



# ADDENDA

**ANSI/ASHRAE Addendum s to  
ANSI/ASHRAE Standard 135.1-2019**

# Method of Test for Conformance to BACnet®

Approved by ASHRAE and the American National Standards Institute on March 31, 2023.

This addendum was approved by a Standing Standard Project Committee (SSPC) for which the Standards Committee has established a documented program for regular publication of addenda or revisions, including procedures for timely, documented, consensus action on requests for change to any part of the standard. Instructions for how to submit a change can be found on the ASHRAE® website ([www.ashrae.org/continuous-maintenance](http://www.ashrae.org/continuous-maintenance)).

The latest edition of an ASHRAE Standard may be purchased on the ASHRAE website ([www.ashrae.org](http://www.ashrae.org)) or from ASHRAE Customer Service, 180 Technology Parkway, Peachtree Corners, GA 30092. E-mail: [orders@ashrae.org](mailto:orders@ashrae.org). Fax: 678-539-2129. Telephone: 404-636-8400 (worldwide), or toll free 1-800-527-4723 (for orders in US and Canada). For reprint permission, go to [www.ashrae.org/permissions](http://www.ashrae.org/permissions).

© 2023 ASHRAE

ISSN 1041-2336



**ASHRAE Standing Standard Project Committee 135**

**Cognizant TC: 1.4, Control Theory and Application**

**SPLS Liaison: William Bahnfleth**

Coleman L. Brumley, Jr.\*, *Chair*  
Scott Ziegenfus, *Vice-Chair*  
Nate Benes\*  
Steven T. Bushby\*  
Salvatore Cataldi\*  
James F. Butler  
Tyler Cove  
Brandon M. DuPrey\*

David Fisher  
Alexander Gurciullo\*  
Bernhard Isler  
Thomas Kurowski\*  
Shahid Naeem  
Frank Victor Neher\*  
Carl Neilson\*  
Michael Osborne

Scott Reed  
David Robin\*  
Frank Schubert  
Marcelo R. da Silva  
Ted Sunderland  
Lori Tribble  
Grant N. Wichenko\*  
Christoph Zeller

\* Denotes members of voting status when the document was approved for publication

---

**ASHRAE STANDARDS COMMITTEE 2022–2023**

Susanna S. Hanson, *Chair*  
Jonathan Humble, *Vice-Chair*  
William P. Bahnfleth  
Thomas E. Cappellin  
Douglas D. Fick  
Patricia Graef  
Jaap Hogeling  
Jennifer A. Isenbeck  
Phillip A. Johnson  
Srinivas Katipamula

Gerald J. Kettler  
Jay A. Kohler  
Cesar L. Lim  
Paul A. Lindahl, Jr.  
James D. Lutz  
Julie Majurin  
Lawrence C. Markel  
Patrick C. Marks  
Margaret M. Mathison  
Kathleen Owen

Gwelen Paliaga  
Karl L. Peterman  
Justin M. Prosser  
David Robin  
Christopher J. Seeton  
Christian R. Taber  
Paolo M. Tronville  
William F. Walter  
Steven C. Sill, *BOD ExO*  
Sarah E. Maston, *CO*

Connor Barbaree, *Senior Manager of Standards*

---

**SPECIAL NOTE**

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

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

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

The Senior Manager of Standards of ASHRAE should be contacted for

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

**DISCLAIMER**

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

**ASHRAE INDUSTRIAL ADVERTISING POLICY ON STANDARDS**

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

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

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

## FOREWORD

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

- 135.1-2019s-1    Add new and correct existing tests for the Network Layer, p. 3**
- 135.1-2019s-2    Add new and correct existing tests for the MS/TP Data Link Layer, p. 15**
- 135.1-2019s-3    Add tests for the BACnet Secure Connect Data Link Layer, p. 17**
- 135.1-2019s-4    Add tests for IPv6, p. 68**
- 135.1-2019s-5    Add new and correct existing BACnet/IP Functionality Tests, p. 84**
- 135.1-2019s-6    Renumber Clause 15, p. 107**
- 135.1-2019s-7    Add new and correct existing Object Support Tests, p. 108**
- 135.1-2019s-8    Add new and correct existing Application Service Initiation Tests, p. 335**
- 135.1-2019s-9    Add new and correct existing Application Service Execution Tests, p. 419**
- 135.1-2019s-10   Improve existing DEFINITIONS, p. 562**
- 135.1-2019s-11   Improve the EPICS FILE FORMAT and the EPICS CONSISTENCY TESTS, p. 563**
- 135.1-2019s-12   Add new and improve existing CONVENTIONS FOR SPECIFYING BACnet CONFORMANCE TESTS, p. 565**
- 135.1-2019s-13   Add new and improve existing SPECIAL FUNCTIONALITY TESTS, p. 568**
- 135.1-2019s-14   Apply miscellaneous editorial changes, p. 587**

In the following document, language to be added to existing clauses of ANSI/ASHRAE 135.1-2019 and Addenda is indicated through the use of *italics*, while deletions are indicated by ~~striketrough~~. Where entirely new subclauses are proposed to be added, plain type is used throughout. Only this new and deleted text is open to comment as this time. All other material in this document is provided for context only and is not open for public review comment except as it relates to the proposed changes.

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

## 135.1-2019s-1 Add new and correct existing tests for the Network Layer

### Rationale

Errors have been identified in a number of router and non-router functionality tests in ANSI/ASHRAE Standard 135.1-2019. In addition, the test coverage shall be increased by new tests.

[Change **Clause 10.1.1**, p. 457]

[Reviewer Note: The test assumes that the IUT and the TD are located on the same network. For the IUT, the TD appears to be the appropriate router to the network specified in step 1.]

### 10.1.1 Processing Application Layer Messages Originating from Remote Networks

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

~~BACnet Reference Clause: 6.5.4.~~

Purpose: To verify that the IUT can respond to requests that originate from a remote network.

Test Concept: The TD transmits a ReadProperty-Request message that contains network layer information indicating that it originated from a remote network. The response from the IUT shall include correct DNET and DADR information so that the message can reach the original requester. The MAC layer destination address in the response can be either a *local* broadcast, indicating that the IUT does not know the address of the router, or the MAC address of the ~~appropriate router~~ *TD*.

Test Steps:

1. TRANSMIT  
DESTINATION = IUT,  
SOURCE = TD,  
SNET = (any network number that is not the local network),  
SADR = (any valid MAC address consistent with the source network),  
ReadProperty-Request,  
'Object Identifier' = (any supported object),  
'Property Identifier' = (any required property of the specified object)
2. RECEIVE  
DESTINATION = LOCAL BROADCAST | ~~(an appropriate router address)~~ *TD*,  
SOURCE = IUT,  
DNET = (the SNET specified in step 1),  
DADR = (the SADR specified in step 1),  
Hop Count = 255,  
ReadProperty-ACK,  
'Object Identifier' = (the object specified in step 1),  
'Property Identifier' = (the property specified in step 1),  
'Property Value' = (any valid value *for this property*)

[Change **Clause 10.2.2.7.2**, p. 472]

[Reviewer Note: Corrected the name of the reject reason parameter in the Reject-Message-To-Network specification. The correct name is 'Rejection Reason', not 'Reject Reason'.]

### 10.2.2.7.2 Unknown Network Layer Message Type

Purpose: To verify that the IUT will reject a network layer message with an unknown message type in the range of message types reserved for use by ASHRAE.

Test Steps:

1. TRANSMIT PORT A,  
     DESTINATION = IUT,  
     SOURCE = D1A,  
     Message Type = (any value in the range reserved for use by ASHRAE that is undefined in the protocol revision claimed by the device)
2. RECEIVE PORT A,  
     DESTINATION = ~~D1A~~,  
     SOURCE = IUT,  
     Reject-Message-To-Network,  
     Rejection Reason = 3 (unknown network layer message type),  
     DNET = (any value)

### 10.2.3 Routing of Unicast APDUs

[Change **Clause 10.2.3.2**, p. 474]

[Reviewer Note: Modified test to support new Extended MSTP Frame testing.]

#### 10.2.3.2 Route Message from a Local Device to a Local Device

Purpose: To verify that the IUT can route a *maximum sized NPDU* unicast message from a local device on Network 1 to a device on Network 2.

*Test Concept: A message is sent from Network 1 destined for Network 2 via the IUT, and router functionality is verified by observing the message on Network 2. The sequence is repeated in the opposite direction to verify the IUT can also route messages from Network 2 to Network 1. The messages may have an NPDU length, L, such that L equals the Maximum NPDU length as defined in Table 6-1 for the smaller data link.*

Test Steps:

1. TRANSMIT PORT A,  
     DA = IUT,  
     SA = D1A,  
     DNET = 2,  
     DADR = D2C,  
     Hop Count = 255,  
     BACnet-Confirmed-Request-PDU,  
     'Service Choice' = ~~ReadProperty-Request~~ WriteProperty-Request,  
     'Object Identifier' = (OI = any object identifier),  
     'Property Identifier' = (PI = any property of the specified object with a *CharacterString* datatype),  
     'Property Value' = (V = *CharacterString* with a length such that the NPDU length = L)
2. RECEIVE PORT B,  
     DA = D2C,  
     SA = IUT,  
     SNET = 1,  
     SADR = D1A,  
     BACnet-Confirmed-Request-PDU,  
     'Service Choice' = ~~ReadProperty-Request~~ WriteProperty-Request,  
     'Object Identifier' = OI (any object identifier),  
     'Property Identifier' = PI (any property of the specified object),  
     'Property Value' = V
3. TRANSMIT PORT B,  
     DA = IUT,  
     SA = D2C,  
     DNET = 1,  
     DADR = D1A,  
     Hop Count = 255,  
     BACnet-Confirmed-Request-PDU,

'Service Choice' = ~~ReadProperty-Request~~ WriteProperty-Request,  
 'Object Identifier' = (OI = any object identifier),  
 'Property Identifier' = (PI = any property of the specified object with a *CharacterString* datatype),  
 'Property Value' = (V = *CharacterString* with a length such that the NPDU length = L)

4. RECEIVE PORT A,  
     DA = D1A,  
     SA = IUT,  
     SNET = 2,  
     SADR = D2C,  
     BACnet-Confirmed-Request-PDU,  
     'Service Choice' = ~~ReadProperty-Request~~ WriteProperty-Request,  
     'Object Identifier' = OI (any object identifier),  
     'Property Identifier' = PI (any property of the specified object),  
     'Property Value' = V

### 10.2.3.6 Attempt to Locate Downstream Routers

[Change **Clause 10.2.3.6.1**, p. 476]

#### 10.2.3.6.1 Failed Attempt to Locate Router

Purpose: To verify that the IUT will attempt to locate a router to an unknown network. Upon failing to locate such a router the IUT will transmit a Reject-Message-To-Network to the source device.

