested using Issue Confirmed Notifications set to TRUE resulting in ConfirmedEventNotification service messages from the Notification source object that must be acknowledged at the communication level, but it is not necessary to test both settings unless specifically directed otherwise by the test.

Test Steps:

1. MAKE (Local_Forwarding_Only = TRUE)
2. TRANSMIT AddListElement-Request,
   'Object Identifier' = (the object being tested),
   'Property Identifier' = Subscribed_Recipients,
   'List of Elements' = {DEST_OBJ_ID, -- Recipient D1
   DEST_PROCESS_ID, -- Process Identifier
   FALSE, -- Issue Confirmed Notifications
   TR -- Time Remaining where TR > test duration
   } -- One list element
3. RECEIVE BACnet-SimpleACK-PDU
4. TRANSMIT SOURCE = DS, UnconfirmedEventNotification-Request,
   'Process Identifier' = SRC_PROCESS_ID,
   'Initiating Device Identifier' = SRC_NOTIF_DEV1, -- DS
   'Event Object Identifier' = SRC_NOTIF_OBJ1,
   'Time Stamp' = (any valid time stamp),
   'Notification Class' = SRC_NOTIF_CLS1,
   'Priority' = (any valid priority),
   'Event Type' = (any valid event type),
   'Message Text' = (optional, any valid message text),
   'Notify Type' = SRC_NOTIF_TYP,
   'AckRequired' = (any valid value), -- absent if Notify Type is ACK_NOTIFICATION
   'From State' = (any valid From_State), -- absent if Notify Type is ACK_NOTIFICATION
   'To State' = (any valid To_State),
   'Event Values' = (any valid event values) -- absent if Notify Type is ACK_NOTIFICATION
5. WAIT **Notification Fail Time**
6. CHECK (the IUT did not transmit an event notification)
7. MAKE (IUT Notification source object generate an UnconfirmedEventNotification-Request,
   'Process Identifier' = SRC_PROCESS_ID,
   'Initiating Device Identifier' = SRC_NOTIF_DEV2, --IUT
   'Event Object Identifier' = SRC_NOTIF_OBJ2,
   'Time Stamp' = (any valid time stamp),
   'Notification Class' = SRC_NOTIF_CLS2,
   'Priority' = (any valid priority),
   'Event Type' = (any valid event type),
   'Message Text' = (optional, any valid message text),
   'Notify Type' = SRC_NOTIF_TYP,
   'AckRequired' = (any valid value), -- absent if Notify Type is ACK_NOTIFICATION
   'From State' = (any valid From_State), -- absent if Notify Type is ACK_NOTIFICATION
   'To State' = (any valid To_State),
   'Event Values' = (any valid event values))-- absent if Notify Type is ACK_NOTIFICATION
8. BEFORE ***Notification Fail Time***
   RECEIVE DESTINATION = D1, UnconfirmedEventNotification-Request

'Process Identifier' = DEST_PROCESS_ID,
(other parameter values match the parameter values used in the previous step)

7.3.2.30.11.2 Forwards Locally And Remotely When False

Purpose: Verify a Notification Forwarder object with Local_Forwarding_Only set FALSE will forward both locally generated events and externally generated events.

Test Concept: Set the Local_Forwarding_Only property in the IUT to FALSE and add a Notification Forwarder subscription with D1 as the recipient. Send an Event Notification from DS to the IUT and check that the event is forwarded. Send a different Event Notification from a source within the IUT and check that the event was forwarded.

If the IUT device cannot contain a Notification Forwarder object with either a writable Local_Forwarding_Only property, or a Local_Forwarding_Only property that has a value of FALSE, then omit this test.

Configuration Requirements:

Begin with base setup 1 for Notification Forwarder object tests, perform the following additional steps:

Notification source object in DS configured as follows:

```
Source notification device      = SRC_NOTIF_DEV1,
Source notification object      = SRC_NOTIF_OBJ1,
Notification_Class              = SRC_NOTIF_CLS1
Event_Enable                    = {T, T, T}
```

Notification source object in IUT configured as follows:

```
Source notification device      = IUT,
Source notification object      = SRC_NOTIF_OBJ2,
Notification_Class              = SRC_NOTIF_CLS2
Event_Enable                    = {T, T, T}
```

