ADDENDA

ANSI/ASHRAE Addendum l to
ANSI/ASHRAE Standard 135.1-2009

Method of Test for Conformance to BACnet®

Approved by the ASHRAE Standards Committee on January 21, 2012; by the ASHRAE Board of Directors on January 25, 2012; and by the American National Standards Institute on January 26, 2012.

© 2012 ASHRAE ISSN 1041-2336

Approved American National Standard
ANSI
www.ansi.org

SPECIAL NOTE

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

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

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

The Manager of Standards of ASHRAE should be contacted for:

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

DISCLAIMER

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

ASHRAE INDUSTRIAL ADVERTISING POLICY ON STANDARDS

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

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

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

FOREWORD

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

135.1-2009l-1. Generalize the Notify_Type Test, p. 1.
135.1-2009l-2. Add in a Test for Resizable Array Properties, p. 3.
135.1-2009l-3. Correct the Usage of VERIFY vs. CHECK in the Record_Count Test, p. 4.
135.1-2009l-4. Correct the Trend Log COV Subscription Failure Test, p. 5.
135.1-2009l-5. Remove the Testing Requirement That Status_Flags be Changeable, p. 6.
135.1-2009l-6. Allow WritePropertyMultiple Tests to be Applied to Array Properties, p. 8.
135.1-2009l-7. Modify Event Notifications Tests to Allow Use of Event Enrollment Objects, p. 10.
135.1-2009l-8. Add a Test for Acknowledging Offnormal Events, p. 23.
135.1-2009l-9. Update Expected Error Codes Negative COV Tests, p. 24.
135.1-2009l-10. Correct the Use of WAIT vs. BEFORE in COV Notification Tests, p. 28.
135.1-2009l-11. Improve Reading Multiple Properties with Multiple Embedded Access Errors Test, p. 34.
135.1-2009l-12. Expand Allowable Errors for Older Product When Reading Array Properties, p. 35.
135.1-2009l-13. Improve the Basic DeviceCommunicationControl Tests, p. 37.
135.1-2009l-14. Add Alarm Summarization Tests, p. 40.
135.1-2009l-15. Add Event Log Tests, p. 43.
135.1-2009l-16. Add Structured View Tests, p. 48.
135.1-2009l-17. Correct AddListElement Test, p. 50.
135.1-2009l-18. Add ReadRange Test, p. 51.
135.1-2009l-19. Remove Reliance on EPICS from DCC Test, p. 53.
135.1-2009l-20. Add Who-Has Tests, p. 54.
135.1-2009l-21. Correct Unknown Network Layer Message Type For Someone Else Test, p. 56.
135.1-2009l-22. Clarify TRANSMIT And RECEIVE Addressing Information, p. 58.

In the following document, language to be added to existing clauses of ANSI/ASHRAE 135.1-2009 and Addenda is indicated through the use of *italics*, while deletions are indicated by ~~strikethrough~~. Where entirely new subclauses are proposed to be added, plain type is used throughout. Only this new and deleted text is open to comment at this time. All other material in this addendum is provided for context only and is not open for public review comment except as it relates to the proposed changes.

135.1-2009l-1. Generalize the Notify_Type Test.

Rationale

The existing Notify_Type test does not take into account objects that do not generate TO-OFFNORMAL transitions. The test is modified to test transitions to/from any event states.

[Change **Clause 7.3.1.12**, p. 45]

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 ~~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.

Configuration Requirements: The IUT shall be configured with two event-generation objects, E_1 and E_2. Object E1 shall be configured with a Notify_Type of ALARM and E2 shall be configured with a Notify_Type of EVENT. ~~Both objects shall be in a NORMAL Event_State at the beginning of the test. 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 value of the Transitions parameter for all recipients shall be (TRUE, TRUE, TRUE).

In the test description below X_1 and X_2 are used to designate the event-triggering property linked to E_1 and E_2 respectively.

Test Steps:

[The test steps have been renumbered without showing the change marking for clarity.]

```
1.  VERIFY (E₁), Event_State = NORMAL
2.  VERIFY (E₂), Event_State = NORMAL
3.  WAIT (Time_Delay + **Notification Fail Time**)
4.  IF (X₁ is writable) THEN
        WRITE X₁ = (a value that ~~is OFFNORMAL~~ *will cause a transition in E₁*)
    ELSE
        MAKE (X₁ a value that ~~is OFFNORMAL~~ *will cause a transition in E₁*)
5.  *IF (the transition is not a FAULT transition)*
        WAIT (Time_Delay)
6.  BEFORE **Notification Fail Time**
        RECEIVE ConfirmedEventNotification-Request,
            'Process Identifier' =             (any valid process ID),
            'Initiating Device Identifier' =   IUT,
            'Event Object Identifier' =        (E₁),
            'Time Stamp' =                     (the current local time),
            'Notification Class' =             (the class corresponding to the object being tested),
            'Priority' =                       (*any valid value*~~the value configured to correspond to a TO-OFFNORMAL transition~~),
            'Event Type' =                     (any valid event type),
            'Notify Type' =                 ALARM,
            'AckRequired' =                 TRUE | FALSE,
            'From State' =                  NORMAL,
            'To State' =                    ~~OFFNORMAL~~*(any valid value)*,
            'Event Values' =                (values appropriate to the event type)
*7.*  *TRANSMIT SimpleAck-PDU*
8.  IF ($X_2$ is writable) THEN
        WRITE $X_2$ = (a value that ~~is OFFNORMAL~~ *will cause a transition in $E_1$*)
    ELSE
        MAKE ($X_2$ a value that ~~is OFFNORMAL~~ *will cause a transition in $E_1$*)
*9.*  *IF (the transition is not a FAULT transition) THEN*
        WAIT (Time_Delay)
*10.* BEFORE **Notification Fail Time**
        RECEIVE ConfirmedEventNotification-Request,
            'Process Identifier' =          (any valid process ID),
            'Initiating Device Identifier' = IUT,
            'Event Object Identifier' =     ($E_2$),
            'Time Stamp' =                  (the current local time),
            'Notification Class' =          (the class corresponding to the object being tested),
            'Priority' =                    (*any valid value*~~the value configured to correspond to a TO-OFFNORMAL transition~~),
            'Event Type' =                  (any valid event type),
            'Notify Type' =                 EVENT,
            'AckRequired' =                 TRUE | FALSE,
            'From State' =                  ~~NORMAL~~*Event_State*,
            'To State' =                    ~~OFFNORMAL~~*(any valid value)*,
            'Event Values' =                (values appropriate to the event type)
*11.* *TRANSMIT 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.

135.1-2009l-2. Add in a Test for Resizable Array Properties.

Rationale

There is no existing test that verifies the requirements placed on resizable arrays.

[Add new **Clause 7.3.1.X3**, p. 49]

7.3.1.X3 Array Resizing Test

The test in this clause shall be applied to resizable arrays in devices claiming Protocol_Revision 4 or higher. It may be applied to resizable arrays in devices claiming Protocol_Revision 3 or lower, but only where conformance to the rules on resizing arrays of Protocol_Revision 4 is claimed.

Dependencies: None

BACnet Reference Clause: 12.

Purpose: To verify that resizable arrays are resized in accordance with the rules added in Protocol_Revision 4.

Test Concept: The array is written as a whole to set it to a non-zero size. It is then resized smaller and larger by writing the entire array. It is then resized smaller and larger by writing to element number zero. An attempt is made to increase it with an invalid write. After each operation, the array size and array contents are checked. Finally, if it can be resized to have zero elements, it is then written to size zero. If possible, all elements in the arrays should be distinguishable from each other and across write operations.

Test Steps:

```
1.  WRITE (the array property being tested) = (array of non-zero size $N_1$)
2.  VERIFY (array is as written in step 1)
3.  WRITE (the array property being tested) = (array of non-zero size $N_2$, where $N_2 \leq N_1$)
4.  VERIFY (array is as written in step 3)
5.  WRITE (the array property being tested) = (array of non-zero size $N_3$, where $N_3 \geq N_1$)
6.  VERIFY (array is as written in step 5)
7.  WRITE (the array property being tested) = (a non-zero unsigned value $N_4$, where $N_4 \leq N_1$), ARRAY_INDEX = 0
8.  VERIFY (array contains first $N_4$ elements of the array written in step 5)
9.  WRITE (the array property being tested) = ($N_5$, where $N_5 \geq N_4$), ARRAY_INDEX = 0
10. VERIFY (array contains first $N_4$ elements of the array written in step 5, plus $N_5 - N_4$
        additional elements, initialized to particular values if specified for the array property
        being tested)
11. TRANSMIT WriteProperty-Request,
        'Object Identifier' =       (the object being tested),
        'Property Identifier' =     (the array property being tested),
        'Property Array Index' =    ($N_6$, where $N_6 \geq N_5$),
        'Property Value' =          (one array element)
12. RECEIVE BACnet-Error-PDU
        Error Class  =              PROPERTY,
        Error Code  =               INVALID_ARRAY_INDEX
13. VERIFY (array is unchanged from step 10)
14. IF (the array can be resized to have zero elements) THEN
        WRITE (the array property being tested) = (empty array)
        VERIFY (array is empty)
```

135.1-2009l-3. Correct the Usage of VERIFY vs. CHECK in the Record_Count Test.

Rationale

The test incorrectly uses the VERIFY statement instead of the CHECK statement.

[Change **Clause 7.3.2.24.8**, p. 100]

7.3.2.24.8 Record_Count Test

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

BACnet Reference Clause: 12.25.15.

Purpose: To verify that the Record_Count property indicates the number of records that are stored in the Log_Buffer.

Test Concept: The Trend Log is configured to acquire data by whatever means. Record_Count is set to zero and Log_Buffer is read to verify no records are present. Collection of data proceeds until Record_Count is about Buffer_Size/2, collection is halted and Log_Buffer is read to verify the Record_Count value. Collection then resumes until Buffer_Size records are read; collection is then halted and Log_Buffer read to verify Record_Count 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. Log_Enable shall be set to FALSE.

Test Steps:

1. WRITE Record_Count = 0
2. WAIT **Internal Processing Fail Time**
3. CHECK (that Log_Buffer has no records)
4. WRITE Log_Enable = TRUE
5. WHILE (Record_Count < Buffer_Size/2) DO { }
6. WRITE Log_Enable = FALSE
7. WAIT **Internal Processing Fail Time**
8. ~~VERIFY~~ *CHECK* (that Log_Buffer has the number of records indicated by Record_Count)
9. WRITE Log_Enable = TRUE
10. WHILE (Record_Count < Buffer_Size) DO { }
11. WRITE Log_Enable = FALSE
12. WAIT **Internal Processing Fail Time**
13. ~~VERIFY~~ *CHECK* (that Log_Buffer has the number of records indicated by Record_Count)

135.1-2009l-4. Correct the Trend Log COV Subscription Failure Test.

Rationale

Change the test to not require that a COV subscription be initiated as soon as the logging object is enabled.

[Change **Clause 7.3.2.24.12**, p. 104]

7.3.2.24.12 COV Subscription Failure Test

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

BACnet Reference Clauses: 12.25.5, 12.25.9, and 12.25.10.

Purpose: To verify that a failed COV subscription causes a TO-FAULT transition.

Test Concept: ~~The Trend Log is configured to acquire data by COV subscription from the TD. After it attempts to subscribe with the TD the Trend Log is halted and Event_State is checked.~~*A Trend Log configured to acquire samples via COV is monitored to ensure that when a COV subscription fails, the object will go into fault.*

Configuration Requirements: *The Trend Log is configured to acquire data by COV subscription from the TD.* 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. Log_Enable shall be set to *TRUE*~~FALSE~~.

Test Steps:

[The test steps have been renumbered without showing the change marking for clarity.]