*Configuration Requirements: The IUT shall be configured to know only about its directly connected networks.*

*T<sub>Omax</sub>: vendor defined time the router waits before sending a Reject-Message-to-Network request.*

*T<sub>Omin</sub>: vendor defined minimum time the router waits for a response to the Who-Is-Router-To-Network request.*

*Notes to Tester: The standard does not provide any guidance on how long a router should wait before declaring that the attempt to locate the next router failed. While there is no explicit minimum time, it is expected that routers wait long enough that the attempt would succeed if the next hop router responded immediately.*

Test Steps:

1. TRANSMIT PORT A, DA = IUT,  
     SA = R1-5, DNET = 3,  
     DADR = D3D,  
     SNET = 5,  
     SADR = D5F,  
     Hop Count = 254,  
     BACnet-Confirmed-Request-PDU,  
     'Service Choice' = ReadProperty-Request,  
     'Object Identifier' = (any object identifier),  
     'Property Identifier' = (any property of the specified object)
2. RECEIVE PORT B,  
     DESTINATION = LOCAL BROADCAST, SOURCE = IUT, Who-Is-Router-To-Network,  
     Network Number = 3
3. WAIT T<sub>Omin</sub>
4. BEFORE T<sub>Omax</sub>  
     RECEIVE PORT A, DA = R1-5,  
     SOURCE = IUT,  
     DNET = 5,  
     DADR = D5F,  
     Hop Count = 255,  
     Reject-Message-To-Network,

Rejection Reason = 1 (unknown destination network), DNET = 3

[Change **Clause 10.2.3.6.2**, p. 477]

#### **10.2.3.6.2 Successful Attempt to Locate Router**

Purpose: To verify that the IUT will attempt to locate a router to an unknown network. When successful it forwards the message to the next router on the path.

*Configuration Requirements: The IUT shall be configured to know only about the directly-connected networks.*

*T<sub>Omin</sub>: vendor defined minimum time the router waits for a response to the Who-Is-Router-To-Network request.*

*Notes to Tester: The standard does not provide any guidance on how long a router should wait before declaring that the attempt to locate the next router failed. While there is no explicit minimum time, it is expected that routers wait long enough that the attempt would succeed if the next hop router responded immediately.*

Test Steps:

1. TRANSMIT PORT A, DA = IUT,  
SA = R1-5,  
DNET = 3,  
DADR = D3D,  
SNET = 5,  
SADR = D5F,  
Hop Count = 254,  
BACnet-Confirmed-Request-PDU,  
'Service Choice' = ReadProperty-Request,  
'Object Identifier' = (any object identifier),  
'Property Identifier' = (any property of the specified object)
2. RECEIVE PORT B,  
DESTINATION = LOCAL BROADCAST, SOURCE = IUT, Who-Is-Router-To-Network,  
Network Number = 3
3. *WAIT any time less than T<sub>Omin</sub>*
4. TRANSMIT PORT B,  
DESTINATION = LOCAL BROADCAST, SOURCE = R2-3, I-Am-Router-To-Network,  
Network Numbers = 3
5. RECEIVE PORT B, SA = R2-3,  
SA = IUT, DNET = 3, DADR = D3D,  
SNET = 5,  
SADR = D5F,  
Hop Count = (any integer x: 0 < x < 254), BACnet-Confirmed-Request-PDU,  
'Service Choice' = ReadProperty-Request,  
'Object Identifier' = (the object identifier used in step 1), 'Property Identifier' = (the property identifier used in step 1)

[Add new **Clause 10.2.X1**, p. 483]

[Reviewer Note: Added missing test as per 135-2008g (protocol revision 11).]

#### **10.2.X1 Initiates Network-Number-Is on Startup**

Purpose: To verify that a router initiates Network-Number-Is on startup for each port with a known network number.

Test Concept: The IUT is reset and the tester verifies that the IUT broadcasts a Network-Number-Is message out each port. The vendor can specify a time, or physically observable event after reset, which marks the time at which IUT has completed its startup sequence, including the sending of the Network-Number-Is messages.

Configuration Requirements: The IUT is configured with a network number for each of its enabled ports. If the IUT claims a protocol revision of less than 11, this test shall be skipped.

Test Steps:

1. MAKE (the IUT reset)
2. BEFORE the IUT has completed its startup sequence  
    REPEAT X = (for each enabled port) DO {  
        RECEIVE PORT X,  
        DESTINATION = LOCAL BROADCAST,  
        Network-Number-Is,  
        Network Number = (the configured Network Number for port X)  
    }

[Add new **Clause 10.2.X2**, p. 483]

[Reviewer Note: Added missing test as per 135-2008g (protocol revision 11)]

### **10.2.X2 Routers Execute What-Is-Network-Number**

Purpose: To verify that a router responds to a What-Is-Network-Number request within 10 seconds.

Test Concept: A What-Is-Network-Number is broadcast on the local network and the tester verifies that the IUT responds with a Network-Number-Is message within 10 seconds.

Configuration Requirements: The IUT knows its network number, N1. If the IUT claims a protocol revision of less than 11, this test shall be skipped.

Test Steps:

1. TRANSMIT What-Is-Network-Number,  
    DESTINATION = LOCAL\_BROADCAST
2. BEFORE 10s + Internal Processing Fail Time  
    RECEIVE Network-Number-Is,  
    Network Number = (the configured value),  
    Configured =(any valid value)

## **10.5 Initiating Network Layer Messages**

### **10.5.2 Managing Router Tables**

[Add new **Clause 10.5.2.X1**, p. 493]

#### **10.5.2.X1 Query A Router's Known Routes**

Purpose: To verify that the IUT can query a router to determine which routes are accessible through it.

Test Concept: Make the IUT query the router, to determine their routes and verify that the IUT conveys the information to the user.

Configuration: The TD is configured as a router which does not support Network Port objects (Protocol\_Revision < 17).

Test Steps:

1. MAKE (the IUT initiate an Who-Is-Router-To-Network message)
2. RECEIVE Who-Is-Router-To-Network  
    DESTINATION = TD | LOCAL\_BROADCAST



3. TRANSMIT I-Am-Router-To-Network,  
Network Numbers = (L: a valid list of networks)
4. CHECK (the IUT presents the router's known routes)

## 10.6 Non-Router Functionality Tests

[Change **Clause 10.6.3**, p. 495]

[Reviewer Note: Changed test to support a Reject or a discard.]

### 10.6.3 Ignore Router Commands

~~BACnet Reference Clause: 6.6, 6.6.3.8, 6.6.3.10, 6.6.3.11~~

Purpose: This test case verifies that the non-router IUT will *either* quietly accept and discard network layer router services *or respond with a Reject-Message-To-Network*.

Test Concept: The TD transmits the Initialize-Routing-Table, Establish-Connection-To-Network, and Disconnect-Connection-To-Network services. ~~The IUT is required to silently drop the requests because it is not a router.~~

Test Steps:

1. TRANSMIT  
DA = IUT,  
SA = TD,  
Initialize-Routing-Table  
Number of Ports = 0
2. WAIT **Internal Processing Fail Time**
3. (CHECK (that the IUT did not send any packets in response to the Initialize-Routing-Table)) |  
(RECEIVE  
DESTINATION = TD,  
SOURCE = IUT,  
Reject-Message-To-Network  
DNET = (any valid value)  
Rejection Reason = 0 (other) | 3 (unknown))
4. TRANSMIT  
DA = IUT,  
SA = TD,  
Establish-Connection-To-Network  
DNET = DNET3  
Termination Time Value = 0
5. WAIT **Internal Processing Fail Time**
6. (CHECK (that the IUT did not send any packets in response to the Establish-Connection-To-Network)) |  
(RECEIVE  
DESTINATION = TD,  
SOURCE = IUT,  
Reject-Message-To-Network  
DNET = (any valid value)  
Rejection Reason = 0 (other) | 3 (unknown))
7. TRANSMIT  
DA = IUT,  
SA = TD,  
Disconnect-Connection-To-Network  
DNET = NET3
8. WAIT Internal Processing Fail Time

9. (CHECK(that the IUT did not send any packets in response to the Disconnect-Connection-To-Network)) |  
(RECEIVE  
     *DESTINATION* = *TD*,  
     *SOURCE* = *IUT*,  
     *Reject-Message-To-Network*  
     *DNET* = (any valid value)  
     *Rejection Reason* = 0 (other) | 3 (unknown))

## 10.7 Router Functionality

[Change **Clause 10.7.2**, p. 497]

[Reviewer Note: Test modified to allow for directed unicast who-is requests.]