Source notification class object SRC_NOTIF_CLS1 in DS configured as follows:

```
Notification Class              = (instance number of notification class object),
Priority                        = (any valid priority)
Ack_Required                    = (any valid Ack_Required value)
Recipient_List                  = {  (all),          -- Valid Days
                                     (all),          -- From Time, To Time
                                     IUT             -- Recipient
                                  }          -- One list element
Process Identifier              = SRC_PROCESS_ID
Issue Confirmed Notifications   = FALSE
Transitions                     = {T, T, T}
```

Source notification class object SRC_NOTIF_CLS2 in IUT configured as follows:

```
Notification Class              = (instance number of notification class object),
Priority                        = (any valid priority)
Ack_Required                    = (any valid Ack_Required value)
Recipient_List                  = {  (all)           -- Valid Days
                                     (all),          -- From Time, To Time
                                     IUT             -- Recipient
                                  }          -- One list element
Process Identifier              = SRC_PROCESS_ID
Issue Confirmed Notifications   = FALSE
Transitions                     = {T, T, T}
```

Note to Tester: Issue Confirmed Notifications is specified as FALSE within the test. Thus UnconfirmedEventNotification-Request messages from the Notification source object are expected. Behaviors can alternately be tested using Issue Confirmed Notifications set to TRUE resulting in ConfirmedEventNotification service messages from the Notification source

object that must be acknowledged at the communication level, but it is not necessary to test both settings unless specifically directed otherwise by the test.

Test Steps:

1. MAKE (Local_Forwarding_Only = FALSE)
2. TRANSMIT AddListElement-Request,
   'Object Identifier' = (the object being tested),
   'Property Identifier' = Subscribed_Recipients,
   'List of Elements' = {DEST_OBJ_ID, -- Recipient D1
   DEST_PROCESS_ID, -- Process Identifier
   FALSE, -- Issue Confirmed Notifications
   TR -- Time Remaining where TR > test duration
   } -- One list element
3. RECEIVE BACnet-SimpleACK-PDU
4. TRANSMIT SOURCE = DS, UnconfirmedEventNotification-Request,
   'Process Identifier' = SRC_PROCESS_ID,
   'Initiating Device Identifier' = SRC_NOTIF_DEV1, -- DS
   'Event Object Identifier' = SRC_NOTIF_OBJ1,
   'Time Stamp' = (any valid time stamp),
   'Notification Class' = SRC_NOTIF_CLS1,
   'Priority' = (any valid priority),
   'Event Type' = (any valid event type),
   'Message Text' = (optional, any valid message text),
   'Notify Type' = SRC_NOTIF_TYP,
   'AckRequired' = (any valid value), -- absent if Notify Type is ACK_NOTIFICATION
   'From State' = (any valid From_State), -- absent if Notify Type is ACK_NOTIFICATION
   'To State' = (any valid To_State),
   'Event Values' = (any valid event values) -- absent if Notify Type is ACK_NOTIFICATION
5. BEFORE ***Notification Fail Time***
   RECEIVE DESTINATION = D1, UnconfirmedEventNotification-Request
   'Process Identifier' = DEST_PROCESS_ID,
   (other parameter values match the parameter values used in the previous step)
6. MAKE (IUT Notification source object send an UnconfirmedEventNotification-Request,
   'Process Identifier' = SRC_PROCESS_ID,
   'Initiating Device Identifier' = SRC_NOTIF_DEV2, --IUT
   'Event Object Identifier' = SRC_NOTIF_OBJ2,
   'Time Stamp' = (any valid time stamp),
   'Notification Class' = SRC_NOTIF_CLS2,
   'Priority' = (any valid priority),
   'Event Type' = (any valid event type),
   'Message Text' = (optional, any valid message text),
   'Notify Type' = SRC_NOTIF_TYP,
   'AckRequired' = (any valid value), -- absent if Notify Type is ACK_NOTIFICATION
   'From State' = (any valid From_State), -- absent if Notify Type is ACK_NOTIFICATION
   'To State' = (any valid To_State),
   'Event Values' = (any valid event values) -- absent if Notify Type is ACK_NOTIFICATION)
7. BEFORE ***Notification Fail Time***
   RECEIVE DESTINATION = D1, UnconfirmedEventNotification-Request
   'Process Identifier' = DEST_PROCESS_ID,
   (other parameter values match the parameter values used in the previous step)

7.3.2.30.12 Preventing endless cycling / duplication of event forwarding for the same notification

7.3.2.30.12.1 Local Broadcast To Receiving Port Restriction Test

Purpose: Check that locally broadcast event forwarding is not allowed at the same port the source event notification is received at.