```
~~1. VERIFY Event_State = NORMAL~~
~~2. WRITE Log_Enable = TRUE~~

1. IF (the IUT uses SubscribeCOV for this Trend Log)
      BEFORE the lifetime of the COV subscription expires
            RECEIVE  SubscribeCOV-Request,
                  'Subscriber Process Identifier' =      (any value),
                  'Monitored Object Identifier' =        (any object),
                  'Issue Confirmed Notifications' =      (TRUE|FALSE),
                  'Lifetime' =                           (2 * COV_Resubscription_Interval)
   ELSE
      BEFORE the lifetime of the COV subscription expires
            RECEIVE  SubscribeCOVProperty-Request,
                  'Subscriber Process Identifier' =      (any value),
                  'Monitored Object Identifier' =        (any object),
                  'Issue Confirmed Notifications' =      (TRUE|FALSE),
                  'Lifetime' =                           (2 * COV_Resubscription_Interval),
                  'Monitored Property Identifier' =      (the property to be monitored),
                  'COV Increment' =                      (Client_COV_Increment -- optional)
~~4. TRANSMIT BACnet-SimpleACK-PDU~~
~~5. WAIT COV_Resubscription_Interval~~
2. TRANSMIT BACnet-Error-PDU
            Error Class  =                  (any valid error class),
            Error Code  =                   (any valid error code)
3. VERIFY Event_State = FAULT
```

135.1-2009l-5. Remove the Testing Requirement That Status_Flags be Changeable.

Rationale

Change alarming tests to not require that Status_Flags be changeable.

[Change **Clause 8.2.2**, p. 108]

8.2.2 Change of Value Notification from an Analog Input, Analog Output, and Analog Value Object Status_Flags Property

Purpose: To verify that the IUT can initiate ConfirmedCOVNotification service requests, conveying a change of the Status_Flags property of Analog Input, Analog Output, and Analog Value objects.

Test Concept: A subscription for COV notifications is established. The Status_Flags property of the monitored object is then changed and a notification shall be received. The value of the Status-Flags property can be changed by using the WriteProperty service or by another means. For some implementations writing to the Out_Of_Service property will accomplish this task. For implementations where it is not possible to write to Status_Flags or Out_Of_Service *or change the Status_Flags by any other means, this test shall be skipped.*~~, the vendor shall provide an alternative trigger mechanism to accomplish this task. All of these methods are equally acceptable.~~

[Change **Clause 8.2.4**, p. 110]

8.2.4 Change of Value Notification from a Binary Input, Binary Output, and Binary Value Object Status_Flags Property

Purpose: To verify that the IUT can initiate ConfirmedCOVNotification service requests, conveying a change of the Status_Flags property of Binary Input, Binary Output, and Binary Value objects.

Test Concept: A subscription for COV notifications is established. The Status_Flags property of the monitored object is then changed and a notification shall be received. The value of the Status-Flags property can be changed by using the WriteProperty service or by another means. For some implementations writing to the Out_Of_Service property will accomplish this task. For implementations where it is not possible to write to Status_Flags or Out_Of_Service *or change the Status_Flags by any other means, this test shall be skipped.*~~, the vendor shall provide an alternative trigger mechanism to accomplish this task. All of these methods are equally acceptable.~~

[Change **Clause 8.2.6**, p. 113]

8.2.6 Change of Value Notification from a Multi-state Input, Multi-state Output Multi-state Value, Life Safety Point, and Life Safety Zone Object Status_Flags Property

Purpose: To verify that the IUT can initiate ConfirmedCOVNotification service requests, conveying a change of the Status_Flags property of Multi-state Input, Multi-state Output, Multi-state Value, Life Safety Point, and Life Safety Zone objects.

Test Concept: A subscription for COV notifications is established. The Status_Flags property of the monitored object is then changed and a notification shall be received. The value of the Status-Flags property can be changed by using the WriteProperty service or by another means. For some implementations writing to the Out_Of_Service property will accomplish this task. For implementations where it is not possible to write to Status_Flags or Out_Of_Service *or change the Status_Flags by any other means, this test shall be skipped.*~~, the vendor shall provide an alternative trigger mechanism to accomplish this task. All of these methods are equally acceptable.~~

[Change **Clause 8.2.8**, p. 115]

8.2.8 Change of Value Notification from a Loop Object Status_Flags Property

Purpose: To verify that the IUT can initiate ConfirmedCOVNotification service requests, conveying a change of the Status_Flags property of a Loop object.

Test Concept: A subscription for COV notifications is established. The Status_Flags property of the monitored object is then changed and a notification shall be received. The value of the Status-Flags property can be changed by using the WriteProperty service or by another means. For some implementations writing to the Out_Of_Service property will accomplish this task. For implementations where it is not possible to write to Status_Flags or Out_Of_Service *or change the Status_Flags by any other means, this test shall be skipped.*~~, the vendor shall provide an alternative trigger mechanism to accomplish this task. All of these methods are equally acceptable.~~

135.1-2009l-6. Allow WritePropertyMultiple Tests to be Applied to Array Properties.

Rationale

Most WritePropertyMultiple tests are written for non-array properties. These changes allow those tests to be applied to array properties.

[Change **Clauses 8.23.1** through **8.23.4**, p. 169]

8.23.1 Writing a Single Property of a Single Object

Purpose: To verify that the IUT can correctly initiate a WritePropertyMultiple service request containing a single property of a single object. If the IUT cannot be configured to initiate a WritePropertyMultiple service request containing a single property of a single object, then this test shall be omitted.

Test Steps:

1. RECEIVE WritePropertyMultiple-Request,
   'Object Identifier' = (any valid object identifier),
   'Property Identifier' = (any valid ~~non-array~~ property of the specified object),
   'Property Value' = (any value appropriate to the specified property)

Notes to Tester: This test may also be used to test an IUT's ability to write a single array element by checking for the inclusion of the 'Array Index' parameter.

8.23.2 Writing Multiple Properties of a Single Object

Purpose: To verify that the IUT can correctly initiate a WritePropertyMultiple service request containing multiple properties of a single object. If the IUT cannot be configured to initiate a WritePropertyMultiple service request containing multiple properties of a single object, then this test shall be omitted.

Test Steps:

1. RECEIVE WritePropertyMultiple-Request,
   'Object Identifier' = (any valid object identifier),
   'Property Identifier' = (any valid non-array property of the specified object),
   'Property Value' = (any value appropriate to the specified property),
   'Property Identifier' = (any valid ~~non-array~~ property of the specified object that was not previously used),
   'Property Value' = (any value appropriate to the specified property)

-- … (Any number of properties ≥ 2 is acceptable.)

Notes to Tester: This test may also be used to test an IUT's ability to write one or more array elements by checking for the inclusion of the 'Array Index' parameter for one or more of the properties.

8.23.3 Writing Multiple Objects, One Property Each

Purpose: To verify that the IUT can correctly initiate a WritePropertyMultiple service request containing multiple objects and a single property for each object. If the IUT cannot be configured to initiate a WritePropertyMultiple service request containing multiple objects and a single property for each object, then this test shall be omitted.

Test Steps:

```
1.    RECEIVE WritePropertyMultiple-Request,
          'Object Identifier' =     (any valid object identifier),
          'Property Identifier' =  (any valid non-array property of the specified object),
          'Property Value' =        (any value appropriate to the specified property),
          'Object Identifier' =     (any valid object identifier not previously used),
          'Property Identifier' =  (any valid ~~non-array~~ property of the specified object),
          'Property Value' =        (any value appropriate to the specified property)
--        …  (Any number of (object, property, value) tuples ≥ 2 is acceptable.)
```

Notes to Tester: This test may also be used to test an IUT's ability to write one or more array elements by checking for the inclusion of the 'Array Index' parameter for one or more of the properties.

8.23.4 Writing Multiple Objects, Multiple Properties for Each

Purpose: To verify that the IUT can correctly initiate a WritePropertyMultiple service request containing multiple objects and multiple properties for each object. If the IUT cannot be configured to initiate a WritePropertyMultiple service request containing multiple objects and multiple properties for each object, then this test shall be omitted.

Test Steps:

```
1.    RECEIVE WritePropertyMultiple-Request,
          'Object Identifier' =     (any valid object identifier),
          'Property Identifier' =  (any valid non-array property of the specified object),
          'Property Value' =        (any value appropriate to the specified property),
          'Property Identifier' =  (any valid ~~non-array~~ property of the specified object that was not previously used),
          'Property Value' =        (any value appropriate to the specified property),
--        …  (Additional properties may be included here.)

          'Object Identifier' =     (any valid object identifier not previously used),
          'Property Identifier' =  (any valid non-array property of the specified object),
          'Property Value' =        (any value appropriate to the specified property),
          'Property Identifier' =  (any valid ~~non-array~~ property of the specified object that was not previously used),
          'Property Value' =        (any value appropriate to the specified property)
--        …  (Additional properties may be included here.)

--        …  (Additional objects and properties may be included here.)
```

Notes to Tester: This test may also be used to test an IUT's ability to write one or more array elements by checking for the inclusion of the 'Array Index' parameter for one or more of the properties.

135.1-2009l-7. Modify Event Notifications Tests to Allow Use of Event Enrollment Objects.

Rationale

Modify event reporting tests so that they may be applied to any object type that implements the algorithm tested.

Similar changes were made to Clause 8.2.2 in 135.1-2009h.

[Change **Clause 8.4.1**, p. 118]

8.4.1 CHANGE_OF_BITSTRING Tests

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. ~~This test applies to Event Enrollment objects with an Event_Type of CHANGE_OF_BITSTRING.~~

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

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 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_State = NORMAL
2.  IF (the referenced property is writable) THEN
        WRITE (referenced property) = (a value x: x = one of the List_Of_Bitstring_Values after the bitmask is
                                      applied)
    ELSE
        MAKE (the referenced property have a value x: x = one of the List_Of_Bitstring_Values after the
                bitmask is applied)
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 Enrollment~~ object being tested),
            '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_BITSTRING,
            'Notify Type' =                    EVENT | ALARM,
            'AckRequired' =                    TRUE | FALSE,
            'From State' =                     NORMAL,
            'To State' =                       OFFNORMAL,
            ' Event Values' =                  referenced-bitstring, Status_Flags
5.  TRANSMIT BACnet-SimpleACK-PDU
6.  *IF (the object being tested is not an Event Enrollment object) THEN*
```

```
        VERIFY Status_Flags = (TRUE, FALSE, ?, ?)
7.  VERIFY Event_State = 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 is writable) THEN
        WRITE (referenced property) = (a value x: x corresponds to a NORMAL state)
    ELSE
        MAKE (the referenced property have a value x: x corresponds to a NORMAL state)
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 ~~Event Enrollment~~ 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_BITSTRING,
            'Notify Type' =                     EVENT | ALARM,
            'AckRequired' =                     TRUE | FALSE,
            'From State' =                      OFFNORMAL,
            'To State' =                        NORMAL,
            ' Event Values' =                   referenced-bitstring, Status_Flags
12. TRANSMIT BACnet-SimpleACK-PDU
13. *IF (the object being tested is not an Event Enrollment object) THEN*
        VERIFY Status_Flags = (FALSE, FALSE, ?, ?)
14. VERIFY Event_State = 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 **Clauses 8.4.3 through 8.4.6,** p. 122]

8.4.3 CHANGE_OF_VALUE Tests

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

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. ~~This test applies to Event Enrollment objects with an Event_Type of CHANGE_OF_VALUE.~~

Test Concept: The object begins the test in a NORMAL state. The referenced property is changed by a value that is less than the Referenced_Property_Increment. The tester verifies that no event notification is transmitted. The referenced property is changed again to a value that differs from the original value by an amount greater than the

Referenced_Property_Increment. The tester verifies that an event notification message is transmitted and that the proper 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 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 = NORMAL
2.  IF (the referenced property is writable) THEN
        WRITE (referenced property) = (a value x: x differs from the initial value by less than
                                  Referenced_Property_Increment)
    ELSE
        MAKE (the referenced property have a value x: x differs from the initial value by less than
              Referenced_Property_Increment)
3.  WAIT (Time_Delay + **Notification Fail Time**)
4.  CHECK (verify that no event notification message is transmitted)
5.  IF (the referenced property is writable) THEN
        WRITE (referenced property) = (a value x: x differs from the initial value in step 1 by more than
                                  Referenced_Property_Increment)
    ELSE
        MAKE (the referenced property have a value x: x differs from the initial value in step 1 by more than
              Referenced_Property_Increment)
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 ~~Event Enrollment~~ 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_VALUE,
            'Notify Type' =                   EVENT | ALARM,
            'AckRequired' =                   TRUE | FALSE,
            'From State' =                    NORMAL,
            'To State' =                      NORMAL,
            ' Event Values' =                 changed-value, Status_Flags
8.  TRANSMIT BACnet-SimpleACK-PDU
9.  *IF (the object being tested is not an Event Enrollment object) THEN*
        VERIFY Status_Flags = (FALSE, FALSE, ?, ?)
10. VERIFY Event_State = 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

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. ~~This test applies to Event Enrollment objects with an Event_Type of CHANGE_OF_VALUE.~~

Test Concept: The object begins the test in a NORMAL state. The referenced property is changed to a new value such that none of the bits in the Bitmask are changed. The tester verifies that no event notification is transmitted. The referenced property is changed again to a value that differs in one or more bits that are included in the Bitmask. The tester verifies that an event notification message is transmitted and that the proper 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 shall have a value of TRUE. The Bitmask shall be configured so that at least one but not all bits of the referenced property 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 = NORMAL
2.  IF (the referenced property is writable) THEN
        WRITE (referenced property) = (a value x: x differs from the initial value but only in bits that are not
                                       included in Bitmask)
    ELSE
        MAKE (the referenced property have a value x: x differs from the initial value but only in bits that are not
              included in Bitmask)
3.  WAIT (Time_Delay + **Notification Fail Time**)
4.  CHECK (verify that no event notification message is transmitted)
5.  IF (the referenced property is writable) THEN
        WRITE (referenced property) = (a value x: x differs from the initial value in one or more bits included in
                                       Bitmask)
    ELSE
        MAKE (the referenced property have a value x: x differs from the initial value one or more bits included
               in Bitmask)
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 ~~Event Enrollment~~ 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_VALUE,
            'Notify Type' =                   EVENT | ALARM,
            'AckRequired' =                   TRUE | FALSE,
            'From State' =                    NORMAL,
            'To State' =                      NORMAL,
            ' Event Values' =                 changed-value, Status_Flags
8.  TRANSMIT BACnet-SimpleACK-PDU
9.  *IF (the object being tested is not an Event Enrollment object) THEN*
        VERIFY Status_Flags = (FALSE, FALSE, ?, ?)
10. VERIFY Event_State = 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.

8.4.4 COMMAND_FAILURE Tests

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. ~~This test 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 Feedback_Value (Feedback_Property_Reference) shall be decoupled from the input signal that is normally used to verify the output. Initially Present_Value (referenced property) and Feedback_Value (Feedback_Property_Reference) are in agreement. Present_Value (the referenced property) is changed and an event notification should be transmitted indicating a transition to an OFFNORMAL state. The Feedback_Value (Feedback_Property_Reference) is changed to again agree with the Present_Value (referenced property). 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 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.

[Note to Reviewers: The modifications to steps 5 and 12 shown below supersede the modifications to these steps shown in section 4 of published Addendum 135.1-2009h.]

Test Steps:

```
1.  VERIFY Event_State = NORMAL
2.  *IF (the object being tested is not an Event Enrollment object) THEN*
        VERIFY Status_Flags = (FALSE, FALSE, FALSE, FALSE)
3.  IF (Present_Value is writable) THEN
        WRITE Present_Value = (a different value)
    ELSE
        MAKE (Present_Value take on a different value)
4.  WAIT (Time_Delay)
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 object referenced by the Event Enrollment~~ object being tested),
            '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' =                    COMMAND_FAILURE,
            'Notify Type' =                   EVENT | ALARM,
            'AckRequired' =                   TRUE | FALSE,
          'From State' =                NORMAL,
          'To State' =                  OFFNORMAL,
          ' Event Values' =             Present_Value, Status_Flags, Feedback_Value
6.  TRANSMIT BACnet-SimpleACK-PDU
7.  IF (the object being tested is not an Event Enrollment object) THEN
        VERIFY Status_Flags = (TRUE, FALSE, ?, ?)
8.  VERIFY Event_State = 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 is writable) THEN
        WRITE Feedback_Value = (a value consistent with Present_Value)
    ELSE
        MAKE (Feedback_Value take on a value consistent with Present_Value)
11. WAIT (Time_Delay)
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 object referenced by the Event Enrollment~~ 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' =                COMMAND_FAILURE,
          'Notify Type' =               EVENT | ALARM,
          'AckRequired' =               TRUE | FALSE,
          'From State' =                OFFNORMAL,
          'To State' =                  NORMAL,
          ' Event Values' =             Present_Value, Status_Flags, Feedback_Value
13. TRANSMIT BACnet-SimpleACK-PDU
14. IF (the object being tested is not an Event Enrollment object) THEN
        VERIFY Status_Flags = (FALSE, FALSE, ?, ?)
15. VERIFY Event_State = 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.

8.4.5 FLOATING_LIMIT Tests

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. ~~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 object begins the test in a NORMAL state. The referenced property is raised to a value that is below but within Deadband of the high limit. At this point the object should still be in a NORMAL state. The referenced property 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. The referenced property is lowered to a value that is below the high limit but still within Deadband of the limit. The object should remain in the HIGH_LIMIT state. The referenced property is lowered further to a normal value that is not within 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. 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.

[Note to Reviewers: The modifications to steps 8 and 19 shown below supersede the modifications to these steps shown in section 4 of published Addendum 135.1-2009h.]

Test Steps:

```
1.  VERIFY Event_State = NORMAL
2.  IF (the referenced property is writable) THEN
        WRITE (referenced property) = (a value x:
                (Setpoint_Reference + High_Diff_Limit – Deadband)< x < (Setpoint_Reference + High_Diff_Limit))
    ELSE
        MAKE (the referenced property have a value x:
                (Setpoint_Reference + High_Diff_Limit – Deadband)< x < (Setpoint_Reference + High_Diff_Limit))
3.  WAIT (Time_Delay + **Notification Fail Time**)
4.  CHECK (verify that no notification message has been transmitted)
5.  VERIFY Event_State = NORMAL
6.  IF (the referenced property is writable) THEN
        WRITE (referenced property) = (a value x: x > (Setpoint_Reference + High_Diff_Limit))
    ELSE
        MAKE (the referenced property have a value x: x > (Setpoint_Reference + High_Diff_Limit))
7.  WAIT (Time_Delay)
8.  BEFORE **Notification Fail Time**
        RECEIVE ConfirmedEventNotification-Request,
            'Process Identifier' =            (any valid process ID),
            'Initiating Device Identifier' =  IUT,
            'Event Object Identifier' =       (the ~~Loop object being tested or the object referenced by the Event
Enrollment~~object being tested),
            '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' =                    FLOATING_LIMIT,
            'Notify Type' =                   EVENT | ALARM,
            'AckRequired' =                   TRUE | FALSE,
            'From State' =                    NORMAL,
            'To State' =                      HIGH_LIMIT,
            ' Event Values' =                 reference-value, Status_Flags, setpoint-value, error-limit,
9.  TRANSMIT BACnet-SimpleACK-PDU
10. *IF (the object being tested is not an Event Enrollment object) THEN*
        VERIFY Status_Flags = (TRUE, FALSE, ?, ?)
11. VERIFY Event_State = HIGH_LIMIT
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 is writable) THEN
        WRITE (referenced property) = (a value x:
                (Setpoint_Reference + High_Diff_Limit – Deadband)< x < Setpoint_Reference + High_Diff_Limit))
    ELSE
        MAKE (the referenced property have a value x:
                (Setpoint_Reference + High_Diff_Limit – Deadband)< x < Setpoint_Reference + High_Diff_Limit))
14. WAIT (Time_Delay + **Notification Fail Time**)
15. CHECK (verify that no notification message has been transmitted)
```

```
16. VERIFY Event_State = HIGH_LIMIT
17. IF (the referenced property is writable) THEN
        WRITE (referenced property) = (a value x:
            (Setpoint_Reference - Low_Diff_Limit + Deadband) < x < (Setpoint_Reference + High_Diff_Limit – Deadband))
    ELSE
        MAKE (the referenced property have a value x:
            (Setpoint_Reference - Low_Diff_Limit + Deadband) < x < (Setpoint_Reference + High_Diff_Limit – Deadband))
18. WAIT (Time_Delay)
19. BEFORE Notification Fail Time
        RECEIVE ConfirmedEventNotification-Request,
            'Process Identifier' =              (any valid process ID),
            'Initiating Device Identifier' =    IUT,
            'Event Object Identifier' =         (the ~~Loop object being tested or the object referenced by the Event Enrollment~~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' =                      FLOATING_LIMIT,
            'Notify Type' =                     EVENT | ALARM,
            'AckRequired' =                     TRUE | FALSE,
            'From State' =                      HIGH_LIMIT,
            'To State' =                        NORMAL,
            ' Event Values' =                   reference-value, Status_Flags, setpoint-value, error-limit,
20. TRANSMIT BACnet-SimpleACK-PDU
21. IF (the object being tested is not an Event Enrollment object) THEN
        VERIFY Status_Flags = (FALSE, FALSE, ?, ?)
22. VERIFY Event_State = 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 is writable) THEN
        WRITE (referenced property) = (a value x:
                (Setpoint_Reference - Low_Diff_Limit < x < (Setpoint_Reference - Low_Diff_Limit + Deadband))
    ELSE
        MAKE (the referenced property have a value x:
                (Setpoint_Reference - Low_Diff_Limit < x < (Setpoint_Reference - Low_Diff_Limit + Deadband))
25. WAIT (Time_Delay + Notification Fail Time)
26. CHECK (verify that no notification message has been transmitted)
27. VERIFY Event_State = NORMAL
28. IF (the referenced property is writable) THEN
        WRITE (referenced property) = (a value x such x < (Setpoint_Reference -  Low_Diff_Limit))
    ELSE
        MAKE (referenced property have a value x: x < (Setpoint_Reference - Low_Diff_Limit))
29. WAIT (Time_Delay)
30. BEFORE Notification Fail Time
        RECEIVE ConfirmedEventNotification-Request,
            'Process Identifier' =              (any valid process ID),
            'Initiating Device Identifier' =    IUT,
            'Event Object Identifier' =         (the ~~Loop object being tested or the Event Enrollment ~~object being tested),
            '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' =                      FLOATING_LIMIT,
            'Notify Type' =                     EVENT | ALARM,
            'AckRequired' =                     TRUE | FALSE,
            'From State' =                  NORMAL,
            'To State' =                    LOW_LIMIT,
            ' Event Values' =               reference-value, Status_Flags, setpoint-value, error-limit,
31. TRANSMIT BACnet-SimpleACK-PDU
32. IF (the object being tested is not an Event Enrollment object) THEN
        VERIFY Status_Flags = (TRUE, FALSE, ?, ?)
33. VERIFY Event_State = 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 is writable) THEN
        WRITE (referenced property) = (a value x:
                (Setpoint_Reference - Low_Limit) < x < (Setpoint_Reference - Low_Limit + Deadband))
    ELSE
        MAKE (the referenced property have a value x:
                (Setpoint_Reference - Low_Limit) < x < (Setpoint_Reference - Low_Limit + Deadband))
36. WAIT (Time_Delay + Notification Fail Time)
37. CHECK (verify that no notification message has been transmitted)
38. VERIFY Event_State = Low_Limit
39. IF (the referenced property is writable) THEN
        WRITE (referenced property) = (a value x:
            (Setpoint_Reference - Low_Limit + Deadband) < x < (Setpoint_Reference + High_Diff_Limit –
Deadband))
    ELSE
        MAKE (the referenced property have a value x:
            (Setpoint_Reference - Low_Limit + Deadband) < x < (Setpoint_Reference + High_Diff_Limit –
Deadband))
40. WAIT (Time_Delay)
41. BEFORE Notification Fail Time
        RECEIVE ConfirmedEventNotification-Request,
            'Process Identifier' =          (any valid process ID),
            'Initiating Device Identifier' = IUT,
            'Event Object Identifier' =     (the ~~Loop object being tested or the Event Enrollment~~ 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' =                  OUT_OF_RANGE,
            'Notify Type' =                 EVENT | ALARM,
            'AckRequired' =                 TRUE | FALSE,
            'From State' =                  LOW_LIMIT,
            'To State' =                    NORMAL,
            ' Event Values' =               reference-value, Status_Flags, setpoint-value, error-limit,
42. TRANSMIT BACnet-SimpleACK-PDU
43. IF (the object being tested is not an Event Enrollment object) THEN
        VERIFY Status_Flags = (FALSE, FALSE, ?, ?)
44. VERIFY Event_State = 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.

8.4.6 OUT_OF_RANGE Tests

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. ~~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 object begins the test in a NORMAL state. The Present_Value (referenced property) is raised to a value that is below but within Deadband of the high limit. At this point the object should still be in a NORMAL state. The Present_Value (referenced property) 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. The Present_Value (referenced property) is lowered to a value that is below the high limit but still within Deadband of the limit. The object should remain in the HIGH_LIMIT state. The Present_Value (referenced property) is lowered further to a normal value that is not within 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 events. 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. If an Event Enrollment object is being tested Present_Value should be replaced by the appropriate property reference.

[Note to Reviewers: The modifications to steps 8, 19, 30, and 41 shown below supersede the modifications to these steps shown in section 4 of published Addendum 135.1-2009h.]

Test Steps:

```
1.  VERIFY Event_State = NORMAL
2.  IF (Present_Value is writable) THEN
        WRITE Present_Value = (a value x: (High_Limit – Deadband)< x < High_Limit)
    ELSE
        MAKE (Present_Value have a value x: (High_Limit – Deadband)< x < High_Limit)
3.  WAIT (Time_Delay + **Notification Fail Time**)
4.  CHECK (verify that no notification message has been transmitted)
5.  VERIFY Event_State = NORMAL
6.  IF (Present_Value is writable) THEN
        WRITE Present_Value = (a value x such x > High_Limit)
    ELSE
        MAKE (Present_Value have a value x: x > High_Limit)
7.  WAIT (Time_Delay)
8.  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 object referenced by the
                                               Event Enrollment~~ object being tested),
            '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' =                     OUT_OF_RANGE,
            'Notify Type' =                    EVENT | ALARM,
            'AckRequired' =                TRUE | FALSE,
            'From State' =                 NORMAL,
            'To State' =                   HIGH_LIMIT,
            ' Event Values' =              Present_Value, Status_Flags, Deadband, High_Limit
9.  TRANSMIT BACnet-SimpleACK-PDU
10. IF (the object being tested is not an Event Enrollment object) THEN
        VERIFY Status_Flags = (TRUE, FALSE, ?, ?)
11. VERIFY Event_State = 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 is writable) THEN
        WRITE Present_Value = (a value x: (High_Limit – Deadband)< x < High_Limit)
    ELSE
        MAKE (Present_Value have a value x: (High_Limit – Deadband)< x < High_Limit)
14. WAIT (Time_Delay + Notification Fail Time)
15. CHECK (verify that no notification message has been transmitted)
16. VERIFY Event_State = HIGH_LIMIT
17. IF (Present_Value is writable) THEN
        WRITE Present_Value = (a value x: (Low_Limit + Deadband) < x < (High_Limit – Deadband))
    ELSE
        MAKE (Present_Value have a value x: (Low_Limit + Deadband) < x < (High_Limit – Deadband))
18. WAIT (Time_Delay)
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 or the object referenced by the Event Enrollment~~ 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' =                 OUT_OF_RANGE,
            'Notify Type' =                EVENT | ALARM,
            'AckRequired' =                TRUE | FALSE,
            'From State' =                 HIGH_LIMIT,
            'To State' =                   NORMAL,
            ' Event Values' =              Present_Value, Status_Flags, Deadband, High_Limit
20. TRANSMIT BACnet-SimpleACK-PDU
21. IF (the object being tested is not an Event Enrollment object) THEN
        VERIFY Status_Flags = (FALSE, FALSE, ?, ?)
22. VERIFY Event_State = 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 is writable) THEN
        WRITE Present_Value = (a value x: Low_Limit < x < (Low_Limit + Deadband))
    ELSE
        MAKE (Present_Value have a value x: Low_Limit < x < (Low_Limit + Deadband))
25. WAIT (Time_Delay + Notification Fail Time)
26. CHECK (verify that no notification message has been transmitted)
27. VERIFY Event_State = NORMAL
28. IF (Present_Value is writable) THEN
        WRITE Present_Value = (a value x such x < Low_Limit)
    ELSE
        MAKE (Present_Value have a value x: x < Low_Limit)
29. WAIT (Time_Delay)
30. 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 object referenced by the
                                                ~~Event Enrollment~~ object being tested),
            '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' =                      OUT_OF_RANGE,
            'Notify Type' =                     EVENT | ALARM,
            'AckRequired' =                     TRUE | FALSE,
            'From State' =                      NORMAL,
            'To State' =                        LOW_LIMIT,
            ' Event Values' =                   Present_Value, Status_Flags, Deadband, Low_Limit
31. TRANSMIT BACnet-SimpleACK-PDU
32. IF (the object being tested is not an Event Enrollment object) THEN
        VERIFY Status_Flags = (TRUE, FALSE, ?, ?)
33. VERIFY Event_State = 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 is writable) THEN
        WRITE Present_Value = (a value x: Low_Limit < x < (Low_Limit + Deadband))
    ELSE
        MAKE (Present_Value have a value x: Low_Limit < x < (Low_Limit + Deadband))
36. WAIT (Time_Delay + Notification Fail Time)
37. CHECK (verify that no notification message has been transmitted)
38. VERIFY Event_State = LOW_LIMIT
39. IF (Present_Value is writable) THEN
        WRITE Present_Value = (a value x: (Low_Limit + Deadband) < x < (High_Limit – Deadband))
    ELSE
        MAKE (Present_Value have a value x: (Low_Limit + Deadband) < x < (High_Limit – Deadband))
40. WAIT (Time_Delay)
41. 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 object referenced by the
                                                ~~Event Enrollment~~ 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' =                      OUT_OF_RANGE,
            'Notify Type' =                     EVENT | ALARM,
            'AckRequired' =                     TRUE | FALSE,
            'From State' =                      LOW_LIMIT,
            'To State' =                        NORMAL,
            ' Event Values' =                   Present_Value, Status_Flags, Deadband, Low_Limit
42. TRANSMIT BACnet-SimpleACK-PDU
43. IF (the object being tested is not an Event Enrollment object) THEN
        VERIFY Status_Flags = (FALSE, FALSE, ?, ?)
44. VERIFY Event_State = 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.

135.1-2009l-8. Add a Test for Acknowledging Offnormal Events.

Rationale

Add a test that ensures that offnormal transitions that indicate a 'To State' other than OFFNORMAL are successfully acknowledged using a 'To State' of OFFNORMAL.

[Add new **Clauses 9.1.1.Y1 & 9.1.1.Y2**, p. 194]

9.1.1.Y1 Successful Alarm Acknowledgment of Confirmed Event Notifications when 'To State' is an Offnormal State other than OFFNORMAL

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 an offnormal state other than OFFNORMAL (e.g. HIGH_LIMIT) and the 'Event State Acknowledged' parameter is OFFNORMAL.

Test Concept: An alarm is triggered that causes the IUT to notify the TD and at least one other device with an offnormal 'To State' other than OFFNORMAL. The TD acknowledges the alarm using all of the correct parameters and using an 'Event State Acknowledged' parameter of OFFNORMAL 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 shall be recipients of the alarm notification.

Test Steps: The test steps defined in Clause 9.1.1.1 shall be followed except that the 'To State' parameter shall be an offnormal state other than OFFNORMAL. When acknowledging the alarm, the TD shall use an 'Event State Acknowledged' parameter of OFFNORMAL.

Notes to Tester: A passing result is the same message sequence described as the passing result in 9.1.1.1.

9.1.1.Y2 Successful Alarm Acknowledgment of Unconfirmed Event Notifications when 'To State' is an Offnormal State other than OFFNORMAL

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, when the 'To State' parameter is an offnormal state other than OFFNORMAL (e.g. HIGH_LIMIT) and the 'Event State Acknowledged' parameter is OFFNORMAL.

Test Concept: An alarm is triggered that causes the IUT to notify the TD and at least one other device with an offnormal 'To State' other than OFFNORMAL. The TD acknowledges the alarm using all of the correct parameters and using an 'Event State Acknowledged' parameter of OFFNORMAL 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 shall be recipients of the alarm notification.

Test Steps: The test steps defined in Clause 9.1.1.4 shall be followed except that the 'To State' parameter shall be an offnormal state other than OFFNORMAL. When acknowledging the alarm, the TD shall use an 'Event State Acknowledged' parameter of OFFNORMAL.

Notes to Tester: A passing result is the same message sequence described as the passing result in Clause 9.1.1.4.

135.1-2009l-9. Update Expected Error Codes Negative COV Tests.

Rationale

Update the negative COV notification tests to expect the mandated error codes.

Also, test 9.2.2.5 is removed as it tests for functionality not mandated by 135-2008 or its addenda.

[Change **Clauses 9.2.2.1, 9.2.2.2, and 9.2.2.3,** pp. 202-203]

9.2.2.1 Change of Value Notification Arrives after Subscription has Expired

Purpose: To verify that an appropriate error is returned if a COV notification arrives after the subscription time period has expired.

Test Steps:

```
1.  RECEIVE SubscribeCOV,
        'Subscriber Process Identifier' =       (any valid process identifier),
        'Monitored Object Identifier' =         (any object X of a type that supports COV notification),
        'Issue Confirmed Notifications ' =      TRUE,
        'Lifetime' =                            (a value no greater than one minute)
2.  TRANSMIT BACnet-SimpleACK-PDU
3.  WAIT (a value two times Lifetime)
4.  TRANSMIT ConfirmedCOVNotification-Request,
            'Subscriber Process Identifier' =   (the process identifier used in step 2),
            'Initiating Device Identifier' =    TD,
            'Monitored Object Identifier' =     X,
            'Time Remaining' =                  (any amount of time greater than 0),
            'List of Values' =                  (a list of values appropriate to object X)
5.  IF (Protocol_Revision is present and Protocol_Revision ≥ 10) THEN
        RECEIVE
            BACnet-Error-PDU,
                Error Class  =                  SERVICES,
                Error Code  =                   (UNKNOWN_SUBSCRIPTION) |
            (BACnet-SimpleACK-PDU)
    ELSE
        RECEIVE
            BACnet-Error-PDU,
                Error Class  =                  SERVICES,
                Error Code  =                   (any valid error code for class SERVICES)  |
            (BACnet-SimpleACK-PDU)
```

9.2.2.2 Change of Value Notifications with Invalid Process Identifier

Purpose: To verify that an appropriate error is returned if a COV notification arrives that contains a process identifier that does not match any current subscriptions.

Test Steps:

```
1.  RECEIVE SubscribeCOV,
        'Subscriber Process Identifier' =       (any valid process identifier),
        'Monitored Object Identifier' =         (any object X of a type that supports COV notification),
        'Issue Confirmed Notifications ' =      TRUE,
        'Lifetime' =                            (a value no greater than one minute)
2.  TRANSMIT BACnet-SimpleACK-PDU
3.  TRANSMIT ConfirmedCOVNotification-Request,
            'Subscriber Process Identifier' =   (a process identifier different from the one used in step 2),
            'Initiating Device Identifier' =    TD,
            'Monitored Object Identifier' =     X,
            'Time Remaining' =                  (any amount of time greater than 0),
            'List of Values' =                  (a list of values appropriate to object X)
4.  IF (Protocol_Revision is present and Protocol_Revision ≥ 10) THEN
        RECEIVE
            BACnet-Error-PDU,
                Error Class  =                  SERVICES,
                Error Code  =                   (UNKNOWN_SUBSCRIPTION) |
            (BACnet-SimpleACK-PDU)
    ELSE
        RECEIVE
            BACnet-Error-PDU,
                Error Class  =                  SERVICES,
                Error Code  =                   (any valid error code for class SERVICES)  |
            (BACnet-SimpleACK-PDU)
```

9.2.2.3 Change of Value Notifications with Invalid Initiating Device Identifier

Purpose: To verify that an appropriate error is returned if a COV notification arrives that contains an initiating device identifier that does not match any current subscriptions.

Test Steps:

```
1.  RECEIVE SubscribeCOV,
        'Subscriber Process Identifier' =       (any valid process identifier),
        'Monitored Object Identifier' =         (any object X of a type that supports COV notification),
        'Issue Confirmed Notifications ' =      TRUE,
        'Lifetime' =                            (a value no greater than one minute)
2.  TRANSMIT BACnet-SimpleACK-PDU
3.  TRANSMIT ConfirmedCOVNotification-Request,
            'Subscriber Process Identifier' =   (the process identifier different used in step 2),
            'Initiating Device Identifier' =    (any valid Device object except TD),
            'Monitored Object Identifier' =     X,
            'Time Remaining' =                  (any amount of time greater than 0),
            'List of Values' =                  (a list of values appropriate to object X)
4.  IF (Protocol_Revision is present and Protocol_Revision ≥ 10) THEN
        RECEIVE
            BACnet-Error-PDU,
                Error Class  =                  SERVICES,
                Error Code  =                   (UNKNOWN_SUBSCRIPTION) |
            (BACnet-SimpleACK-PDU)
    ELSE
        RECEIVE
            BACnet-Error-PDU,
                Error Class  =                  SERVICES,
            Error Code  =                    (any valid error code for class SERVICES)  |
        (BACnet-SimpleACK-PDU)
```

[Remove **Clause 9.2.2.5**, p 203]

~~**9.2.2.5 Change of Value Notifications with an Invalid List of Values**~~

~~Purpose: To verify that an appropriate reject is returned if a COV notification arrives that contains a list of values that is not appropriate for the object type of the monitored object.~~

~~Test Steps:~~

```
~~1.  RECEIVE SubscribeCOV,~~
~~        'Subscriber Process Identifier' =      (any valid process identifier),~~
~~        'Monitored Object Identifier' =        (any object X of a type that supports COV notification),~~
~~        'Issue Confirmed Notifications ' =     TRUE,~~
~~        'Lifetime' =                           (a value no greater than one minute)~~
~~2.  TRANSMIT BACnet-SimpleACK-PDU~~
~~3.  TRANSMIT ConfirmedCOVNotification-Request,~~
~~            'Subscriber Process Identifier' =  (the process identifier used in step 2),~~
~~            'Initiating Device Identifier' =   TD,~~
~~            'Monitored Object Identifier' =    X,~~
~~            'Time Remaining' =                 (any amount of time greater than 0),~~
~~            'List of Values' =                 (a list of values that is not appropriate to object X)~~
~~4.      RECEIVE BACnet-Reject-PDU,~~
~~            'Reject Reason' =                  INCONSISTENT_PARAMETERS |~~
~~                                               INVALID_PARAMETER_DATATYPE |~~
~~                                               INVALID_TAG~~
```

[Change **Clause 9.10.2.1**, p 219]

9.10.2.1 The Monitored Object Does Not Support COV Notification

Purpose: To verify that the IUT correctly responds to a SubscribeCOV request to establish a subscription when the monitored object does not support COV notifications.