### 10.7.2 Router Binding via Application Layer Services

~~Dependencies: ReadProperty Service Initiation Tests, 8.18, ReadProperty Service Execution Tests, 9.18, Who-Is Service Initiation Tests, 8.34~~

~~BACnet Reference Clause: 6.5.3~~

Purpose: To verify that the IUT can initiate requests to a remote network and respond to requests from a remote network after the IUT uses the Who-Is and I-Am Application Layer services to discover the MAC address of the router to that remote network.

Test Concept: The IUT broadcasts a Who-Is request to discover device D2A and notes the MAC address of the intervening router in the corresponding I-Am reply. The TD transmits a request to a device on the remote network and responds to a request from the remote network without performing any further form of dynamic router binding. If the IUT does not support application layer router binding, then this test shall be omitted. If the IUT cannot initiate a ReadProperty request, then another confirmed service can be substituted. The IUT may use the deviceInstanceRange form of Who-Is.

Clause 6.5.3 specifically mentions router binding via Who-Is and does not mention router binding by initiating other application layer services (such as Who-Has) or by lurking and noting the router MAC addresses for incoming application layer requests. For this reason the test only allows for router binding via Who-Is.

Test Steps:

1. MAKE (IUT transmit Who-Is to discover the device on the remote network)
2. RECEIVE
  - DA = BROADCAST,
  - SA = IUT,
  - DNET = GLOBAL BROADCAST,
  - Hop Count = 255,
  - BACnet-Unconfirmed-Request-PDU,
  - 'Service Choice' = ~~who-Is~~Who-Is  
| (DA = BROADCAST,
  - SA = IUT,
  - DNET = DNET2,
  - DADR= BROADCAST, or D2A
  - Hop Count = 255,
  - BACnet-Unconfirmed-Request-PDU,
  - 'Service Choice' = ~~who-Is~~Who-Is)
3. TRANSMIT
  - DA = BROADCAST,
  - SA = TD,
  - SNET = DNET2,
  - SADR = D2A,
  - BACnet-Unconfirmed-Request-PDU,
  - 'Service Choice' = I-Am,

- 'I Am Device Identifier' = (device object, instance number of D2A),
- 'Max APDU Length Accepted' = (any valid value),
- 'Segmentation Supported' = (any valid value),
- 'Vendor ID' = (any valid value)
- 4. MAKE (IUT transmit a ReadProperty request to the D2A device on the remote network)
- 5. RECEIVE
  - DA = TD,
  - SA = IUT,
  - DNET = DNET2,
  - DADR = D2A,
  - Hop Count = 255,
  - BACnet-Confirmed-Request-PDU,
  - 'Service Choice' = ReadProperty-Request,
  - 'Object Identifier' = (O1, any BACnet standard object in D2A),
  - 'Property Identifier' = (P1, any required property of the specified object)
- 6. TRANSMIT
  - DA = IUT,
  - SA = TD,
  - SNET = DNET2,
  - SADR = D2A,
  - BACnet-ComplexACK-PDU,
  - 'Service ACK Choice' = ReadProperty-ACK,
  - 'Object Identifier' = O1,
  - 'Property Identifier' = P1,
  - 'Property Value' = (any valid value)

## 10.8 Virtual Routing Functionality Tests

### 10.8.3 Routing of Unicast APDUs

[Change **Clause 10.8.3.1**, p. 505]

#### 10.8.3.1 Route Request Message from a Local Device to a Virtual Device and Route Response Message from the Virtual Device to the Local Device

Purpose: To verify that the IUT can route a unicast request message from a local device to a virtual device and route the response from the virtual device to the local device.

*Notes to Tester: The destination device (VD1A) can be any virtual device in the IUT.*

Test Steps:

1. TRANSMIT,
  - DA = LOCAL BROADCAST,
  - SA = TD,
  - DNET = 1,
  - DADR = VD1A,
  - Hop Count = 255,
  - ReadProperty-Request,
  - 'Object Identifier' = (the object identifier of any object in the target device),
  - 'Property Identifier' = (any property of the specified object containing a value small enough so that the response will not need to be segmented)
2. RECEIVE,
  - DA = TD,
  - SA = IUT,
  - SNET = 1,
  - SADR = VD1A,
  - ReadProperty-ACK,

- 'Object Identifier' = (the object identifier used in step 1),  
 'Property Identifier' = (the property identifier used in step 1),  
 'Property Value' = (the contents of the specified property)
3. TRANSMIT,  
 DA = IUT,  
 SA = TD,  
 DNET = 1,  
 DADR = VD1A,  
 Hop Count = 255,  
 ReadProperty-Request,  
 'Object Identifier' = (the object identifier of any object in the target device),  
 'Property Identifier' = (any property of the specified object containing a value small enough so that the response will not need to be segmented, but not the same property as in step 1)
4. RECEIVE,  
 DA = TD,  
 SA = IUT,  
 SNET = 1,  
 SADR = VD1A,  
 ReadProperty-ACK,  
 'Object Identifier' = (the object identifier used in step 3),  
 'Property Identifier' = (the property identifier used in step 3),  
 'Property Value' = (the contents of the specified property)

[Change **Clause 10.8.3.2**, p. 506]

### 10.8.3.2 Route Request Message from a Virtual Device to a Local Device

Purpose: To verify that the IUT can route a unicast request message from a virtual device to a local device.

Test Concept: Make one of the virtual devices generate a unicast request, and verify that the NPCI is correctly formed. This test shall be skipped if none of the IUT's virtual devices can issue a confirmed or unconfirmed request in a unicast message.

*Configuration Requirements: The IUT shall be configured or otherwise stimulated so that one of its virtual devices will send a unicast message to a particular target device on Network 2.*

Notes to Tester: During the test, the TD shall answer any requests that the IUT generates while attempting to locate the route to the target device.

*Notes to Tester: This test should be run repeatedly in order to exercise all ways that the IUT can be configured or stimulated to send a unicast message to a device on a local network. Depending on the capabilities of the IUT, this may involve sending a message from the target device to the IUT (unicast or broadcast), writing the network address of the target device to an object property in the IUT, writing the Device ID of the target device to an object property in the IUT, writing the Device Name of the target device to an object property in the IUT, or configuring the IUT using a proprietary method. The IUT may need to broadcast a Who-Is or Who-Has request in order to discover the network address of the target device if the network address is unknown.*

Test Steps:

1. MAKE (the virtual device generate a unicast APDU to a device on the IUT's local network) 2. RECEIVE,  
 DA = TD  
 SA = IUT  
 SNET = 1,  
 SADR = (MAC address of any virtual device on Network 1),  
 BACnet-Confirmed-Request-PDU or BACnet-Unconfirmed-Request-PDU

### 10.8.3.5 Unicast Messages That Should Not Be Routed

[Change **Clause 10.8.3.5.1**, p. 507]

#### 10.8.3.5.1 Unknown Network

Purpose: To verify that the IUT will not attempt to route a message directed to a device on an unknown network if the message was transmitted using a local broadcast MAC address.

Test Concept: Direct at one of the virtual devices a ReadProperty request that is correct in all aspects, except for the network number. Ensure that the virtual device does not reply. The request is sent as a local broadcast so that the IUT will receive it and not attempt to re-route it via another router to the unknown network.

*Notes to Tester: Choose a virtual device on Network 1 for this test.*

*Test Steps:*

1. TRANSMIT,  
DA = LOCAL BROADCAST,  
SA = TD,  
DNET = 59001,  
DADR = (the MAC address of the selected virtual device),  
Hop Count = 255,  
ReadProperty-Request,  
'Object Identifier' = (any object identifier of an object in the virtual device),  
'Property Identifier' = (any property of the specified object)
2. WAIT **Internal Processing Fail Time**
3. CHECK (verify that the IUT did not transmit I-Am-Router-To-Network (Network Numbers = 59001...)  
or Reject-Message-To-Network (Network Number = 59001)  
or any message in response to the Read Property request on Network 2)

[Add new **Clause 10.8.3.X1**, p. 508]

#### 10.8.3.X1 Silently Drop Messages to a Virtual Device that is Offline

Purpose: To verify that the IUT does not return any message in response to an NPDU with a destination that is offline.

Test Concept: The non-BACnet device is verified to be online and recognized by the IUT. It is then made to go offline, and the IUT is made to recognize that the device is offline. A property, P1, from Object1 which is derived from the data in a virtual device is read from the IUT. Verify that when a virtual device is off-line, that the IUT sends no response to messages that are directed to that off-line device.

Configuration Requirements: The IUT acting as a virtual router shall be configured so that a virtual device VD1A, which can sometimes be online, is initially online for this test. If no virtual device can become off-line, then this test shall be skipped.

*Test Steps:*

1. CHECK (any vendor-specified indication, that the virtual device is online)
2. MAKE (the virtual device containing Object1 go offline)
3. MAKE (the IUT notice that the virtual device is offline)
4. TRANSMIT ReadProperty-Request,  
DESTINATION = VD1A  
'Object Identifier' = Object1,  
'Property Identifier' = P1
5. CHECK (that no responsive message is returned from IUT)
6. TRANSMIT  
DESTINATION = VD1A,  
Message Type = (any valid value)
7. CHECK (that no responsive message is returned from IUT)

#### 10.8.4 Routing of Broadcast APDUs to Virtual Devices

[Change **Clause 10.8.4.7**, p. 516]

##### 10.8.4.7 Route Remote Broadcast Message from a Virtual Device to a Local Network

Purpose: To verify that the IUT can route a remote broadcast message from a virtual device to a local physical network.

Test Concept: Make one of the virtual devices generate a remote broadcast directed to the non-virtual network that the IUT is connected to, and verify that it is correctly formulated. This test shall be skipped if none of the IUT's virtual devices can issue a remote broadcast message.