Test Concept: Network number NN is the BACnet network number the IUT is connected to at IUT device port PT. Set Local_Forwarding_Only to FALSE in the IUT. Set a recipient in the Subscribed_Recipients property and the Recipient_List property to the BACnetAddress for a local broadcast to network number NN. Configure the notification source to send an event notification to the IUT device at network NN. Monitoring the IUT shall be done at network number NN. Transmit the source event notification directly to the IUT device at network number NN. Check that the IUT did not forward the event notification.

If the IUT Notification Forwarder object Local_Forwarding_Only property value cannot be set to FALSE, then omit this test.

Configuration Requirements:

Begin with base setup 1 for Notification Forwarder object tests, perform the following additional steps:
Notification source object is in DS.
Source notification class is in DS.
Reconfigure the source notification class SRC_NOTIF_CLS in DS as follows:

```
Recipient_List = {   (all),                          -- Valid Days
                     (all),                          -- From Time, To Time
                     (IUT at network NN)             -- Recipient
                 }                          -- One list element
```

Test Steps:

1. MAKE (Local_Forwarding_Only = FALSE)
2. TRANSMIT AddListElement-Request,

```
'Object Identifier' =   (the object being tested),
'Property Identifier' = Subscribed_Recipients,
'List of Elements' =    {(BACnetAddress; network-number 00, mac-address length 0),    --Recipient
                         DEST_PROCESS_ID,     --Process Identifier
                         FALSE,               --Issue Confirmed Notifications
                         TR                   --Time Remaining where TR > test duration
                         }                    --One list element
```

3. RECEIVE BACnet-SimpleACK-PDU
4. MAKE (Recipient_List =

```
{(all),                                  -- Valid Days
(all),                                   -- From Time, To Time
(BACnetAddress; network-number NN, mac-address length 0),    --Recipient
DEST_PROCESS_ID,                         -- Process Identifier
FALSE,                                   -- Issue Confirmed Notifications
{T, T, T}                                -- Transitions
})              --One list element
```

5. TRANSMIT SOURCE = DS, UnconfirmedEventNotification-Request,

```
'Process Identifier'              = SRC_PROCESS_ID,
'Initiating Device Identifier'    = SRC_NOTIF_DEV,
'Event Object Identifier'         = SRC_NOTIF_OBJ,
'Time Stamp'                      = (any valid time stamp),
'Notification Class'              = SRC_NOTIF_CLS,
'Priority'                        = (any valid priority),
'Event Type'                      = (any valid event type),
'Message Text'                    = (optional, any valid message text),
'Notify Type'                     = SRC_NOTIF_TYP,
'AckRequired'                     = (any valid value),          -- absent if Notify Type is ACK_NOTIFICATION
'From State'                      = (any valid From_State),  -- absent if Notify Type is ACK_NOTIFICATION
'To State'                        = (any valid To_State),
'Event Values'                    = (any valid event values)  -- absent if Notify Type is ACK_NOTIFICATION
```

6. WAIT **Notification Fail Time**
7. CHECK (the IUT did not transmit an event notification)

7.3.2.30.12.2 Globally Broadcast Event Notification Received Restriction Test

Purpose: Verify global broadcast event notifications are not forwarded.

Test Concept: Set D1 as a recipient in the Subscribed_Recipients property. Send a globally broadcast event notification to the IUT and check that it is not forwarded.

Configuration Requirements:

Begin with base setup 2 for Notification Forwarder object tests with TR lifetime sufficient for this test.
Perform the following additional steps:
Reconfigure the source notification class SRC_NOTIF_CLS in DS (or IUT if Local_Forwarding_Only is TRUE) as follows:

```
Recipient_List = {  (all),                                  --Valid Days
                    (all),                                  --From Time, To Time
                    (BACnetAddress; network-number 65535, mac-address length 0)  --Recipient = global broadcast
                 }          --One list element
```

Test Steps:

1. TRANSMIT SOURCE = DS, UnconfirmedEventNotification-Request,

```
'Process Identifier'            = SRC_PROCESS_ID,
'Initiating Device Identifier'  = SRC_NOTIF_DEV,
'Event Object Identifier'       = SRC_NOTIF_OBJ,
'Time Stamp'                    = (any valid time stamp),
'Notification Class'            = SRC_NOTIF_CLS,
'Priority'                      = (any valid priority),
'Event Type'                    = (any valid event type),
'Message Text'                  = (optional, any valid message text),
'Notify Type'                   = SRC_NOTIF_TYP,
'AckRequired'                   = (any valid value),          -- absent if Notify Type is ACK_NOTIFICATION
'From State'                    = (any valid From_State),  -- absent if Notify Type is ACK_NOTIFICATION
'To State'                      = (any valid To_State),
'Event Values'                  = (any valid event values)  -- absent if Notify Type is ACK_NOTIFICATION
```