Test Steps:

```
1.  TRANSMIT SubscribeCOV-Request,
        'Subscriber Process Identifier' =   (any valid process identifier),
        'Monitored Object Identifier' =     (any object that does not support COV notifications),
        'Issue Confirmed Notifications' =   TRUE,
        'Lifetime' =                        60
2.  IF (Protocol_Revision is present and Protocol_Revision ≥ 10) THEN
        RECEIVE BACnet-Error PDU,
            Error Class =     OBJECT,
            Error Code =      OPTIONAL_FUNCTIONALITY_NOT_SUPPORTED
    ELSE
        RECEIVE BACnet-Error PDU,
            Error Class =     OBJECT,
            Error Code =      OPTIONAL_FUNCTIONALITY_NOT_SUPPORTED |
        (~~RECEIVE~~ BACnet-Error PDU,
            Error Class  =    SERVICES,
            Error Code  =     SERVICE_REQUEST_DENIED | OTHER ~~| COV_SUBSCRIPTION_FAILED~~) |
        (BACnet-Error PDU,
```

Error Class = PROPERTY,
Error Code = NOT_COV_PROPERTY)

135.1-2009l-10. Correct the Use of WAIT vs. BEFORE in COV Notification Tests.

Rationale

A number of COV notification tests incorrectly use a WAIT construct instead of the BEFORE construct.

Some of the tests also allow for a negative response to a COV Subscription, which is inappropriate within positive COV subscription tests. This allowance is removed.

Add in the sending BACnet-SimpleACK-PDUs in response to confirmed requests.

In 9.10.1.7, the allowable value for the lifetime subscription is relaxed.

[Change **Clauses 9.10.1.1** through **9.10.1.3**, p. 215]

9.10.1.1 Confirmed COV Notifications

Purpose: To verify that the IUT correctly responds to a SubscribeCOV request to establish a subscription for confirmed COV notifications. An implementation that supports COV reporting cannot respond with an error for both this test and the test in 9.10.1.2.

Test Steps:

```
1.  TRANSMIT SubscribeCOV-Request,
        'Subscriber Process Identifier' =       (any valid process identifier),
        'Monitored Object Identifier' =         (any object supporting COV notifications),
        'Issue Confirmed Notifications' =       TRUE,
        'Lifetime' =                            (any value > 0 if automatic cancellation is supported, otherwise 0)
2.  RECEIVE BACnet-SimpleACK-PDU
~~3.  WAIT **Notification Fail Time**~~
~~4.  IF (the IUT supports confirmed notifications) THEN~~
*3.  BEFORE **Notification Fail Time***
        RECEIVE ConfirmedCOVNotification-Request,
            'Subscriber Process Identifier' =   (the same identifier used in the subscription),
            'Initiating Device Identifier' =    IUT,
            'Monitored Object Identifier' =     (the same object used in the subscription),
            'Time Remaining' =                  (any value > 0 if automatic cancellation is supported, otherwise 0),
            'List of Values' =                  (values appropriate to the object type of the monitored object)
    ~~ELSE~~
        ~~RECEIVE BACnet-Error PDU,~~
                ~~Error Class =                     SERVICES,~~
                ~~Error Code =                      SERVICE_REQUEST_DENIED | OTHER~~
*4.  TRANSMIT BACnet-SimpleACK-PDU*
```

9.10.1.2 Unconfirmed COV Notifications

Purpose: To verify that the IUT correctly responds to a SubscribeCOV request to establish a subscription for unconfirmed COV notifications. An implementation that supports COV reporting cannot respond with an error for both this test and the test in 9.10.1.1.

Test Steps:

```
1.  TRANSMIT SubscribeCOV-Request,
        'Subscriber Process Identifier' =       (any valid process identifier),
        'Monitored Object Identifier' =         (any object supporting COV notifications),
        'Issue Confirmed Notifications' =       FALSE,
        'Lifetime' =                            (any value > 0 if automatic cancellation is supported, otherwise 0)
2.  RECEIVE BACnet-SimpleACK-PDU
~~3.  WAIT Notification Fail Time~~
~~4.  IF (the IUT supports confirmed notifications) THEN~~
3.  BEFORE Notification Fail Time
        RECEIVE UnconfirmedCOVNotification-Request,
            'Subscriber Process Identifier' =   (the same identifier used in the subscription),
            'Initiating Device Identifier' =    IUT,
            'Monitored Object Identifier' =     (the same object used in the subscription),
            'Time Remaining' =                  (any value > 0 if automatic cancellation is supported, otherwise 0),
            'List of Values' =                  (values appropriate to the object type of the monitored object)
~~    ELSE~~
~~        RECEIVE BACnet-Error PDU,~~
~~                Error Class  =    SERVICES,~~
~~                Error Code   =    SERVICE_REQUEST_DENIED | OTHER~~
4.  TRANSMIT BACnet-SimpleACK-PDU
```

9.10.1.3 Explicit Indefinite Lifetime COV Subscriptions

Purpose: To verify that the IUT correctly responds to a SubscribeCOV request to establish a subscription with an indefinite lifetime (lifetime = 0). Either confirmed or unconfirmed notifications may be used but at least one of these options must be supported by the IUT.

Test Steps:

[The test steps have been renumbered without showing the change marking for clarity.]

```
1.  TRANSMIT SubscribeCOV-Request,
        'Subscriber Process Identifier' =       (any valid process identifier),
        'Monitored Object Identifier' =         (any object supporting COV notifications),
        'Issue Confirmed Notifications' =       TRUE | FALSE,
        'Lifetime' =                            0
2.  RECEIVE BACnet-SimpleACK-PDU
~~3.  WAIT Notification Fail Time~~
3.  IF (the subscription was for confirmed notifications) THEN
        BEFORE Notification Fail Time
            RECEIVE ConfirmedCOVNotification-Request,
                'Subscriber Process Identifier' =   (the same identifier used in the subscription),
                'Initiating Device Identifier' =    IUT,
                'Monitored Object Identifier' =     (the same object used in the subscription),
                'Time Remaining' =                  0,
                'List of Values' =                  (values appropriate to the object type of the monitored object)
            TRANSMIT BACnet-SimpleACK-PDU
    ELSE
        BEFORE Notification Fail Time
            RECEIVE UnconfirmedCOVNotification-Request,
                'Subscriber Process Identifier' =   (the same identifier used in the subscription),
                'Initiating Device Identifier' =    IUT,
                'Monitored Object Identifier' =     (the same object used in the subscription),
                'Time Remaining' =                  0,
                'List of Values' =                  (values appropriate to the object type of the monitored object)
4.  MAKE (a change to the monitored object that should cause a COV notification)
5.  IF (the subscription was for confirmed notifications) THEN
        BEFORE Notification Fail Time
            RECEIVE ConfirmedCOVNotification-Request,
                'Subscriber Process Identifier' =   (the same identifier used in the subscription),
                'Initiating Device Identifier' =    IUT,
                'Monitored Object Identifier' =     (the same object used in the subscription),
                'Time Remaining' =                  0,
                'List of Values' =                  (values appropriate to the object type of the monitored object
                                                     including the changed value that triggered the notification)
            TRANSMIT BACnet-SimpleACK-PDU
    ELSE
        BEFORE Notification Fail Time
            RECEIVE UnconfirmedCOVNotification-Request,
                'Subscriber Process Identifier' =   (the same identifier used in the subscription),
                'Initiating Device Identifier' =    IUT,
                'Monitored Object Identifier' =     (the same object used in the subscription),
                'Time Remaining' =                  0,
                'List of Values' =                  (values appropriate to the object type of the monitored object
                                                    including the changed value of that triggered the notification)
```

[Change **Clauses 9.10.1.7** and **9.10.1.8**, p 217]

9.10.1.7 Finite Lifetime Subscriptions

Purpose: To verify that the IUT correctly responds to a SubscribeCOV request to establish a subscription with a temporary lifetime. Either confirmed or unconfirmed notifications may be used but at least one of these options must be supported by the IUT.

Test Steps:

[The test steps have been renumbered without showing the change marking for clarity.]

```
1.  TRANSMIT SubscribeCOV-Request,
        'Subscriber Process Identifier' =     (any valid process identifier),
        'Monitored Object Identifier' =       (any object supporting COV notifications),
        'Issue Confirmed Notifications' =     TRUE | FALSE,
        'Lifetime' =                          ( a value between 60 seconds and 300 seconds)
2.  RECEIVE BACnet-SimpleACK-PDU
~~3.  WAIT Notification Fail Time~~
3.  IF (the subscription was for confirmed notifications) THEN
        BEFORE Notification Fail Time
            RECEIVE ConfirmedCOVNotification-Request,
                'Subscriber Process Identifier' =   (the same identifier used in the subscription),
                'Initiating Device Identifier' =    IUT,
                'Monitored Object Identifier' =     (the same object used in the subscription),
                'Time Remaining' =                  (A value approximately equal to, but not greater than, the
                                                    requested subscription lifetime),
                'List of Values' =                  (values appropriate to the object type of the monitored object)
            TRANSMIT BACnet-SimpleACK-PDU
    ELSE
        BEFORE Notification Fail Time
            RECEIVE UnconfirmedCOVNotification-Request,
                'Subscriber Process Identifier' =   (the same identifier used in the subscription),
                'Initiating Device Identifier' =    IUT,
                'Monitored Object Identifier' =     (the same object used in the subscription),
                'Time Remaining' =                  (A value approximately equal to, but not greater than, the
                                                    requested subscription lifetime),
                'List of Values' =                  (values appropriate to the object type of the monitored object)
4.  MAKE (a change to the monitored object that should cause a COV notification)
```

```
5.  IF (the subscription was for confirmed notifications) THEN
        BEFORE Notification Fail Time
            RECEIVE ConfirmedCOVNotification-Request,
                'Subscriber Process Identifier' =   (the same identifier used in the subscription),
                'Initiating Device Identifier' =    IUT,
                'Monitored Object Identifier' =     (the same object used in the subscription),
                'Time Remaining' =                  (a value greater than 0 and less than the requested subscription
                                                    lifetime),
                'List of Values' =                  (values appropriate to the object type of the monitored object)
            TRANSMIT BACnet-SimpleACK-PDU
    ELSE
        BEFORE Notification Fail Time
            RECEIVE UnconfirmedCOVNotification-Request,
                'Subscriber Process Identifier' =   (the same identifier used in the subscription),
                'Initiating Device Identifier' =    IUT,
                'Monitored Object Identifier' =     (the same object used in the subscription),
                'Time Remaining' =                  (a value greater than 0 and less than the requested subscription
                                                    lifetime),
                'List of Values' =                  (values appropriate to the object type of the monitored object
                                                    including the changed value of that triggered the notification)
6.  WAIT (the lifetime of the subscription)
7.  MAKE (a change to the monitored object that would cause a COV notification if there were an active subscription)
8.  CHECK (verify that the IUT did not transmit a COV notification message)
```

9.10.1.8 Updating Existing Subscriptions

Purpose: To verify that the IUT correctly responds to a SubscribeCOV request to update the lifetime of a subscription. Either confirmed or unconfirmed notifications may be used but at least one of these options must be supported by the IUT.

Test Concept: A subscription for COV notifications is made for 60 seconds. Before that subscription has expired a second subscription is made for 300 seconds. When the notification is sent in response to the second subscription the lifetime is checked to verify that it is greater than 60 but less than 300 seconds.