*Configuration Requirements: The IUT shall be configured or otherwise stimulated so that one of its virtual devices will send a remote broadcast message to Network 2.*

*Notes to Tester: This test should be run repeatedly in order to exercise all ways that the IUT can be configured or stimulated to send a broadcast message to a local (physical) network. Depending on the capabilities of the IUT, this may involve sending a message from a device on the target network to the IUT (unicast or broadcast), writing a broadcast address to an object property in the IUT, or configuring the IUT using a proprietary method.*

Test Steps:

1. MAKE (the virtual device generate a remote broadcast message to the local network of the IUT)
2. RECEIVE,  
DA = LOCAL BROADCAST,  
SA = IUT,  
SNET = 1,  
SADR = (MAC address of a virtual device on Network 1),  
BACnet-Unconfirmed-Request-PDU

#### 10.8.7 Multiple Devices on a Single Virtual Network

[Add new **Clause 10.8.7.4**, p. 519]

##### 10.8.7.4 Who-Is Specifying Unknown Device Ids

Purpose: To verify that the IUT will not respond to discovery for devices that it does not contain.

Test Steps:

1. TRANSMIT,  
DA = IUT,  
SA = TD,  
DNET = 1,  
DLEN = 0,  
Hop Count = 255,  
BACnet-Unconfirmed-Request-PDU,  
Who-Is-Request,  
'Device Instance Range Low Limit' = (Low Limit of instance range excluding all virtual devices)  
'Device Instance Range High Limit' = (High Limit of instance range excluding all virtual devices)
2. CHECK (verify that the IUT does not transmit an I-Am-Request-PDU)

[Add new **Clause 10.8.7.5**, p. 519]

##### 10.8.7.5 Who-Has Specifying Unknown Device Ids

Purpose: To verify that the IUT will not respond to discovery for devices that it does not contain.

Test Steps:

1. TRANSMIT,  
DA = IUT,  
SA = TD,  
DNET = 1,  
DLEN = 0,  
Hop Count = 255,  
BACnet-Unconfirmed-Request-PDU,  
Who-Has-Request,  
'Device Instance Range Low Limit' = (Low Limit of instance range excluding all virtual devices)  
'Device Instance Range High Limit' = (High Limit of instance range excluding all virtual devices)  
'Object Identifier' = (Device object identifier of VD1B)
2. CHECK (verify that the IUT does not transmit an I-Have-Request-PDU)

## 135.1-2019s-2 Add new and correct existing tests for the MS/TP Data Link Layer

### Rationale

Configuration requirements are missing for an existent test. In addition, the test coverage shall be increased by new tests.

## 12.1 MS/TP State Machine Tests

### 12.1.3 MS/TP Data Link Layer Tests (Alternate)

[Change **Clause 12.1.3.3**, p. 581]

#### 12.1.3.3 Verify $T_{\text{frame\_gap}}$

Purpose: Verify that the maximum idle time between data octets when transmitting a frame is 20 bit times or less.

*Configuration Requirements: Run the IUT and a Reference Master (or Router) on the same MS/TP network.*

Test Steps:

1. Elicit the transmission of any frame from the IUT.
2. Measure the longest EIA-485 idle time that appears between octets within the data frame transmitted by the IUT. If there is no idle time between octets, pass the IUT.
3. Fail the IUT if the time measured in step 2 is greater than the time intervals shown below for each baud rate.

9600 baud:	fail if interval is greater than 2,083 microseconds
19200 baud:	fail if interval is greater than 1,042 microseconds
38400 baud:	fail if interval is greater than 521 microseconds
76800 baud:	fail if interval is greater than 261 microseconds
115200 baud:	fail if interval is greater than 173 microseconds
x baud:	fail if interval is greater than (20/x) seconds

[Add new **Clause 12.1.3.X1**, p. 587]

#### 12.1.3.X1 Frame Type Based on Transmitted NPDU size

Purpose: To verify that the IUT selects the correct frame type based on the transmitted NPDU size.

Test Concept: The IUT is made to send a frame such that the NPDU size is less than or equal to 501 octets (a non-COBS encoded frame) and the frame type is checked. The IUT is then made to send a frame such that the NPDU size is greater than 501 octets (a COBS encoded frame) and the frame type is checked.

It is expected that this test can be executed using ReadPropertyMultiple service requests to generate responses from the IUT of different sizes. If the IUT does not support execution of ReadPropertyMultiple service requests and the IUT cannot be made to send a frame larger than 501 octets by any other means, this test shall be skipped.

Test Steps:

1. MAKE (the IUT generate a frame with an NPDU less than 501 octets)
2. CHECK (Frame type = BACnet Data Expecting Reply(5) or BACnet Data Not Expecting Reply(6))
3. MAKE (the IUT generate a frame with an NPDU greater than 501 octets)
4. CHECK (Frame type = BACnet Extended Data Expecting Reply(32) or BACnet Extended Data Not Expecting Reply(33))

[Add new **Clause 12.1.3.X2**, p. 587]



### 12.1.3.X2 Executing COBS Encoded Frames

Purpose: To verify that the IUT can properly execute COBS encoded frames

Test Concept: A COBS encoded service request is sent to the IUT and proper execution of the request is verified.

It is expected that this test can be executed for server devices using a large ReadPropertyMultiple service request that the server can execute. If the IUT does not support execution of ReadPropertyMultiple service requests, the vendor must provide instructions and means for verifying correct execution of a request.

Test Steps:

1. IF (the IUT supports execution of ReadPropertyMultiple service requests) THEN  
    READ V = (Object1), P1  
    TRANSMIT (a ReadPropertyMultiple request with an NPDU larger than 501 octets including (Object1), P1)  
    VERIFY (Object1), P1 = V
2. ELSE  
    (Use vendor supplied instructions to verify execution of a service request)

[Add new **Clause 12.1.3.X3**, p. 587]

### 12.1.3.X3 Data Not For Us Test

Purpose: Verify that the IUT properly skips the complete data portion of frames not intended for the IUT.

Test Concept: Send a BACnet Data Not Expecting Reply frame that contains the frame pre-amble octet sequence to an address other than the IUT. Follow it immediately with a ReadProperty request for the IUT's device object to ensure that the IUT will correctly receive and process the ReadProperty request.

Test Steps:

1. TRANSMIT  
    Frame Type = BACnet Data Not Expecting Reply  
    Destination Address = (any Unicast address other than IUT),  
    Length = 7,  
    Data = (55 FF 05 FF 00 01 F5)
2. TRANSMIT ReadProperty-Request  
    'Object Identifier' = (device, 4194303),  
    'Property Identifier' = Object\_Name
3. RECEIVE ReadProperty-Response  
    'Object Identifier' = (device, IUT),  
    'Property Identifier' = Object\_Name,  
    'Value' = (any valid value)

### 135.1-2019s-3 Add tests for the BACnet Secure Connect Data Link Layer

#### Rationale

Addendum 135-2016*bj* introduced the new BACnet Secure Connect Data Link Layer option. Addendum 135.1-2019s adds new tests to meet the requirements of addendum *bj*.

[Add new **Clause 12.YY**, p. 625]

#### 12.YY Secure Connect Functionality Tests

This clause defines the tests necessary to demonstrate Secure Connect functionality, as defined in Annex YY of the BACnet standard.

In the diagrams that follow, the following legend applies. Nodes and hub functions are shown within the BACnet device in which they reside by having the circle or rounded square located inside a BACnet device rectangle.



Secure Connect differs from other datalinks in that a single network consists of numerous logical connections instead of a shared bus. While the messages do usually exist on a shared ethernet segment, they are described as if each WebSocket is a separate link.

Where it is not clear which WebSocket the messages are expected on, the PORT keyword is used to identify the WebSocket. This construct is mostly used when the IUT has multiple connections such as when it is a hub or participating in direct connections.

#### 12.YY.1 Basic Node Tests

This group of tests verifies secure connect devices operating in a non-hub mode. The logical configuration of the network used for these tests is shown in Figure X1. The test descriptions in this clause assume that the TD plays the role of all of the other devices in the network configuration. For the tests in this section, unless specified through a PORT parameter, messages specified by the test are on the IUT to TD hub WebSocket connection (primary or failover).

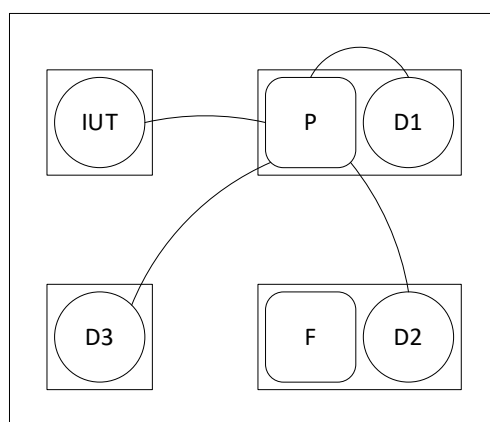


Figure X1: Network setup for basic node tests showing connections when the primary hub is active.

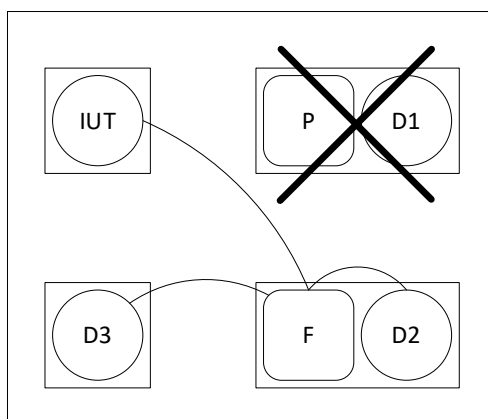


Figure X2: Network setup for basic node tests showing connections when the primary hub is inactive.