2. WAIT **Notification Fail Time**
3. CHECK (the IUT did not transmit an event notification)

7.3.2.30.12.3 Forwarding As Global Broadcast Restriction Test

Purpose: Verify forwarded event notifications cannot be globally broadcast by the IUT.

Test Concept: Attempt to configure an IUT Subscribed_Recipients value to forward an event as a global broadcast. Check that the configuration is either not accepted by the IUT or does not send a global broadcast when so configured. Attempt to configure an IUT Recipient_List value to forward an event as a global broadcast. Check that the configuration is either not accepted by the IUT or does not send a global broadcast when so configured.

Configuration Requirements:

Base setup 1 for Notification Forwarder object tests.

Test Steps:

1. TRANSMIT AddListElement-Request,

```
'Object Identifier' =   (the object being tested),
```

'Property Identifier' = Subscribed_Recipients,
'List of Elements' = {(BACnetAddress; network-number 65535, mac-address length 0), --Recipient = global broadcast
DEST_PROCESS_ID, -- Process Identifier
FALSE, -- Issue Confirmed Notifications
TR -- Time Remaining where TR > test duration
} -- One list element

2. IF (RECEIVE BACnet-SimpleACK-PDU) THEN
   TRANSMIT SOURCE = DS, UnconfirmedEventNotification-Request,
   'Process Identifier' = SRC_PROCESS_ID,
   'Initiating Device Identifier' = SRC_NOTIF_DEV,
   'Event Object Identifier' = SRC_NOTIF_OBJ,
   'Time Stamp' = (any valid time stamp),
   'Notification Class' = SRC_NOTIF_CLS,
   'Priority' = (any valid priority),
   'Event Type' = (any valid event type),
   'Message Text' = (optional, any valid message text),
   'Notify Type' = SRC_NOTIF_TYP,
   'AckRequired' = (any valid value) -- absent if Notify Type is ACK_NOTIFICATION
   'From State' = (any valid From_State) -- absent if Notify Type is ACK_NOTIFICATION
   'To State' = (any valid To_State),
   'Event Values' = (any valid event values) -- absent if Notify Type is ACK_NOTIFICATION
3. WAIT **Notification Fail Time**
4. CHECK (the IUT did not transmit an event notification)
5. If (The IUT can contain a global broadcast address in the Recipient_List) THEN
   MAKE (Recipient_List = {(all), --Valid Days
   (all), --From Time, To Time
   (BACnetAddress; network-number 65535, mac-address length 0), --Recipient = global broadcast
   DEST_PROCESS_ID, -- Process Identifier
   FALSE, -- Issue Confirmed Notifications
   {T, T, T} -- Transitions
   } -- One list element
   TRANSMIT SOURCE = DS, UnconfirmedEventNotification-Request,
   'Process Identifier' = SRC_PROCESS_ID,
   'Initiating Device Identifier' = SRC_NOTIF_DEV,
   'Event Object Identifier' = SRC_NOTIF_OBJ,
   'Time Stamp' = (any valid time stamp),
   'Notification Class' = SRC_NOTIF_CLS,
   'Priority' = (any valid priority),
   'Event Type' = (any valid event type),
   'Message Text' = (optional, any valid message text),
   'Notify Type' = SRC_NOTIF_TYP,
   'AckRequired' = (any valid value), -- absent if Notify Type is ACK_NOTIFICATION
   'From State' = (any valid From_State), -- absent if Notify Type is ACK_NOTIFICATION
   'To State' = (any valid To_State),
   'Event Values' = (any valid event values) -- absent if Notify Type is ACK_NOTIFICATION
   WAIT **Notification Fail Time**
   CHECK (the IUT did not transmit an event notification)

7.3.2.30.12.4 Directed Broadcast Received Forwarding To BACnetAddress Restriction Test

Purpose: Check that a directed broadcast event notification received at a particular port is not forwarded to an external device identified as a BACnetRecipient using the BACnetAddress choice when it is residing on the same network that the port is attached to.