Test Steps:

[The test steps have been renumbered without showing the change marking for clarity.]

```
1.  TRANSMIT SubscribeCOV-Request,
        'Subscriber Process Identifier' =     (any valid process identifier),
        'Monitored Object Identifier' =       (any object supporting COV notifications),
        'Issue Confirmed Notifications' =     TRUE | FALSE,
        'Lifetime' =                          60
2.  RECEIVE BACnet-SimpleACK-PDU
~~3.  WAIT Notification Fail Time~~
3.  IF (the subscription was for confirmed notifications) THEN
        BEFORE Notification Fail Time
            RECEIVE ConfirmedCOVNotification-Request,
                'Subscriber Process Identifier' =   (the same identifier used in the subscription),
                'Initiating Device Identifier' =    IUT,
                'Monitored Object Identifier' =     (the same object used in the subscription),
                'Time Remaining' =                  60,
                'List of Values' =                  (values appropriate to the object type of the monitored object)
            TRANSMIT BACnet-SimpleACK-PDU
    ELSE
        BEFORE Notification Fail Time
```

```
            RECEIVE UnconfirmedCOVNotification-Request,
                'Subscriber Process Identifier' =    (the same identifier used in the subscription),
                'Initiating Device Identifier' =     IUT,
                'Monitored Object Identifier' =      (the same object used in the subscription),
                'Time Remaining' =                   60,
                'List of Values' =                   (values appropriate to the object type of the monitored object)
4.  TRANSMIT SubscribeCOV-Request,
        'Subscriber Process Identifier' =            (any valid process identifier),
        'Monitored Object Identifier' =              (any object supporting COV notifications),
        'Issue Confirmed Notifications' =            TRUE | FALSE,
        'Lifetime' =                                 (a value between 180 and 300 seconds)
5.  RECEIVE BACnet-SimpleACK-PDU
~~7.  WAIT Notification Fail Time~~
6.  IF (the subscription was for confirmed notifications) THEN
        BEFORE Notification Fail Time
            RECEIVE ConfirmedCOVNotification-Request,
                'Subscriber Process Identifier' =    (the same identifier used in the subscription),
                'Initiating Device Identifier' =     IUT,
                'Monitored Object Identifier' =      (the same object used in the subscription),
                'Time Remaining' =                   (the requested subscription lifetime),
                'List of Values' =                   (values appropriate to the object type of the monitored object)
            TRANSMIT BACnet-SimpleACK-PDU
    ELSE
        BEFORE Notification Fail Time
            RECEIVE UnconfirmedCOVNotification-Request,
                'Subscriber Process Identifier' =    (the same identifier used in the subscription),
                'Initiating Device Identifier' =     IUT,
                'Monitored Object Identifier' =      (the same object used in the subscription),
                'Time Remaining' =                   (the requested subscription lifetime),
                'List of Values' =                   (values appropriate to the object type of the monitored object)
```

135.1-2009l-11. Improve Reading Multiple Properties with Multiple Embedded Access Errors Test.

Rationale

The language in the existing test could lead to the selection of object identifiers that would elicit error responses other than an Error Class of OBJECT with an Error Code of UNKNOWN_OBJECT. The language is changed to clarify the error condition to check for.

[Change Clause **9.20.1.6**, p. 258]

9.20.1.6 Reading Multiple Properties with Multiple Embedded Access Errors

Purpose: To verify the ability to correctly execute a ReadPropertyMultiple service request for which the 'List of Read Access Specifications' contains specifications for multiple unsupported properties.

Test Steps:

```
1.  TRANSMIT ReadPropertyMultiple-Request,
        'Object Identifier' =    O1,
        'Property Identifier' =  P1,
        'Property Identifier' =  P2,
        'Property Identifier' =  (any property, P3, not supported in this object),
        'Property Identifier' =  (any property, P4, not supported in this object),
        'Object Identifier' =    (any *non-existent* object, O2, *which is of a type supported by the IUT),* ~~not supported in the IUT)~~
        'Property Identifier' =  P5,
        'Property Identifier' =  P6
2.  RECEIVE ReadPropertyMultiple-ACK,
        'Object Identifier' =    O1,
        'Property Identifier' =  P1,
        'Property Value' =       (the value of P1 specified in the EPICS),
        'Property Identifier' =  P2,
        'Property Value' =       (the value of P2 specified in the EPICS),
        'Property Identifier' =  P3,
        'Error Class' =          PROPERTY,
        'Error Code' =           UNKNOWN_PROPERTY,
        'Property Identifier' =  P4,
        'Error Class' =          PROPERTY,
        'Error Code' =           UNKNOWN_PROPERTY,
        'Object Identifier' =    O2,
        'Property Identifier' =  P5,
        'Error Class' =          OBJECT,
        'Error Code' =           (UNKNOWN_OBJECT),
        'Property Identifier' =  P6,
        'Error Class' =          OBJECT,
        'Error Code' =           (UNKNOWN_OBJECT)
```

135.1-2009l-12. Expand Allowable Errors for Older Product When Reading Array Properties.

Rationale

Improve the allowance for different error returns in products that claim conformance to older versions of the standard.

[Change **Clause 9.20.2.3**, p. 263]

9.20.2.3 Reading a Single Non-Array Property with an Array Index

Purpose: This test case verifies that the IUT can execute ReadPropertyMultiple service requests when the requested property value is not an array but an array index is included in the service request. This test shall only be performed if Protocol_Revision is present and has a value greater than or equal to 4.

Test Steps:

```
1.  TRANSMIT ReadPropertyMultiple-Request,
        ‘Object Identifier’ =        (Device, X),
        ‘Property Identifier’ =      Vendor_Name,
        ‘Array Index’ =              1
2.  IF (Protocol_Revision is present and Protocol_Revision ≥ 4) THEN
        RECEIVE
            (BACnet-Error-PDU,
                        ‘Error Class’  =            PROPERTY,
                        ‘Error Code’  =             PROPERTY_IS_NOT_AN_ARRAY) |
            (ReadPropertyMultiple-ACK,
                        ‘Object Identifier’ =       (Device, X),
                        ‘Property Identifier’ =     Vendor_Name,
                        ‘Array Index’ =             1,
                        ‘Error Class’ =             PROPERTY,
                        ‘Error Code’ =              PROPERTY_IS_NOT_AN_ARRAY)
    ELSE
        RECEIVE BACnet-Reject-PDU,
            'Reject Reason' = INCONSISTENT_PARAMETERS |
        (BACnet-Reject-PDU,
            ‘Reject Reason’ = INVALID_TAG) |
        (BACnet-Error-PDU,
            Error Class  =     PROPERTY,
            Error Code  =      INVALID_ARRAY_INDEX) |
        (ReadPropertyMultiple-ACK,
            'Object Identifier' =         (Device, X),
            'Property Identifier' =       Vendor_Name,
            'Array Index' =               1
            Error Class  =                PROPERTY,
            Error Code  =                 INVALID_ARRAY_INDEX) |
        (ReadPropertyMultiple-ACK,
            'Object Identifier' =         (Device, X),
            'Property Identifier' =       Vendor_Name,
            'Array Index' =               1
            Error Class  =                SERVICES,
            Error Code  =                 INCONSISTENT_PARAMETERS) |
         (BACnet-Error-PDU,
            Error Class  =     SERVICES,
            Error Code  =     INCONSISTENT_PARAMETERS)   |
        (BACnet-Error-PDU,
            Error Class  =    PROPERTY,
            Error Code  =     PROPERTY_IS_NOT_AN_ARRAY) |
        (ReadPropertyMultiple-ACK,
            'Object Identifier' =        (Device, X),
            'Property Identifier' =      Vendor_Name,
            'Array Index' =              1
            'Error Class'  =             PROPERTY,
            'Error Code'  =              PROPERTY_IS_NOT_AN_ARRAY)
```

135.1-2009l-13. Improve the Basic DeviceCommunicationControl Tests.

Rationale

Three of the DCC tests are changed to verify that the IUT does not initiate service requests for a longer period, and the use of the Who-Is service request is added to ensure more complete coverage.

All of the DCC execution tests are changed to clarify the CHECK language such that only the issuance of APDUs is checked for.

Remove the reliance of the DCC tests on EPICS property values.

[Change **Clauses 9.24.1.1** thru **9.24.1.3**, p. 279]

9.24.1.1 Indefinite Time Duration Restored by DeviceCommunicationControl

Purpose: To verify the correct execution of the DeviceCommunicationControl request service procedure when indefinite time duration is specified and communication is restored using the DeviceCommunicationControl service.

Test Steps:

[The test steps have been renumbered without showing the change marking for clarity.]

```
1.  TRANSMIT DeviceCommunicationControl-Request,
        'Enable/Disable' =          DISABLE,
        'Password' =                (any appropriate password as described in the Test Concept)
2.  RECEIVE BACnet-Simple-ACK-PDU
3.  WAIT **Internal Processing Fail Time**
4.  *WHILE (an arbitrary time > **Internal Processing Fail Time** selected by the tester has not expired) DO {*
        TRANSMIT ReadProperty-Request,
            'Object Identifier' =   (Device, X),
            'Property Identifier' = (any required non-array property of the Device object)
        *TRANSMIT*
            *DESTINATION =*         *LOCAL BROADCAST,*
            *Who-Is-Request*
        *WAIT (1 second)*           *-- poll delay*
    *}*
~~5.  WAIT (an arbitrary time > Internal Processing Fail Time selected by the tester)~~
*5.*  CHECK (Verify that the IUT has not transmitted any ~~messages~~ *APDUs* since the acknowledgment in step 2)
*6.*  TRANSMIT DeviceCommunicationControl-Request,
        'Enable/Disable' =      ENABLE,
        'Password' =            (any appropriate password as described in the Configuration Requirements)
*7.*  RECEIVE BACnet-Simple-ACK-PDU
*8.*  VERIFY (Device, X), (any required non-array property) = (*any valid value*~~the value for this property specified in the EPICS~~)
```

9.24.1.2 Indefinite Time Duration Restored by ReinitializeDevice

Purpose: To verify the correct execution of the DeviceCommunicationControl request service procedure when indefinite time duration is specified and communication is restored using the ReinitializeDevice service.

Dependencies: ReinitializeDevice Service Execution Tests, 9.27.

Test Steps:

[The test steps have been renumbered without showing the change marking for clarity.]

```
1.  TRANSMIT DeviceCommunicationControl-Request,
        'Enable/Disable' =      DISABLE,
        'Password' =            (any appropriate password as described in the Test Concept)
2.  RECEIVE BACnet-Simple-ACK-PDU
3.  WAIT **Internal Processing Fail Time**
4.  *WHILE (an arbitrary time > **Internal Processing Fail Time** selected by the tester has not expired) DO {*
        TRANSMIT ReadProperty-Request,
            'Object Identifier' =       (Device, X),
            'Property Identifier' =     (any required non-array property of the Device object)
        *TRANSMIT*
            *DESTINATION =      LOCAL BROADCAST,*
            *Who-Is-Request*
        *WAIT (1 second)                -- poll delay*
    *}*
~~5.  WAIT (an arbitrary time > **Internal Processing Fail Time** selected by the tester)~~
*5.*  CHECK (Verify that the IUT has not transmitted any ~~messages~~ *APDUs* since the acknowledgment in step 2)
*6.*  TRANSMIT ReinitializeDevice-Request,
        Reinitialized State of Device' =    WARMSTART,
        'Password' =                        (any appropriate password as described in the Test Concept)
*7.*  RECEIVE BACnet-Simple-ACK-PDU
*8.*  CHECK (Did the IUT perform a ~~COLDSTART~~ *WARMSTART* reboot?)
*9.*  VERIFY (Device, X), (any required non-array property) = (*any valid value*~~the value for this property specified in the EPICS~~)
```

9.24.1.3 Finite Time Duration

Purpose: To verify the correct execution of the DeviceCommunicationControl request service procedure when finite time duration is specified.

Test Steps:

[The test steps have been renumbered without showing the change marking for clarity.]

```
1.  TRANSMIT DeviceCommunicationControl-Request,
        'Time Duration' =       (a value T > 1, in minutes, selected by the tester),
        'Enable/Disable' =      DISABLE,
        'Password' =            (any appropriate password as described in the Test Concept)
2.  RECEIVE BACnet-Simple-ACK-PDU
3.  WAIT **Internal Processing Fail Time**
4.  *WHILE (Time Duration T in step 1 has not expired) DO {*
        TRANSMIT ReadProperty-Request,
        'Object Identifier' =       (Device, X),
        'Property Identifier' =     (any required non-array property of the Device object)
        *TRANSMIT*
            *DESTINATION =      LOCAL BROADCAST,*
            *Who-Is-Request*
        *WAIT (1 second)                -- poll delay*
    *}*
~~5.  WAIT (T )~~
*5.*  CHECK (Verify that the IUT did not transmit any ~~messages~~ *APDUs* between the acknowledgment in step 2 and
          expiration of timer T)
*6.*  VERIFY (Device, X), (any required non-array property) = (*any valid value*~~the value for this property specified in the EPICS~~)
```

[Change **Clause 9.24.1.4**, p. 281]

6. CHECK (Verify that the IUT has not transmitted any ~~messages~~ *APDUs* since the acknowledgment in step 2)

[Change **Clause 9.24.1.5**, p. 281]

6. CHECK (Verify that the IUT has not transmitted any ~~messages~~ *APDUs* since the acknowledgment in step 2)

[Change **Clause 9.24.1.6**, p. 282]

6. CHECK (Verify that the IUT has not transmitted any ~~messages~~ *APDUs* since the acknowledgment in step 2)

…

11. MAKE (do something to cause the IUT to initiate ~~a message~~ *an APDU*)

…

13. CHECK (Verify that the IUT initiated ~~a message~~ *an APDU*)

[Change **Clause 9.24.1.7**, p. 282]

6. CHECK (Verify that the IUT has not transmitted any ~~messages~~ *APDUs* since the acknowledgment in step 2)

…

12. MAKE (do something to cause the IUT to initiate ~~a message~~ *an APDU*)

…

14. CHECK (Verify that the IUT initiated ~~a message~~ *an APDU*)

[Change **Clause 9.24.1.8**, p. 283]

6. CHECK (Verify that the IUT has not transmitted any ~~messages~~ *APDUs* since the acknowledgment in step 2)

…

10. MAKE (do something to cause the IUT to initiate ~~a message~~ *an APDU*)

…

12. CHECK (Verify that the IUT initiated ~~a message~~ *an APDU*)

135.1-2009l-14. Add Alarm Summarization Tests.