## 12.YY.1.1 Basic Node Positive Tests

### 12.YY.1.1.1 Connect and Maintain Hub Connection Test

Purpose: To verify that the IUT connects to the configured primary hub and maintains the connection over time.

Test Concept: With the IUT configured to connect to the TD as the primary hub, allow the IUT to connect. Wait an arbitrary amount of time and have the hub silently close the WebSocket. Verify that the IUT detects the closure and attempts to reconnect to the hub. Allow the hub to accept the new connection. Wait an arbitrary amount of time and have the hub request a disconnect. Verify that the IUT acknowledges the disconnect. Verify that the IUT attempts to reconnect to the hub. Allow the hub to accept the new connection.

Configuration Requirements: The TD is configured as the primary hub, and the IUT's primary hub URI is configured to reference it.

Test Steps:

1. MAKE(the IUT connect to the primary hub)
2. WAIT a tester selected amount of time
3. MAKE(the hub close the WebSocket)
4. CHECK(that the IUT opens a new WebSocket with the primary hub)
5. RECEIVE Connect-Request,
  - 'Message ID' = (M1: any valid value),
  - 'Originating Virtual Address' absent
  - 'Destination Virtual Address' absent
  - 'Destination Options' (absent or any valid value),
  - 'Data Options' absent
  - 'VMAC Address' = (IUT's VMAC),
  - 'Device UUID' = (IUT's UUID),
  - 'Maximum BVLC Length' = (the IUT's maximum BVLC accepted length),
  - 'Maximum NPDU Length' = (the IUT's maximum NPDU accepted length)
6. TRANSMIT Connect-Accept,
  - 'Message ID' = M1,
  - 'Originating Virtual Address' absent
  - 'Destination Virtual Address' absent
  - 'Destination Options' (absent or any valid value),
  - 'Data Options' absent
  - 'VMAC Address' = (TD's VMAC),
  - 'Device UUID' = (TD's UUID),
  - 'Maximum BVLC Length' = (the TD's maximum BVLC accepted length),
  - 'Maximum NPDU Length' = (the TD's maximum NPDU accepted length)
7. WAIT a tester selected amount of time

8. TRANSMIT Disconnect-Request,
  - 'Message ID' = (M2: any valid value),
  - 'Originating Virtual Address' absent
  - 'Destination Virtual Address' absent
  - 'Destination Options' (absent or any valid value),
  - 'Data Options' absent
9. RECEIVE Disconnect-ACK,
  - 'Message ID' = M2,
  - 'Originating Virtual Address' absent
  - 'Destination Virtual Address' absent
  - 'Destination Options' (absent or any valid value),
  - 'Data Options' absent
10. MAKE(the TD close the WebSocket)
11. CHECK(that the IUT opens a new WebSocket with the TD)
12. RECEIVE Connect-Request,
  - 'Message ID' = (M3: any valid value),
  - 'Originating Virtual Address' absent
  - 'Destination Virtual Address' absent
  - 'Destination Options' (absent or any valid value),
  - 'Data Options' absent
  - 'VMAC Address' = (IUT's VMAC),
  - 'Device UUID' = (IUT's UUID),
  - 'Maximum BVLC Length' = (the IUT's maximum BVLC accepted length),
  - 'Maximum NPDU Length' = (the IUT's maximum NPDU accepted length)
13. TRANSMIT Connect-Accept,
  - 'Message ID' = M3,
  - 'Originating Virtual Address' absent
  - 'Destination Virtual Address' absent
  - 'Destination Options' (absent or any valid value),
  - 'Data Options' absent
  - 'VMAC Address' = (TD's VMAC),
  - 'Device UUID' = (TD's UUID),
  - 'Maximum BVLC Length' = (the TD's maximum BVLC accepted length),
  - 'Maximum NPDU Length' = (the TD's maximum NPDU accepted length)

## 12.YY.1.1.2 Connect to Failover Hub Test

Purpose: To verify that the IUT connects to the configured failover hub and maintains the connection when the connection to the primary is lost.

Test Concept: With the IUT configured with a primary and a failover hub, allow the IUT to connect to the primary. Wait an arbitrary amount of time and have the hub silently close the WebSocket and stop responding on that port. Verify that the IUT detects the closure and attempts to reconnect to the primary hub. Verify that after the reconnect attempt fails, the IUT attempts to connect to the failover hub. Allow the connection to succeed.

Configuration Requirements: The IUT configured to connect to the TD as the primary hub, and as the failover hub.

Test Steps:

1. MAKE(the IUT connect to the primary hub)
2. WAIT a tester selected amount of time
3. MAKE(the hub close the WebSocket)
4. CHECK(that the IUT opens a new WebSocket with the primary hub)
5. RECEIVE PORT (IUT-TD primary hub WebSocket)
  - Connect-Request,
    - 'Message ID' = (M1: any valid value),
    - 'Originating Virtual Address' absent
    - 'Destination Virtual Address' absent
    - 'Destination Options' (absent or any valid value),
    - 'Data Options' absent

- 'VMAC Address' = (IUT's VMAC),
- 'Device UUID' = (IUT's UUID),
- 'Maximum BVLC Length' = (the IUT's maximum BVLC accepted length),
- 'Maximum NPDU Length' = (the IUT's maximum NPDU accepted length)
- 6. TRANSMIT PORT (IUT-TD primary hub WebSocket)
  - Connect-Accept,
  - 'Message ID' = M1,
  - 'Originating Virtual Address' absent
  - 'Destination Virtual Address' absent
  - 'Destination Options' (absent or any valid value),
  - 'Data Options' absent
  - 'VMAC Address' = (TD's VMAC),
  - 'Device UUID' = (TD's UUID),
  - 'Maximum BVLC Length' = (the TD's maximum BVLC accepted length),
  - 'Maximum NPDU Length' = (the TD's maximum NPDU accepted length)
- 7. WAIT a tester selected amount of time
- 8. MAKE(the TD, as primary hub, close the WebSocket connection to the IUT and refuse future connections)
- 9. CHECK(that the IUT attempts to open a new WebSocket with the the primary hub)
- 10. CHECK(that the IUT opens a WebSocket with the failover hub)
- 11. RECEIVE PORT (IUT-TD failover hub WebSocket)
  - Connect-Request,
  - 'DA' = D2,
  - 'Message ID' = (M2: any valid value),
  - 'Originating Virtual Address' absent
  - 'Destination Virtual Address' absent
  - 'Destination Options' (absent or any valid value),
  - 'Data Options' absent
  - 'VMAC Address' = (IUT's VMAC),
  - 'Device UUID' = (IUT's UUID),
  - 'Maximum BVLC Length' = (the IUT's maximum BVLC accepted length),
  - 'Maximum NPDU Length' = (the IUT's maximum NPDU accepted length)
- 12. TRANSMIT PORT (IUT-TD failover hub WebSocket)
  - Connect-Accept,
  - 'SA' = D2,
  - 'Message ID' = M2,
  - 'Originating Virtual Address' absent
  - 'Destination Virtual Address' absent
  - 'Destination Options' (absent or any valid value),
  - 'Data Options' absent
  - 'VMAC Address' = (D2 VMAC),
  - 'Device UUID' = (D2's UUID),
  - 'Maximum BVLC Length' = (the D2's maximum BVLC accepted length),
  - 'Maximum NPDU Length' = (the D2's maximum NPDU accepted length)

### 12.YY.1.1.3 Connect to Failover Hub on Startup Test

Purpose: To verify that the IUT connects to the configured failover hub when it powers on and the primary is offline.

Test Concept: With the IUT configured with a primary and a failover hub, the primary hub offline, and the IUT powered off, power on the IUT. Verify that the IUT attempts to connect to the primary. Verify that the IUT attempts to connect to the failover hub when the attempt to the primary fails.

Configuration Requirements: The IUT configured to connect to the TD as the primary hub, and as the failover hub. The primary hub is offline. The IUT starts the test powered off.

Test Steps:

1. MAKE(power on the IUT)
2. CHECK(that the IUT attempts open a WebSocket to the primary hub)
3. CHECK(that the IUT opens a new WebSocket with the failover hub)
4. RECEIVE PORT (IUT-TD failover hub WebSocket)
  - Connect-Request,

'Message ID' = (M1: any valid value),  
 -- 'Originating Virtual Address' absent  
 -- 'Destination Virtual Address' absent  
 'Destination Options' (absent or any valid value),  
 -- 'Data Options' absent  
 'VMAC Address' = (IUT's VMAC),  
 'Device UUID' = (IUT's UUID),  
 'Maximum BVLC Length' = (the IUT's maximum BVLC accepted length),  
 'Maximum NPDU Length' = (the IUT's maximum NPDU accepted length)

5. TRANSMIT PORT (IUT-TD failover hub WebSocket)

Connect-Accept,  
 SA = D2,  
 'Message ID' = M1,  
 -- 'Originating Virtual Address' absent  
 -- 'Destination Virtual Address' absent  
 'Destination Options' (absent or any valid value),  
 -- 'Data Options' absent  
 'VMAC Address' = (D2's VMAC),  
 'Device UUID' = (D2's UUID),  
 'Maximum BVLC Length' = (the D2's maximum BVLC accepted length),  
 'Maximum NPDU Length' = (the D2's maximum NPDU accepted length)

#### 12.YY.1.1.4 Reconnect to Primary Hub Test

Purpose: To verify that the IUT reconnects to the configured primary hub within 600 seconds of when it comes back online.

Test Concept: The test starts with the IUT connected to the failover hub and the primary hub offline. Wait an arbitrary amount of time and have the primary hub come back online. Verify that the IUT attempts to reconnect to the primary within 600 seconds of it coming back online.

Configuration Requirements: The IUT configured to connect to the TD as the primary hub, and as the failover hub. The test starts with the primary hub offline and the IUT connected to the failover hub.