Test Concept: Network number NN is the BACnet network number the IUT is connected to. Configure the notification source to send an event notification as a directed broadcast to Network number NN. Set Local_Forwarding_Only to FALSE, and set an IUT recipient in the Subscribed_Recipients property to the BACnetAddress of a device connected to network number NN. Monitoring the IUT shall be done at network number NN. Transmit the source event notification as a directed broadcast to network number NN. The event notification source may be on a local or remote network. Check that the IUT did not forward the event notification.
Set an IUT recipient in the Recipient_List property to the BACnetAddress of a device connected to network number NN. Transmit the source event notification as a directed broadcast to network number NN. Check that the IUT did not forward the event notification.

If the IUT Notification Forwarder object Local_Forwarding_Only property value cannot be set to FALSE, then omit this test.

Configuration Requirements:

Begin with base setup 1 for Notification Forwarder object tests, perform the following additional steps:
Notification source object must be in DS.
Source notification class must be in DS.
Reconfigure the source notification class SRC_NOTIF_CLS in DS as follows:

```
Recipient_List = {  (all),                              -- Valid Days
                    (all),                              -- From Time, To Time
                    (BACnetAddress; network-number NN, mac-address length 0)
                                                        -- Recipient is directed broadcast
                 }              --One list element
```

Test Steps:

1. MAKE (Local_Forwarding_Only = FALSE)
2. TRANSMIT AddListElement-Request,

```
'Object Identifier' =    (the object being tested),
'Property Identifier' =  Subscribed_Recipients,
'List of Elements' =     {(BACnetAddress; network-number NN, mac-address MA),    --Recipient
                         DEST_PROCESS_ID,     -- Process Identifier
                         FALSE,               -- Issue Confirmed Notifications
                         TR                   -- Time Remaining where TR > test duration
                         }                    -- One list element
```

3. RECEIVE BACnet-SimpleACK-PDU
4. TRANSMIT SOURCE = DS, UnconfirmedEventNotification-Request,

```
'Process Identifier'              = SRC_PROCESS_ID,
'Initiating Device Identifier'    = SRC_NOTIF_DEV,
'Event Object Identifier'         = SRC_NOTIF_OBJ,
'Time Stamp'                      = (any valid time stamp),
'Notification Class'              = SRC_NOTIF_CLS,
'Priority'                        = (any valid priority),
'Event Type'                      = (any valid event type),
'Message Text'                    = (optional, any valid message text),
'Notify Type'                     = SRC_NOTIF_TYP,
'AckRequired'                     = (any valid value),          -- absent if Notify Type is ACK_NOTIFICATION
'From State'                      = (any valid From_State),  -- absent if Notify Type is ACK_NOTIFICATION
'To State'                        = (any valid To_State),
'Event Values'                    = (any valid event values)  -- absent if Notify Type is ACK_NOTIFICATION
```

5. WAIT **Notification Fail Time**
6. CHECK (the IUT did not transmit an event notification)
7. MAKE (Recipient_List = {(all), --Valid Days

```
                          (all),                        -- From Time, To Time
                          (BACnetAddress; network-number NN, mac-address MA),     -- Recipient
```

DEST_PROCESS_ID, -- Process Identifier
FALSE, -- Issue Confirmed Notifications
{T, T, T} -- Transitions
} -- One list element

8. TRANSMIT SOURCE = DS, UnconfirmedEventNotification-Request,
'Process Identifier' = SRC_PROCESS_ID,
'Initiating Device Identifier' = SRC_NOTIF_DEV,
'Event Object Identifier' = SRC_NOTIF_OBJ,
'Time Stamp' = (any valid time stamp),
'Notification Class' = SRC_NOTIF_CLS,
'Priority' = (any valid priority),
'Event Type' = (any valid event type),
'Message Text' = (optional, any valid message text),
'Notify Type' = SRC_NOTIF_TYP,
'AckRequired' = (any valid value), -- absent if Notify Type is ACK_NOTIFICATION
'From State' = (any valid From_State), -- absent if Notify Type is ACK_NOTIFICATION
'To State' = (any valid To_State),
'Event Values' = (any valid event values) -- absent if Notify Type is ACK_NOTIFICATION
9. WAIT **Notification Fail Time**
10. CHECK (the IUT did not transmit an event notification)

7.3.2.30.12.5 Directed Broadcast Received Forwarding To Object Identifier Restriction Test

Purpose: Check that a directed broadcast event notification received at a particular port is not forwarded to an external device identified as a BACnetRecipient using the BACnetObjectIdentifier choice when it is residing on the same network that the port is attached to.