Rationale

These tests verify that the IUT is able to use the alarm summarization services to update or present an alarm summary to the user.

[Change **Clause 8.6**, p. 156]
[Note that the existing test in Clause 8.6 is renumbered as 8.6.1 and retitled to allow for additional tests in this category]

8.6 GetAlarmSummary Service Initiation Tests

This clause defines the tests necessary to demonstrate support for initiating GetAlarmSummary service requests.

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)

8.6.X 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: 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.

[Add new **Clause 8.8.X1**, p. 159]

8.8.X1 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: 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.

[Add new **Clause 8.8.X2**, p. 159]

8.8.X2 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: 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.

135.1-2009l-15. Add Event Log Tests.

Rationale

Add in tests for the new Event Log object

[Add new **Clause 7.3.2.X**, p. 105]

7.3.2.X Event Log Tests

The tests in this section verify that Event Log objects correctly record event notifications.

Some of the general logging object tests in Clause 7.3.2.24 are also applicable to the Event Log object type.

7.3.2.X.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 Logs, LO1.

Test Steps:

```
1.  WRITE Enable = TRUE
2.  MAKE (IUT generate an EventNotification)
3.  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),
        '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.  TRANSMIT BACnet-SimpleACK-PDU
5.  MAKE (IUT generate an EventNotification)
6.  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),
        '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. TRANSMIT BACnet-SimpleACK-PDU
8. READ RC = LO1, Record_Count
9. TRANSMIT ReadRange-Request,

'Object Identifier' =	LO1,
'Property Identifier' =	Log_Buffer,
‘Reference Index’=	RC,
‘Count’ =	-2

10. 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. 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 SimpleACK-PDU is sent.

7.3.2.X.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 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 SimpleACK-PDU is expected.

7.3.2.X.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 TD. LO1 is an Event Log object in IUT which logs ACK_NOTIFICATIONs.

Test Steps:

1. WRITE Enable = TRUE
2. READ RC = LO1, Record_Count
3. MAKE (the IUT generate a notification)
4. 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),
'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.  TRANSMIT BACnet-SimpleACK-PDU
6.  TRANSMIT AcknowledgeAlarm-Request,
        'Acknowledging Process Identifier' =    (any valid value),
        'Event Object Identifier' =             O1,
        'Event State Acknowledged' =            S2,
        'Time Stamp' =                          T1,
        'Time of Acknowledgment' =              (the current time)
7.  RECEIVE BACnet-SimpleACK-PDU
8.  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),
            'Notify Type' =                     ACK_NOTIFICATION,
            'From State' =                      S1
9.  TRANSMIT BACnet-SimpleACK-PDU
10. TRANSMIT ReadRange-Request,
        'Object Identifier' =                   LO1,
        'Property Identifier' =                 Log_Buffer,
        ‘Reference Index’ =                     RC,
        ‘Count’ =                               -1
11. 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 SimpleACK-PDUs are sent in response to ConfirmedEventNotifications.

7.3.2.X.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 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 SimpleACK-PDU is expected.

135.1-2009l-16. Add Structured View Tests.

Rationale

Add in tests for Structure View objects.

[Add new **Clause 7.3.2.X**, p. 105]

7.3.2.X Structured View Tests

7.3.2.X.1 Subordinate_List Size Changes Subordinate_Annotations

Dependencies: WriteProperty Service Execution Tests, 9.22.

BACnet Reference Clauses: 12.29.7 and 12.29.8.

Purpose: This test case verifies that when the size of the Subordinate_List array is changed, the size of the Subordinate_Annotations array is changed accordingly to the same size. In addition, the test case verifies that the Instance Number in any uninitialized objectIdentifiers in the Subordinate_List has the value 4194303. If the size of the Subordinate_List and Subordinate_Annotations arrays cannot be changed, then this test shall not be performed.

Test Concept: The Subordinate_List and Subordinate_Annotations arrays are set to a certain size. They are then increased by writing the Subordinate_List array element 0, decreased by writing the Subordinate_List array, increased by writing the Subordinate_List array, and decreased by writing the Subordinate_List array element 0.

Configuration Requirements: The IUT shall be configured with a Structured View object with resizable Subordinate_List and Subordinate_Annotations arrays.

Test Steps:

```
1.  WRITE Subordinate_List = 2, ARRAY INDEX = 0
2.  VERIFY Subordinate_List = 2, ARRAY INDEX = 0
3.  VERIFY Subordinate_Annotations = 2, ARRAY INDEX = 0
4.  WRITE Subordinate_List = (L1, some value greater than 2), ARRAY INDEX=0
5.  VERIFY Subordinate_List = L1, ARRAY INDEX = 0
6.  REPEAT X = (values greater than 2 up to L1) DO {
        VERIFY (Instance Number within the Subordinate_List objectIdentifier) = 4194303, ARRAY INDEX = X
    }
7.  VERIFY Subordinate_Annotations = L1, ARRAY INDEX = 0
8.  WRITE Subordinate_List = (Subordinate_List array of length 2)
9.  VERIFY Subordinate_List = 2, ARRAY INDEX = 0
10. VERIFY Subordinate_Annotations = 2, ARRAY INDEX = 0
11. WRITE Subordinate_List = (Subordinate_List array of length L2 greater than 2)
12. VERIFY Subordinate_List = L2, ARRAY INDEX = 0
13. VERIFY Subordinate_Annotations = L2, ARRAY INDEX = 0
14. WRITE Subordinate_List = 2, ARRAY INDEX = 0
15. VERIFY Subordinate_List = (an array consisting of elements 1 & 2 from the array written in step 11)
16. VERIFY Subordinate_Annotations = 2, ARRAY INDEX = 0
```

7.3.2.X.2 Subordinate_Annotations Size Changes Subordinate_List

Dependencies: WriteProperty Service Execution Tests, 9.22.

BACnet Reference Clauses: 12.29.7 and 12.29.8.

Purpose: This test case verifies that when the size of the Subordinate_Annotations array is changed, the size of the Subordinate_List array is changed accordingly to the same size. In addition, the test case verifies that the Instance Number in any uninitialized objectIdentifiers in the Subordinate_List has the value 4194303. If the size of the Subordinate_Annotations and Subordinate_List arrays cannot be changed, then this test shall not be performed.

Test Concept: The Subordinate_Annotations and Subordinate_List arrays are set to a certain size. They are then increased by writing the Subordinate_Annotations array element 0, decreased by writing the Subordinate_Annotations array, increased by writing the Subordinate_Annotations array, and decreased by writing the Subordinate_Annotations array element 0.

Configuration Requirements: The IUT shall be configured with a Structured View object with resizable Subordinate_Annotations and Subordinate_List arrays.

Test Steps:

```
1.  WRITE Subordinate_Annotations = 2, ARRAY INDEX = 0
2.  VERIFY Subordinate_Annotations = 2, ARRAY INDEX = 0
3.  VERIFY Subordinate_List = 2, ARRAY INDEX = 0
4.  WRITE Subordinate_Annotations = (L1, some value greater than 2), ARRAY INDEX=0
5.  VERIFY Subordinate_Annotations = L1, ARRAY INDEX = 0
6.  REPEAT X = (values greater than 2 up to L1) DO {
        VERIFY (Instance Number within the Subordinate_List objectIdentifier) = 4194303, ARRAY INDEX = X
    }
7.  VERIFY Subordinate_List = L1, ARRAY INDEX = 0
8.  WRITE Subordinate_Annotations = (Subordinate_Annotations array of length 2)
9.  VERIFY Subordinate_Annotations = 2, ARRAY INDEX = 0
10. VERIFY Subordinate_List = 2, ARRAY INDEX = 0
11. WRITE Subordinate_Annotations = (Subordinate_Annotations array of length L2 greater than 2)
12. VERIFY Subordinate_Annotations = L2, ARRAY INDEX = 0
13. VERIFY Subordinate_List = L2, ARRAY INDEX = 0
14. REPEAT X = (values greater than 2 up to L2) DO {
        VERIFY (Instance Number within the Subordinate_List objectIdentifier) = 4194303, ARRAY INDEX = X
    }
15. WRITE Subordinate_Annotations = 2, ARRAY INDEX = 0
16. VERIFY Subordinate_Annotations = (an array consisting of elements 1 & 2 from the array written in step 11)
17. VERIFY Subordinate_List = 2, ARRAY INDEX = 0
```

135.1-2009l-17. Correct AddListElement Test.

Rationale

The test incorrectly expects the First Failed Element parameter be set to 0. Since the first element is the item that failed, First Failed Element should be 1.

[Change **Clause 9.14.2.2**, p. 245]

9.14.2.2 Adding a List Element With an Invalid Datatype

Purpose: To verify the ability of the IUT to correctly respond to an AddListElement service request to add an element with an invalid datatype to a list.

Test Steps:

1. TRANSMIT AddListElement-Request,
   'Object Identifier' = L,
   'Property Identifier' = ListProp,
   'List of Elements' = (a single element with a datatype inappropriate for this property)
2. RECEIVE AddListElement-Error,
   Error Class = PROPERTY,
   Error Code = INVALID_DATATYPE,
   'First Failed Element' = ~~0~~*1*

135.1-2009l-18. Add ReadRange Test.

Rationale

No test exists that verifies the use of a negative count where the requested number of entries will not fit in a single response.

[Add new **Clause 9.21.1.X**, p. 266]

9.21.1.X Reading Items with Negative Count and MOREITEMS

Purpose: To verify that the IUT correctly responds to a ReadRange service request by returning the correct subset of items when a sequence number or byTime and a negative count are requested, and the count is more items than the IUT actually returns.

Test Concept: A ReadRange request is transmitted by the TD requesting a range of items known to be in the Log_Buffer. This range is specified using a negative value for 'Count.' The TD shall be configured such that its Max APDU Length Accepted, Segmented Response Accepted, in combination with the chosen 'Count' selected, mean that the results cannot be conveyed in a single ReadRange-Ack, thus forcing the MOREITEMS flag to be TRUE in the response.

Test Steps:

1. TRANSMIT ReadRange-Request,

'Object Identifier' =	(the logging object configured for this test),
'Property Identifier' =	Log_Buffer,
'Reference Sequence Number' =	(any value x: known to be in the Log_Buffer),
'Count' =	(any value y: y < 0 and which forces the MOREITEMS flag TRUE in the response)

2. RECEIVE ReadRange-ACK,

'Object Identifier' =	(the logging object configured for this test),
'Property Identifier' =	Log_Buffer,
'Result Flags' =	{?, ?, TRUE},
'Item Count' =	(any value z: 0 < z < |y|),
'Item Data' =	(the specified z records in order of increasing sequence number. The items specified are all items in the range of (x - z + 1) through x in that order),
'First Sequence Number' =	(x - z + 1)

Notes to Tester: Although expressed as a bySequence exchange, these could alternately be expressed byTime.

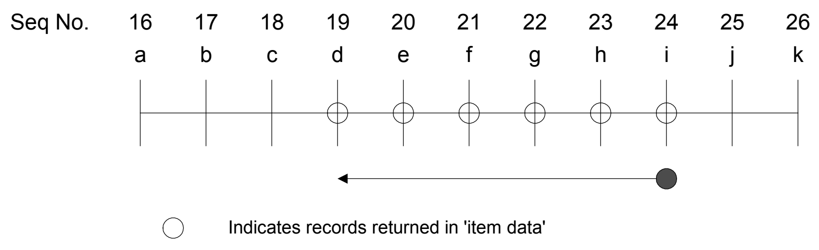
Test Example (using sample buffer shown in diagram below):

1. TRANSMIT ReadRange-Request,

'Object Identifier' =	20:1,
'Property Identifier' =	Log_Buffer,
'Reference Sequence Number' =	24,
'Count' =	-9

2. RECEIVE ReadRange-ACK,

'Object Identifier' =	20:1,
'Property Identifier' =	Log_Buffer,
'Result Flags' =	{FALSE, FALSE, TRUE},
'Item Count' =	6,
'Item Data' =	Records < d, e, f, g, h, i > in that order.
'First Sequence Number' =	19


Seq No.
16 17 18 19 20 21 22 23 24 25 26
a b c d e f g h i j k
○ Indicates records returned in 'item data'

135.1-2009l-19. Remove Reliance on EPICS from DCC Test.

Rationale

Remove the test's reliance on the EPICS.

[Change **Clause 9.24.2.1**, p. 283]

9.24.2.1 Invalid Password

Purpose: To verify the correct execution of DeviceCommunicationControl service procedure when an invalid password is provided. If the IUT does not provide password protection, this test case shall be omitted.

Test Steps:

```
1.  TRANSMIT DeviceCommunicationControl-Request,
        'Enable/Disable' =    DISABLE,
        'Password' =          (any invalid password)
2.  RECEIVE BACnet-Error-PDU,
        Error Class =         SECURITY,
        Error Code =          PASSWORD_FAILURE
3.  VERIFY (Device, S), *System_Status = (any valid value)*
        ~~(any required non-array property) = (the value for this property specified in the EPICS)~~
```

135.1-2009l-20. Add Who-Has Tests.

Rationale

Add tests that ensure that changes to Object_Names and Object_Identifiers are correctly reflected in I-Have messages sent in response to Who-Has requests.

[Add new **Clauses 9.32.1.X1 and 9.32.1.X2**, p. 294]

9.32.1.X1 Who-Has After Object_Name Changed

Dependencies: Who-Has Service Execution Tests, 9.32.1.2

BACnet Reference Clause: 16.9

Purpose: To verify that a device correctly responds to Who-Has service requests after the Object_Name property of an object in the device is changed.

Test Concept: The Object_Name property of the referenced object is read to determine its initial value. The Object_Name property is then changed to a different value, V2, which is not already used by an object in the IUT. The test then verifies correct responses to Who-Has requests that include an 'Object Name' parameter, using the values V1 and V2.

Configuration: An object, O1, exists within the IUT that has a modifiable Object_Name property and has the value V1. If IUT does not support objects with modifiable Object_Name properties, then this test shall be skipped.

Test Steps:

```
1.  READ V1 = O1, Object_Name
2.  IF (Object_Name is writable) THEN
        WRITE O1, Object_Name = V2
    ELSE
        MAKE (O1, Object_Name = V2)
3.  TRANSMIT
        DESTINATION =          GLOBAL BROADCAST,
        Who-Has-Request,
        'Object Name' =        V1
4.  WAIT Internal Processing Fail Time
5.  CHECK (Verify that the IUT does not respond with an I-Have request)
6.  TRANSMIT
        DESTINATION =          GLOBAL BROADCAST,
        Who-Has-Request,
        'Object Name' =        V2
7.  RECEIVE
        DESTINATION =          LOCAL BROADCAST | GLOBAL BROADCAST,
        I-Have-Request,
        'Device Identifier' =  (the IUT's Device object),
        'Object Identifier' =  O1,
        'Object Name' =        V2
```

9.32.1.X2 Who-Has After Object_Identifier Changed

Dependencies: Who-Has Service Execution Tests, 9.32.1.1

BACnet Reference Clause: 16.9

Purpose: To verify that a device correctly responds to Who-Has service requests after the Object_Identifier property of an object in the device is changed.

Test Concept: The Object_ Identifier property of the referenced object, O1, is verified to contain the value O1. The Object_ Identifier property is then changed to a different value, O2, which is not already in use by a different object in the IUT. The test then verifies correct responses to Who-Has requests that include an 'Object Identifier' parameter, using the values O1 and O2.

Configuration: An object, O1, exists within the IUT that has a modifiable Object_ Identifier property. If the IUT does not support objects with modifiable Object_Identifiers, then this test shall be skipped.

Test Steps:

```
1.  VERIFY O1, Object_Identifier = O1
2.  IF (O1 is writable) THEN
        WRITE O1, Object_Identifier = O2
    ELSE
        MAKE (O1, Object_Identifier = O2)
3.  TRANSMIT
        DESTINATION =            GLOBAL BROADCAST,
        Who-Has-Request,
        'Object Identifier' =    O1
4.  WAIT Internal Processing Fail Time
5.  CHECK (Verify that the IUT does not respond with an I-Have request)
6.  TRANSMIT
        DESTINATION =            GLOBAL BROADCAST,
        Who-Has-Request,
        'Object Identifier' =    O2
7.  RECEIVE
        DESTINATION =            LOCAL BROADCAST | GLOBAL BROADCAST,
        I-Have-Request,
        'Device Identifier' =    (the IUT's Device object),
        'Object Identifier' =    O2
```

135.1-2009l-21. Correct Unknown Network Layer Message Type For Someone Else Test.

Rationale

The specifications of the proprietary network messages are incorrect in the expected responses.

[Change **Clause 10.2.2.7.3**, p. 324]

10.2.2.7.3 Unknown Network Layer Message Type For Someone Else

Purpose: This test case verifies that the IUT will not reject a network layer message with an unknown message type when it is destined elsewhere. This test shall not be run if the IUT does not have a Protocol_Revision property or its value is less than 4.

BACnet Reference Clause: 6.6.3.5

Test Steps:

1. TRANSMIT PORT A, DA = IUT, SA = D1A,
   DNET = 2,
   DADR = D2C,
   Hop Count = 255,
   Message Type = (any value in the range reserved for use by ASHRAE)
2. RECEIVE Port B, DA = D2C, SA = IUT,
   SNET = 1,
   SADR = D1A,
   Message Type = (value from step 1)
3. TRANSMIT PORT A, DA = LOCAL BROADCAST, SA = D1A,
   DNET = GLOBAL BROADCAST,
   DLEN = 0,
   Hop Count = 255,
   Message Type = (any value in the range reserved for use by ASHRAE)
4. RECEIVE PORT B, DA = LOCAL BROADCAST, SA = IUT,
   DNET = GLOBAL BROADCAST,
   DLEN = 0,
   SNET = 1,
   SADR = D1A,
   Hop Count = (any value greater than 1 and less than 255),
   Message Type = (value from step 3)
5. TRANSMIT PORT A, DA = IUT, SA = D1A,
   DNET = 2,
   DADR = D2C,
   Hop Count = 255,
   Message Type = (any value in the range available for vendor proprietary messages),
   Vendor ID = (any value, when paired with Message Type, that is not supported by the IUT)
6. RECEIVE Port B, DA = D2C, SA = IUT,
   SNET = 1,
   SADR = D1A,
   Message Type = (value from step ~~1~~*5*)
   Vendor ID = (value from step 5)
7. TRANSMIT PORT A, DA = LOCAL BROADCAST, SA = D1A,
   DNET = GLOBAL BROADCAST,
   DLEN = 0,
   Hop Count = 255,
   Message Type = (any value in the range available for vendor proprietary messages),

Vendor ID =	(any value, when paired with Message Type, that is not supported by the IUT)

8. RECEIVE PORT B, DA = LOCAL BROADCAST, SA = IUT,

DNET =	GLOBAL BROADCAST,
DLEN =	0,
SNET =	1,
SADR =	D1A,
Hop Count =	(any value greater than 1 and less than 255),
Message Type =	(value from step 7)
Vendor ID =	*(value from step 7)*

135.1-2009l-22. Clarify TRANSMIT And RECEIVE Addressing Information.

Rationale

The TRANSMIT and RECEIVE statement definitions do not allow for BVLL, NPDU, LPDU, and MPDU fields although the standard uses them.

[Change **Clause 6.2.6**, p. 26]

6.2.6 TRANSMIT Statement

The TRANSMIT statement is used to transmit a packet.

```
<transmit statement> ::=    TRANSMIT <packet desc>
```

Where:

```
<packet desc> ::=           [ <port> ',' ] [ <addressing> ',' ] ( <service specification> | <pdu specification> |
                                                                   <string>)

<port> ::=                  PORT <port identifier>
<port identifier> ::=       'A' | 'B' ... 'Z'
<addressing> ::=            ( <dst> | <src> | <dst> ',' <src> )
<src> ::=                   ( SOURCE '=' <src parm value> ) | <src parm list>
<src parm value> ::=        TD | IUT | <device>
<src parm list>::=          (any src addr parameter name) '=' ( <src parm value> | <addr parm value>)
                                  [ ',' <src parm list> ]
<dst> ::=                   ( DESTINATION '=' <dst parm value> ) | <dst parm list>
<dst parm value> ::=        LOCAL BROADCAST | GLOBAL BROADCAST
                                  | REMOTE BROADCAST <net>
                                  | IUT | TD
                                  | <device>
<dst parm list>::=          (any dst addr parameter name) '=' ( <dst parm value> | <addr parm value>)
                                  [ ',' <dst parm list> ]
<addr parm value> ::=       (any media specific address description)
<device> ::=                '( device ' (device instance) ')' | (a valid device description or variable)
<net> ::=                   (a valid BACnet network number, description or variable)

<service specification> ::= <BACnet service> [ ',' <pdu parm list> ] [ ',' <service parm list> ]
<BACnet service> ::=        (any BACnet service choice)
<service parm list> ::=     <service parameter> '=' <parameter value> [ ',' <service parm list> ]
<service parameter> ::=     (parameter name specific to the BACnet service)

<pdu specification> ::=     <pdu type> [ ',' <pdu parm list> ]
<pdu type> ::=              (any BACnet application, network, link, or MAC layer PDU type)
<pdu parm list> ::=         <pdu parameter> '=' <parameter value> [ ',' <pdu parm list> ]
<pdu parameter> ::=         (any BACnet application, network, data link, or MAC layer PDU
                            parameter)

<parameter value> ::=       ( <atomic value> | <parameter value list> | <parameter cond value> )
<parameter value list> ::=  '(' <parameter value> [ '|' <parameter value> ]... ')'
<parameter cond value> ::= '(' IF <condition> THEN <parameter value> ELSE <parameter value> ')'

<string> ::=                <"> <ASCII Char> <">
<ASCII Char> ::=            (ANSI X3.4 character)
```

The SOURCE and DESTINATION parameters are used to briefly specify common combinations of NPDU, LPDU and MPDU parameter values. ~~If DESTINATION and SOURCE are not specified, the source address shall be TD and the destination address shall be IUT.~~ *If no source addressing information is provided, then the source address shall be TD. If no destination addressing information is provided, then the destination address shall be IUT.*

A list of <pdu parameter> = <parameter value> pairs indicate that the specified parameters shall convey the indicated values. The parameter values may be specified in any order.

A list of <service parameter> = <parameter value> pairs indicate that the specified parameters shall convey the indicated values. The parameter values may be specified in any order.

Example 1:

```
TRANSMIT DESTINATION = GLOBAL BROADCAST, Who-Is
```

In this simple case, the Who-Is service does not have any mandatory parameters and the <pdu type> is known to be a BACnet-Unconfirmed-Request-PDU by definition. The DESTINATION implies parameter values in the NPDU, LPDU and MPDU layers. The following statement is identical, but more completely specified:

```
TRANSMIT
    DA = LOCAL BROADCAST,
    SA = TD,
    DNET = GLOBAL BROADCAST,
    BACnet-Unconfirmed-Request-PDU,
    'Service Choice' = Who-Is
```

Example 2:

```
TRANSMIT ReadProperty-Request,
    'Object Identifier' = (Analog Input,1),
    'Property Identifier' = Present_Value
```

In this case a ReadProperty service request will be sent from the TD to the IUT with the specified service parameter values.

[Change **Clause 6.2.7**, p. 28]

6.2.7 RECEIVE Statement

The RECEIVE procedure is used to define a message from the IUT.

```
<receive statement> ::= RECEIVE ( <packet desc> | '(' <packet desc> ')' [ '|' '(' <packet desc> ')' ] ...)
```

The <pdu specification> parameter is the same as used in the TRANSMIT statement. If unspecified, the ~~SOURCE~~ *source addressing information* defaults to IUT and ~~DESTINATION~~ *destination addressing information* defaults to TD.

```
Example:    TRANSMIT  SubscribeCOV-Request,
                          'Subscriber Process Identifier' =     any value selected by the TD,
                          'Monitored Object Identifier' =       any object supporting COV notification,
                          'Issue Confirmed Notifications' =     TRUE,
                          'Lifetime' =                          300
            RECEIVE ConfirmedCOVNotification-Request,
                          'Subscriber Process Identifier' =     the value from the previous subscription,
                          'Monitored Object' =                  the value from the previous subscription,
                          'Initiating Device Identifier' =      IUT,
```

'Lifetime' =	*30*0,
'List of Values' =	values appropriate to the object type of the monitored object

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

2/12