Test Steps:

1. WAIT a tester selected amount of time
2. MAKE(bring the primary hub online)
4. CHECK(that the IUT opens a new WebSocket with the primary hub within 600 seconds)
5. RECEIVE PORT (IUT-TD primary hub WebSocket)

Connect-Request,  
 'Message ID' = (M1: any valid value),  
 -- 'Originating Virtual Address' absent  
 -- 'Destination Virtual Address' absent  
 'Destination Options' (absent or any valid value),  
 -- 'Data Options' absent  
 'VMAC Address' = (IUT's VMAC),  
 'Device UUID' = (IUT's UUID),  
 'Maximum BVLC Length' = (the IUT's maximum BVLC accepted length),  
 'Maximum NPDU Length' = (the IUT's maximum NPDU accepted length)

6. TRANSMIT PORT (IUT-TD primary hub WebSocket)

Connect-Accept,  
 'Message ID' = M1,  
 -- 'Originating Virtual Address' absent  
 -- 'Destination Virtual Address' absent  
 'Destination Options' (absent or any valid value),  
 -- 'Data Options' absent  
 'VMAC Address' = (TD's VMAC),  
 'Device UUID' = (TD's UUID),  
 'Maximum BVLC Length' = (the TD's maximum BVLC accepted length),  
 'Maximum NPDU Length' = (the TD's maximum NPDU accepted length)

### 12.YY.1.1.5 Unicast Through Hub Test

Purpose: Verify that the IUT correctly composes unicasts aimed at a device through the hub.

Test Concept: With the IUT connected to the BACnet/SC network, send a ReadProperty from OD to the IUT. Verify that the IUT accepts the request and responds with a ReadProperty-ACK to OD, the ack properly composed and sent to the hub.

Configuration Requirements: The TD is configured as the primary hub and the IUT is connected to it. OD is any device connected to the BACnet/SC network through the primary hub (but not the hub's node).

Test Steps:

1. TRANSMIT Encapsulated-NPDU,
  - 'Message ID' = (M1: any valid value),
  - 'Originating Virtual Address' = (OD's VMAC),
  - 'Destination Options' = (absent or a valid list of options),
  - 'Data Options' = ({X'41'}), -- Secure Path
  - 'BACnet NPDU' =
    - ReadProperty-Request
    - 'Object Identifier' = (O: any object in the IUT),
    - 'Property Identifier' = Object\_Name
2. RECEIVE Encapsulated-NPDU,
  - 'Message ID' = (M2: any valid value),
  - 'Originating Virtual Address' absent
  - 'Destination Virtual Address' = (OD's VMAC),
  - 'Destination Options' = (absent or a valid list of options),
  - 'Data Options' = ({X'41' or a list of data options including Secure Path}),
  - 'BACnet NPDU' =
    - ReadProperty-ACK
    - 'Object Identifier' = (O),
    - 'Property Identifier' = Object\_Name,
    - 'Property Value' = (the value from the EPICS)

### 12.YY.1.1.6 Unicast to Hub Test

Purpose: Verify that the IUT correctly composes unicasts aimed at the hub.

Test Concept: With the IUT connected to the BACnet/SC network, send a ReadProperty from the node on the same device as the primary hub to the IUT. Verify that the IUT accepts the request and responds with a ReadProperty-ACK, the ack properly composed and sent to the hub.

Configuration Requirements: The TD is configured as the primary hub.

Test Steps:

1. TRANSMIT PORT (IUT-TD primary hub WebSocket)
  - Encapsulated-NPDU,
  - 'Message ID' = (M1: any valid value),
  - 'Originating Virtual Address' = TD's VMAC,
  - 'Destination Virtual Address' absent
  - 'Destination Options' = (absent or any valid value),
  - 'Data Options' = ({ X'41'}), -- Secure Path
  - 'BACnet NPDU' =
    - ReadProperty-Request,
    - 'Object Identifier' = (the IUT's Device object),
    - 'Property Identifier' = Object\_Name
2. RECEIVE PORT (IUT-TD primary hub WebSocket)
  - Encapsulated-NPDU,
  - 'Message ID' = M2,
  - 'Originating Virtual Address' absent
  - 'Destination Virtual Address' = TD's VMAC,
  - 'Destination Options' = (absent or any valid value),
  - 'Data Options' = ({ X'41' or a list of valid header options including Secure Path}),
  - 'BACnet NPDU' =

ReadProperty-ACK,  
 'Object Identifier' = (the IUT's Device object),  
 'Property Identifier' = Object\_Name,  
 'Property Value' = (the IUT's device object name)

### 12.YY.1.1.7 Local Broadcast Initiation Test

Purpose: To verify that broadcasts are sent to the hub and are sent with the Local broadcast VMAC address

Test Concept: Make the IUT generate a broadcast. Verify that the broadcast is correctly formed and forwarded to the primary hub.

Configuration Requirements: The TD is configured as the primary hub and the IUT is connected to it.

Test Steps:

1. MAKE(the IUT generate a broadcast)
2. RECEIVE Encapsulated-NPDU,
  - 'Message ID' = (M1: any valid value)
  - 'Originating Virtual Address' absent
  - 'Destination Virtual Address' = (Local Broadcast VMAC, X'FFFFFFFFFFFF')
  - 'Destination Options' = (absent or a valid list of options),
  - 'Data Options' = ({X'41' or a list of data options including Secure Path}),
  - 'BACnet NPDU' = (any valid broadcastable NPDU)

### 12.YY.1.1.8 Local Broadcast Execution Test

Purpose: To verify that IUT correctly accepts and processes broadcast messages.

Test Concept: Send a broadcast Who-Is with no device range. Verify that the IUT sends an I-Am in response through the primary hub.

Configuration Requirements: The TD is configured as the primary and the IUT is connected to it.

Test Steps:

1. TRANSMIT Encapsulated-NPDU,
  - 'Message ID' = (M1: any valid value),
  - 'Originating Virtual Address' = (OD's VMAC),
  - 'Destination Virtual Address' = (Local Broadcast VMAC, X'FFFFFFFFFFFF')
  - 'Destination Options' = (absent or a valid list of options),
  - 'Data Options' = ({X'41'}), -- Secure Path
  - 'BACnet NPDU' =
    - Who-Is-Request,
    - 'Device Instance Range Low Limit' = (IUT's device instance),
    - 'Device Instance Range High Limit' = (IUT's device instance)
2. RECEIVE Encapsulated-NPDU,
  - 'Message ID' = (M2: any valid value),
  - 'Originating Virtual Address' absent
  - 'Destination Virtual Address' = (Local Broadcast VMAC, X'FFFFFFFFFFFF'),
  - 'Destination Options' = (absent or a valid list of options),
  - 'Data Options' = ({X'41' or a list of data options including Secure Path}),
  - 'BACnet NPDU' =
    - I-Am-Request,
    - 'I Am Device Identifier' = (the IUT's Device object),
    - 'Max APDU Length Accepted' = (the value specified in the EPICS),
    - 'Segmentation Supported' = (the value specified in the EPICS),
    - 'Vendor Identifier' = (the identifier registered for this vendor)



### 12.YY.1.1.9 VMAC Uniqueness Test

Purpose: To verify that the IUT will attempt to resolve its own VMAC and will generate a new Random-48 VMAC if a conflict is detected.

Test Concept: Have the IUT connect to the primary hub. During the connect sequence, the hub returns an NAK with an error code of NODE\_DUPLICATE\_VMAC. Verify that the IUT retries the connection with a Random-48 VMAC which differs from the initial connect attempt. Verify that the selected VMAC is valid. Again, the hub NAKs the connection request with an error code of NODE\_DUPLICATE\_VMAC. Verify that the IUT retries again with a valid Random-48 VMAC not equal to either of the previous VMACs.

Test Steps:

1. MAKE(the IUT initiate a hub connection TD)
2. CHECK(that the IUT attempts to open a new WebSocket with the TD)
3. RECEIVE Connect-Request,
  - 'Message ID' = (M1: any valid value),
  - 'Originating Virtual Address' absent
  - 'Destination Virtual Address' absent
  - 'Destination Options' (absent or any valid value),
  - 'Data Options' absent
  - 'VMAC Address' = (VMAC1: IUT's VMAC),
  - 'Device UUID' = (IUT's UUID),
  - 'Maximum BVLC Length' = (the IUT's maximum BVLC accepted length),
  - 'Maximum NPDU Length' = (the IUT's maximum NPDU accepted length)
4. TRANSMIT BVLC-Result,
  - 'Message ID' = M1,
  - 'Originating Virtual Address' absent
  - 'Destination Virtual Address' absent
  - 'Destination Options' (absent or any valid value),
  - 'Data Options' absent
  - 'Result for BVLC Function' = X'06', -- Connect-Request
  - 'Result Code' = X'01', -- NAK
  - 'Error Header Marker' = X'00', -- not a header option problem
  - 'Error Class' = COMMUNICATION,
  - 'Error Code' = NODE\_DUPLICATE\_VMAC
5. RECEIVE Connect-Request,
  - 'Message ID' = (M2: any valid value),
  - 'Originating Virtual Address' absent
  - 'Destination Virtual Address' absent
  - 'Destination Options' (absent or any valid value),
  - 'Data Options' absent
  - 'VMAC Address' = (VMAC2: IUT's new VMAC, different than VMAC1),
  - 'Device UUID' = (IUT's UUID),
  - 'Maximum BVLC Length' = (the IUT's maximum BVLC accepted length),
  - 'Maximum NPDU Length' = (the IUT's maximum NPDU accepted length)
6. TRANSMIT BVLC-Result,
  - 'Message ID' = M2,
  - 'Originating Virtual Address' absent
  - 'Destination Virtual Address' absent
  - 'Destination Options' (absent or any valid value),
  - 'Data Options' absent
  - 'Result for BVLC Function' = X'06', -- Connect-Request
  - 'Result Code' = X'01', -- NAK
  - 'Error Header Marker' = X'00', -- not a header option problem
  - 'Error Class' = COMMUNICATION,
  - 'Error Code' = NODE\_DUPLICATE\_VMAC
7. RECEIVE Connect-Request,
  - 'Message ID' = (M3: any valid value),
  - 'Originating Virtual Address' absent
  - 'Destination Virtual Address' absent
  - 'Destination Options' (absent or any valid value),