Test Concept: Network number NN is the BACnet network number the IUT is connected to. Configure the notification source DS to send an event notification as a directed broadcast to Network number NN. Set Local_Forwarding_Only to FALSE, and set recipient D1 in the IUT Subscribed_Recipients property to the BACnetObjectIdentifier of a device connected to network number NN. This device may be the TD or a reference device, but it must be capable of being discovered by the IUT device. Monitoring the IUT shall be done at network number NN. Transmit the source event notification as a directed broadcast. The event notification source may be on a local or remote network. Check that the IUT did not forward the event notification. Set a recipient in the IUT Recipient_List property to the BACnetObjectIdentifier of a device connected to network number NN. Transmit the source event notification as a directed broadcast. Check that the IUT did not forward the event notification.

If the IUT Notification Forwarder object Local_Forwarding_Only property value cannot be set to FALSE, then omit this test.

Configuration Requirements:

Begin with base setup 1 for Notification Forwarder object tests, perform the following additional steps:
Reconfigure the source notification class SRC_NOTIF_CLS in DS as follows:

Recipient_List = {(all), -- Valid Days
(all), -- From Time, To Time
(BACnetAddress; network-number NN (-- 0 if local broadcast), mac-address length 0)
-- Recipient is local broadcast
} -- One list element

Test Steps:

1. MAKE (Local_Forwarding_Only = FALSE)
2. TRANSMIT AddListElement-Request,
'Object Identifier' = (the object being tested),
'Property Identifier' = Subscribed_Recipients,
'List of Elements' = {DEST_OBJ_ID, -- Recipient D1

DEST_PROCESS_ID, -- Process Identifier
FALSE, -- Issue Confirmed Notifications
TR -- Time Remaining where TR > test duration
} -- One list element

3. RECEIVE BACnet-SimpleACK-PDU
4. TRANSMIT SOURCE = DS, UnconfirmedEventNotification-Request,
'Process Identifier' = SRC_PROCESS_ID,
'Initiating Device Identifier' = SRC_NOTIF_DEV,
'Event Object Identifier' = SRC_NOTIF_OBJ,
'Time Stamp' = (any valid time stamp),
'Notification Class' = SRC_NOTIF_CLS,
'Priority' = (any valid priority),
'Event Type' = (any valid event type),
'Message Text' = (optional, any valid message text),
'Notify Type' = SRC_NOTIF_TYP,
'AckRequired' = (any valid value), -- absent if Notify Type is ACK_NOTIFICATION
'From State' = (any valid From_State), -- absent if Notify Type is ACK_NOTIFICATION
'To State' = (any valid To_State),
'Event Values' = (any valid event values) -- absent if Notify Type is ACK_NOTIFICATION
5. WAIT **Notification Fail Time**
6. CHECK (the IUT did not transmit an event notification)
7. MAKE (Recipient_List = {(all), -- Valid Days
(all), -- From Time, To Time
DEST_OBJ_ID, -- Recipient D1
DEST_PROCESS_ID, -- Process Identifier
FALSE, -- Issue Confirmed Notifications
{T, T, T} -- Transitions
}) -- One list element
8. TRANSMIT SOURCE = DS, UnconfirmedEventNotification-Request,
'Process Identifier' = SRC_PROCESS_ID,
'Initiating Device Identifier' = SRC_NOTIF_DEV,
'Event Object Identifier' = SRC_NOTIF_OBJ,
'Time Stamp' = (any valid time stamp),
'Notification Class' = SRC_NOTIF_CLS,
'Priority' = (any valid priority),
'Event Type' = (any valid event type),
'Message Text' = (optional, any valid message text),
'Notify Type' = SRC_NOTIF_TYP,
'AckRequired' = (any valid value), -- absent if Notify Type is ACK_NOTIFICATION
'From State' = (any valid From_State), -- absent if Notify Type is ACK_NOTIFICATION
'To State' = (any valid To_State),
'Event Values' = (any valid event values) -- absent if Notify Type is ACK_NOTIFICATION
9. WAIT **Notification Fail Time**
10. CHECK (the IUT did not transmit an event notification)

7.3.2.30.12.6 Port Restriction Test

Purpose: Check that event notifications received through ports that are not enabled are not forwarded.

Test Concept: Perform test **7.3.2.30.10 Port_Filter Test**.

7.3.2.30.13 Persistence Tests

7.3.2.30.13.1 Recipient_List Persistence Test

Purpose: This test ensures that Recipient_List property value is maintained through a device “restart”.

Test Concept: Initialize the Recipient_List property with a known value, then cycle power or restart the IUT and verify the Recipient_List property value is maintained.

Configuration Requirements: Base setup 1 for Notification Forwarder object tests.

Test Steps:

1. MAKE (Recipient_List = {(all), -- Valid Days
(all), -- From Time, To Time
DEST_OBJ_ID, -- Recipient D1
DEST_PROCESS_ID, -- Any Process Identifier
DEST_CONF_NOTIF, -- Any Issue Confirmed Notifications
{T, T, T} -- Transitions
}) -- One list element
2. IF (IUT supports the ReinitializeDevice service) THEN {
TRANSMIT ReinitializeDevice-Request,
'Reinitialized State of Device' = WARMSTART,
'Password' = (any valid password) --if required by IUT
RECEIVE BACnet-SimpleACK-PDU
}
3. CHECK (Did the IUT perform a WARMSTART restart)
4. WAIT for restart to complete
5. VERIFY Recipient_List = {(all), -- Valid Days
(all), -- From Time, To Time
DEST_OBJ_ID, -- Recipient D1
DEST_PROCESS_ID, -- Process Identifier
DEST_CONF_NOTIF, -- Issue Confirmed Notifications
{T, T, T} -- Transitions
} -- One list element
6. MAKE (The IUT reset by cycling power)

7. WAIT for restart to complete
8. VERIFY Recipient_List = {(all), -- Valid Days
(all), -- From Time, To Time
DEST_OBJ_ID, -- Recipient D1
DEST_PROCESS_ID, -- Process Identifier
DEST_CONF_NOTIF, -- Issue Confirmed Notifications
{T, T, T} -- Transitions
} -- One list element

7.3.2.30.13.2 Subscribed_Recipients Persistence Test

Purpose: This test ensures that Subscribed_Recipients property values are maintained through a device “restart”.

Test Concept: Initialize the Subscribed_Recipients property with a known value, then cycle power to restart the IUT and verify the Subscribed_Recipients property value is maintained.

Configuration Requirements: Base setup 2 for Notification Forwarder object tests with TR lifetime sufficient for this test.

Test Steps:

1. IF (IUT supports the ReinitializeDevice service) THEN {
TRANSMIT ReinitializeDevice-Request,
'Reinitialized State of Device' = WARMSTART,
'Password' = (any valid password) --if required by IUT
RECEIVE BACnet-SimpleACK-PDU
}

2. CHECK (Did the IUT perform a WARMSTART restart)
3. WAIT for restart to complete, making a note of the time to restart as Start_Up_TimeA
4. VERIFY Subscribed_Recipients =
   {DEST_OBJ_ID, -- Recipient D1
   DEST_PROCESS_ID, -- Process Identifier
   DEST_CONF_NOTIF, -- Issue Confirmed Notifications
   TR1 -- Time Remaining where (TR–Start_Up_TimeA-1) <= TR1 <= TR
   } -- One list element
5. MAKE (The IUT reset by cycling power)
6. WAIT for restart to complete, making a note of the time to restart as Start_Up_TimeB
7. VERIFY Subscribed_Recipients =
   {DEST_OBJ_ID, -- Recipient D1
   DEST_PROCESS_ID, -- Process Identifier
   DEST_CONF_NOTIF, -- Issue Confirmed Notifications
   TR1 -- Time Remaining where (TR–Start_Up_TimeA-Start_Up_TimeB-2) <= TR1 <= TR
   } -- One list element

Note To Tester: Start_Up_TimeA or Start_Up_TimeB is the time in minutes required to restart the IUT.

7.3.2.30.14 Capacity And Range Tests

7.3.2.30.14.1 Time Remaining Range Test

Purpose: Test that the Subscribed_Recipients property can be written with the range of Time Remaining values required.

Test Concept: Add a BACnetEventNotificationSubscription to the Notification Forwarder Subscribed_Recipients property with the largest Time Remaining value required. Verify the new BACnetEventNotificationSubscription entry is added.

Configuration Requirements: Base setup 1 for Notification Forwarder object tests.