- 'Data Options' absent
- 'VMAC Address' = (VMAC3: IUT's new VMAC, different than VMAC1 & VMAC2),
- 'Device UUID' = (IUT's UUID),
- 'Maximum BVLC Length' = (the IUT's maximum BVLC accepted length),
- 'Maximum NPDU Length' = (the IUT's maximum NPDU accepted length)
- 8. TRANSMIT Connect-Accept,
  - 'Message ID' = M3,
  - 'Originating Virtual Address' absent
  - 'Destination Virtual Address' absent
  - 'Destination Options' = (absent or any valid value),
  - 'Data Options' absent
  - 'VMAC Address' = (TD's VMAC),
  - 'Device UUID' = (TD's UUID),
  - 'Maximum BVLC Length' = (the TD's maximum BVLC accepted length),
  - 'Maximum NPDU Length' = (the TD's maximum NPDU accepted length)

### 12.YY.1.1.10 UUID Persistence Test

Purpose: To verify that the IUT contains a UUID and that it persists across resets

Test Concept: Configure the IUT and have it connect to the primary hub. Power off the IUT, wait 1 minute, and power on the IUT allowing it to connect again to the hub. Verify that the UUID in both connect messages is the same.

Test Steps:

1. MAKE(IUT connect to the primary hub)
2. RECEIVE Connect-Request,
  - 'Message ID' = (M1: any valid value),
  - 'Originating Virtual Address' absent
  - 'Destination Virtual Address' absent
  - 'Destination Options' = (absent or a valid list of options),
  - 'Data Options' absent
  - 'VMAC Address' = (IUT's VMAC),
  - 'Device UUID' = (U: any valid UUID),
  - 'Maximum BVLC Length' = (the IUT's maximum BVLC accepted length),
  - 'Maximum NPDU Length' = (the IUT's maximum NPDU accepted length)
3. TRANSMIT Connect-Accept,
  - 'Message ID' = M1,
  - 'Originating Virtual Address' absent
  - 'Destination Virtual Address' absent
  - 'Destination Options' = (absent or a valid list of options),
  - 'Data Options' absent
  - 'VMAC Address' = (TD's VMAC),
  - 'Device UUID' = (TD's UUID),
  - 'Maximum BVLC Length' = (the TD's maximum BVLC accepted length),
  - 'Maximum NPDU Length' = (the TD's maximum NPDU accepted length)
4. MAKE(power off the IUT)
5. WAIT 1 minute
6. MAKE(power on the IUT)
7. BEFORE the maximum time it takes for the IUT to reboot and re-establish a connection to the network
  - RECEIVE Connect-Request,
    - 'Message ID' = (M2: any valid value),
    - 'Originating Virtual Address' absent
    - 'Destination Virtual Address' absent
    - 'Destination Options' = (absent or a valid list of options),
    - 'Data Options' absent
    - 'VMAC Address' = (IUT's VMAC),
    - 'Device UUID' = U,
    - 'Maximum BVLC Length' = (the IUT's maximum BVLC accepted length),
    - 'Maximum NPDU Length' = (the IUT's maximum NPDU accepted length)
8. TRANSMIT Connect-Accept,

'Message ID' = M2,  
 -- 'Originating Virtual Address' absent  
 -- 'Destination Virtual Address' absent  
 'Destination Options' = (absent or a valid list of options),  
 -- 'Data Options' absent  
 'VMAC Address' = (TD's VMAC),  
 'Device UUID' = (TD's UUID),  
 'Maximum BVLC Length' = (the TD's maximum BVLC accepted length),  
 'Maximum NPDU Length' = (the TD's maximum NPDU accepted length)

### 12.YY.1.1.11 UUID Persistence When VMAC Changes Test

Purpose: To verify that the IUT's UUID is unrelated to its VMAC.

Test Concept: Configure the IUT and connect it to the network. Let it verify its VMAC successfully. Reset the IUT and when it connects to the network, indicate a VMAC conflict. After the IUT allocates a new VMAC, and send a new advertisement, verify that the UUID has not changed.

Test Steps:

-- allow the IUT to connect and use its chosen VMAC

1. MAKE(the IUT connect to the primary hub)

2. RECEIVE Connect-Request,

'Message ID' = (M1: any valid value),  
 -- 'Originating Virtual Address' absent  
 -- 'Destination Virtual Address' absent  
 'Destination Options' = (absent or a valid list of options),  
 -- 'Data Options' absent  
 'VMAC Address' = (V1: any valid value),  
 'Device UUID' = U,  
 'Maximum BVLC Length' = (the IUT's maximum BVLC accepted length),  
 'Maximum NPDU Length' = (the IUT's maximum NPDU accepted length)

3. TRANSMIT Connect-Accept,

'Message ID' = M1,  
 -- 'Originating Virtual Address' absent  
 -- 'Destination Virtual Address' absent  
 'Destination Options' = (absent or a valid list of options),  
 -- 'Data Options' absent  
 'VMAC Address' = (TD's VMAC),  
 'Device UUID' = (TD's UUID),  
 'Maximum BVLC Length' = (the TD's maximum BVLC accepted length),  
 'Maximum NPDU Length' = (the TD's maximum NPDU accepted length)

-- TD pretending a new device has joined using VMAC V1 with UUID U2

-- the TD will now reject the IUT's VMAC as if another device has connected with the IUT's VMAC

4. MAKE(the IUT reset)

5. BEFORE the maximum time it takes for the IUT to reboot and re-establish a connection to the network

RECEIVE Connect-Request,

'Message ID' = (M2: any valid value),  
 -- 'Originating Virtual Address' absent  
 -- 'Destination Virtual Address' absent  
 'Destination Options' = (absent or a valid list of options),  
 -- 'Data Options' absent  
 'VMAC Address' = V1,  
 'Device UUID' = U,  
 'Maximum BVLC Length' = (the IUT's maximum BVLC accepted length),  
 'Maximum NPDU Length' = (the IUT's maximum NPDU accepted length)

6. TRANSMIT BVLC-Result,

'Message ID' = M2,  
 -- 'Originating Virtual Address' absent  
 -- 'Destination Virtual Address' absent

'Destination Options' = (absent or a valid list of options),  
 -- 'Data Options' absent  
 'Result for BVLC Function' = X'06', -- Connect-Request  
 'Result Code' = X'01', -- NAK  
 'Error Header Marker' = X'00', -- not related to a header option  
 'Error Class' = COMMUNICATION,  
 'Error Code' = NODE\_DUPLICATE\_VMAC

-- verify that the IUT retries with a new VMAC and the same UUID

7. RECEIVE Connect-Request,
  - 'Message ID' = (M3: any valid value),
  - 'Originating Virtual Address' absent
  - 'Destination Virtual Address' absent
  - 'Destination Options' = (absent or a valid list of options),
  - 'Data Options' absent
  - 'VMAC Address' = (V2: any valid value different than V1),
  - 'Device UUID' = U,
  - 'Maximum BVLC Length' = (the IUT's maximum BVLC accepted length),
  - 'Maximum NPDU Length' = (the IUT's maximum NPDU accepted length)
8. TRANSMIT Connect-Accept,
  - 'Message ID' = M3,
  - 'Originating Virtual Address' absent
  - 'Destination Virtual Address' absent
  - 'Destination Options' = (absent or a valid list of options),
  - 'Data Options' absent
  - 'VMAC Address' = (TD's VMAC),
  - 'Device UUID' = (TD's UUID),
  - 'Maximum BVLC Length' = (the TD's maximum BVLC accepted length),
  - 'Maximum NPDU Length' = (the TD's maximum NPDU accepted length)

### 12.YY.1.1.12 Unknown 'Must Understand' is True Message Test

Purpose: To verify that a unicast message is not ignored if the 'Must Understand' flag is set and the option is unknown or not supported.

Test Concept: With the IUT connected to the BACnet/SC network, send a ReadProperty request with a set of header options with one that the IUT does not know marked as 'Must Understand'. Verify that the IUT responds with a NAK and Error Class of COMMUNICATION and an error code of HEADER\_NOT\_UNDERSTOOD correctly identifying the option marked as 'Must Understand'. Verify that the IUT does not send a ReadProperty-ACK. Repeat with a globally broadcast Who-Is request.