Test Steps:

1. TRANSMIT AddListElement-Request,
   'Object Identifier' = (the object being tested),
   'Property Identifier' = Subscribed_Recipients,
   'List of Elements' = {DEST_OBJ_ID, -- Recipient D1
   DEST_PROCESS_ID, -- Process Identifier
   DEST_CONF_NOTIF, -- Issue Confirmed Notifications
   1440 minutes -- Time Remaining
   } -- One list element
2. RECEIVE BACnet-SimpleACK-PDU
3. READ SR = Subscribed_Recipients
4. CHECK (SR = {DEST_OBJ_ID, -- Recipient D1
   DEST_PROCESS_ID, -- Process Identifier
   DEST_CONF_NOTIF, -- Issue Confirmed Notifications
   TR1 -- Time Remaining where 1439 <= TR1 <= 1440
   }) -- One list element

7.3.2.30.14.2 Recipient Capacity Test

Purpose: Test that the capacity of property Recipient_List and property Subscribed_Recipients is sufficient.

Test Concept: Add entries to the Recipient_List and Subscribed_Recipients properties to verify at least 8 of each can be added at the same time. Use different Process Identifier values and recipients for each list element to ensure each one is added separately.

Configuration Requirements: Base setup 1 for Notification Forwarder object tests.

Test Steps:

1. MAKE (Recipient_List = {--list with 8 elements
   {--for each element
   (all), -- Valid Days
   (all), -- From Time, To Time
   DEST_OBJ_IDX, -- One of 8 Recipients DX where X = 1 to 8
   (DEST_PROCESS_ID + X), -- One of 8 Process Identifiers where X = 1 to 8
   DEST_CONF_NOTIF, -- Any Issue Confirmed Notifications
   {T, T, T} -- Transitions
   }
   { … } --Seven additional list elements as above
   })
2. REPEAT X = (values from 1 to 8) DO
   {TRANSMIT AddListElement-Request,
   'Object Identifier' = (the object being tested),
   'Property Identifier' = Subscribed_Recipients,
   'List of Elements' = {DEST_OBJ_ID2X, -- One of 8 Recipients DX where X = 1 to 8
   (DEST_PROCESS_ID2 + X), -- One of 8 Process Identifiers where X = 1 to 8
   DEST_CONF_NOTIF, -- Any Issue Confirmed Notifications
   TR -- Any Time Remaining where TR > test duration
   } -- One list element each time
   RECEIVE BACnet-SimpleACK-PDU
   }
3. VERIFY Subscribed_Recipients = { … } -- List with 8 elements as configured above, except for Time Remaining
4. VERIFY Recipient_List = { … } -- List with 8 elements as configured above

[Add new entry to **HISTORY OF REVISIONS**, p. 662]

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

HISTORY OF REVISIONS

Summary of Changes to the Standard
. . .
Addendum *q* to ANSI/ASHRAE 135.1-2013 Approved by ASHRAE and the American National Standards Institute on December 7, 2018. 1. Update alarm and event tests for protocol revisions 13 and higher

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.

ASHRAE · 1791 Tullie Circle NE · Atlanta, GA 30329 · www.ashrae.org

About ASHRAE

ASHRAE, founded in 1894, is a global society advancing human well-being through sustainable technology for the built environment. The Society and its members focus on building systems, energy efficiency, indoor air quality, refrigeration, and sustainability. Through research, Standards writing, publishing, certification and continuing education, ASHRAE shapes tomorrow's built environment today.

For more information or to become a member of ASHRAE, visit www.ashrae.org.

To stay current with this and other ASHRAE Standards and Guidelines, visit www.ashrae.org/standards.

Visit the ASHRAE Bookstore

ASHRAE offers its Standards and Guidelines in print, as immediately downloadable PDFs, on CD-ROM, and via ASHRAE Digital Collections, which provides online access with automatic updates as well as historical versions of publications. Selected Standards and Guidelines are also offered in redline versions that indicate the changes made between the active Standard or Guideline and its previous version. For more information, visit the Standards and Guidelines section of the ASHRAE Bookstore at www.ashrae.org/bookstore.

IMPORTANT NOTICES ABOUT THIS STANDARD

To ensure that you have all of the approved addenda, errata, and interpretations for this Standard, visit www.ashrae.org/standards to download them free of charge.

Addenda, errata, and interpretations for ASHRAE Standards and Guidelines are no longer distributed with copies of the Standards and Guidelines. ASHRAE provides these addenda, errata, and interpretations only in electronic form to promote more sustainable use of resources.

12/18