Test Steps:

1. TRANSMIT Encapsulated-NPDU,
  - 'Message ID' = (M1: any valid value),
  - 'Originating Virtual Address' = (OVA: any valid value, including absent),
  - 'Destination Virtual Address' absent
  - 'Destination Options' = ({X'BF0003000012', --proprietary,more,not M.U.,len=3,ven=0, opt=x12  
 X'FF0003000000', --proprietary, more,M.U.,len=3,ven=0, opt=x00  
 X'3F0003000034'}), --proprietary,not more,not M.U.,len=3,ven=0,opt=x34
  - 'Data Options' = ({ X'41' }), -- Secure Path
  - 'BACnet NPDU' =
    - ReadProperty-Request,
    - 'Object Identifier' = (the IUT's Device object),
    - 'Property Identifier' = Object\_Name
2. RECEIVE BVLC-Result,
  - 'Message ID' = M1,
  - 'Originating Virtual Address' absent
  - 'Destination Virtual Address' = (OVA),
  - 'Destination Options' = (absent or a valid list of options),
  - 'Data Options' absent
  - 'Result for BVLC Function' = X'01', -- Encapsulated-NPDU

- 'Result Code' = X'01', -- NAK  
 'Error Header Marker' = X'FF', -- second header option  
 'Error Class' = COMMUNICATION,  
 'Error Code' = HEADER\_NOT\_UNDERSTOOD
3. CHECK(that the IUT does not send a ReadProperty-ACK)
  4. TRANSMIT Encapsulated-NPDU,  
 'Message ID' = (M2: any valid value),  
 'Originating Virtual Address' = (OVA: any valid value, including absent),  
 'Destination Virtual Address' = (X'FFFFFFF', the local broadcast VMAC),  
 'Destination Options' = ({X'BF0003000012',--proprietary,more,not M.U.,len=3,ven=0, opt=x12  
 X'FF0003000000', --proprietary, more,M.U.,len=3,ven=0, opt=x00  
 X'3F0003000034'}),--proprietary,not more,not M.U.,len=3,ven=0,opt=x34  
 ({ X'41'}), -- Secure Path  
 'Data Options' =  
 'BACnet NPDU' =  
 Who-Is-Request
  5. CHECK(that the IUT does not route the message, send a BVLC-Result, nor send an IAm-Request)

### 12.YY.1.1.13 Unknown 'Must Understand' is False Message Test

Purpose: To verify that a message is correctly processed when an unknown destination option is present whose 'Must Understand' flag is cleared.

Test Concept: With the IUT connected to the BACnet/SC network, send a ReadProperty request with a header option that the IUT does not know which is not marked as 'Must Understand'. Verify that the IUT accepts the request and responds with a ReadProperty-ACK.

Test Steps:

-- Unicast test

1. TRANSMIT Encapsulated-NPDU,  
 'Message ID' = (M1: any valid value),  
 'Originating Virtual Address' = (OVA: any valid value, including absent),  
 'Destination Options' = ({X'3F0030000034'}), --proprietary,not more,not M.U.,len=0,ven=0,opt=x34  
 'Data Options' = ({ X'41'}) -- Secure Path  
 'BACnet NPDU' =  
 ReadProperty-Request,  
 'Object Identifier' = (the IUT's Device object),  
 'Property Identifier' = Object\_Name
2. RECEIVE Encapsulated-NPDU,  
 'Message ID' = (M2: any valid value),  
 -- 'Originating Virtual Address' absent  
 'Destination Virtual Address' = OVA,  
 'Destination Options' = (absent or a valid list of header options),  
 'Data Options' = ({ X'41' or a list of valid header options including Secure Path}),  
 'BACnet NPDU' =  
 ReadProperty-ACK,  
 'Object Identifier' = (the IUT's Device object),  
 'Property Identifier' = Object\_Name,  
 'Property Value' = (the IUT's device object name)

-- Broadcast test

3. TRANSMIT Encapsulated-NPDU,  
 'Message ID' = (M3: any valid value),  
 'Originating Virtual Address' = (OVA: any valid value, including absent),  
 'Destination Virtual Address' = (X'FFFFFFF' the local broadcast VMAC),  
 'Destination Options' = ({X'3F0030000034'}),--proprietary,not more,not M.U., len=0, ven=0,  
 -- opt=x34  
 'Data Options' = ({ X'41'}), -- Secure Path  
 'BACnet NPDU' =  
 Who-Is-Request
4. RECEIVE Encapsulated-NPDU,

'Message ID' = (M4: any valid value),  
 -- 'Originating Virtual Address' absent  
 'Destination Virtual Address' = (OVA, or the local broadcast VMAC),  
 'Destination Options' = (absent or a valid list of header options),  
 'Data Options' = ({ X'41' or a list of valid header options including Secure Path } ),  
 'BACnet NPDU' =  
   I-Am-Request,  
   'I Am Device Identifier' = (the IUT's Device object),  
   'Max APDU Length Accepted' = (the value specified in the EPICS),  
   'Segmentation Supported' = (the value specified in the EPICS),  
   'Vendor Identifier' = (the identifier registered for this vendor)

#### 12.YY.1.1.14 Multiple Header Options Test

Purpose: To verify that the IUT accepts and processes messages with varying numbers of header options.

Test Concept: With the IUT connected to the BACnet/SC network, send a ReadProperty request with more than 1 header option, with none marked as 'Must Understand'. Verify that the IUT responds with a ReadProperty-ACK.

Test Steps:

1. TRANSMIT Encapsulated-NPDU,  
   'Message ID' = (M1: any valid value),  
   'Originating Virtual Address' = (OVA: any valid value, including absent),  
   -- 'Destination Virtual Address' absent  
   'Destination Options' (absent or any valid value),  
   'Data Options' = ({ X'C1', -- Secure Path, more options  
                       X'BF0003000012', --proprietary,more,not M.U.,len=3,ven=0, opt=12  
                       X'3F0003000034'}), --proprietary,not more,not M.U.,len=3,ven=0, opt=x34  
   'BACnet NPDU' =  
     ReadProperty-Request,  
     'Object Identifier' = (the IUT's Device object),  
     'Property Identifier' = Object\_Name
2. RECEIVE Encapsulated-NPDU,  
   'Message ID' = (M2: any valid value),  
   -- 'Originating Virtual Address' absent  
   'Destination Virtual Address' = (OVA: any valid value, including absent),  
   'Destination Options' (absent or any valid value),  
   'Data Options' = ({ X'41' or a list of valid header options including Secure Path } ),  
   'BACnet NPDU' =  
     ReadProperty-ACK,  
     'Object Identifier' = (the IUT's Device object),  
     'Property Identifier' = Object\_Name,  
     'Property Value' = (the IUT's device object name)

#### 12.YY.1.1.15 Advertisement-Solicitation Execution Test

Purpose: To verify that when executing an Advertisement-Solicitation, the IUT correctly initiates Advertisement messages.

Test Concept: With the IUT connected to the primary hub, send an Advertisement-Solicitation to the IUT and verify it responds with a correct Advertisement. Disconnect the primary hub from the network and wait for the IUT to connect to the failover hub. Send an Advertisement-Solicitation to the IUT and verify it responds with a correct Advertisement.

Test Steps:

1. TRANSMIT Advertisement-Solicitation,  
   'Message ID' = (M1: any valid value),  
   'Originating Virtual Address' = (OVA: absent, or any node on the network),  
   -- 'Destination Virtual Address' absent  
   -- 'Destination Options' absent

- 'Data Options' absent
- 2. RECEIVE Advertisement,
  - 'Message ID' = (M2: any valid value),
  - 'Originating Virtual Address' absent
  - 'Destination Virtual Address' = (OVA),
  - 'Destination Options' = (absent or a valid list of options),
  - 'Data Options' absent
  - 'Result for BVLC Function' = X'04', -- Advertisement
  - 'Hub Connection Status' = X'01', -- connected to primary
  - 'Accept Direct Connections' = (as per the IUT's configuration),
  - 'Maximum BVLC Length' = (as per the IUT's configuration),
  - 'Maximum NPDU Length' = (as per the IUT's configuration)
- 3. MAKE(the primary hub go offline)
- 4. WAIT until the IUT connects to the failover hub
- 5. TRANSMIT Advertisement-Solicitation,
  - 'Message ID' = (M3: any valid value),
  - 'Originating Virtual Address' = (OVA: absent, or any node on the network),
  - 'Destination Virtual Address' absent
  - 'Destination Options' absent
  - 'Data Options' absent
- 6. RECEIVE Advertisement,
  - 'Message ID' = (M4: any valid value),
  - 'Originating Virtual Address' absent
  - 'Destination Virtual Address' = (OVA),
  - 'Destination Options' = (absent or a valid list of options),
  - 'Data Options' absent
  - 'Result for BVLC Function' = X'04', -- Advertisement
  - 'Hub Connection Status' = X'02', -- connected to failover
  - 'Accept Direct Connections' = (as per the IUT's configuration),
  - 'Maximum BVLC Length' = (as per the IUT's configuration),
  - 'Maximum NPDU Length' = (as per the IUT's configuration)

### 12.YY.1.1.16 Heartbeat-Request Initiation Test

Purpose: To verify that the IUT initiates heartbeats as per its config.

Test Concept: With the IUT connected to the BACnet/SC network, send a ReadProperty request to the IUT every heartbeat interval / 2 seconds. Verify that the IUT does not initiate a Heartbeat-Request. Stop sending messages to the IUT. Wait the IUT's configured heart-beat interval plus 10 seconds and verify that the IUT sent a Heartbeat-Request, ensuring that no BVLCs are sent to the IUT during that period.

Configuration Requirements: Place the IUT in a mode where it will not initiate requests for a period longer than the heartbeat interval (except for the heartbeat request). If the IUT does not support DM-DCC-B and cannot be otherwise configured to behave in this manner, this test shall be skipped.

Test Steps:

1. REPEAT N = (1..Z) {
  - TRANSMIT Encapsulated-NPDU,
    - 'Message ID' = (M: any valid value),
    - 'Originating Virtual Address' = (OVA: any valid value, including absent),
    - 'Destination Virtual Address' absent
    - 'Destination Options' = (absent or any valid value),
    - 'Data Options' = ({ X'41' }), -- Secure Path
    - 'BACnet NPDU' =
      - ReadProperty-Request,
      - 'Object Identifier' = (the IUT's Device object),
      - 'Property Identifier' = Object\_Name
  - RECEIVE Encapsulated-NPDU,
    - 'Message ID' = M,
    - 'Originating Virtual Address' absent

```

'Destination Virtual Address' = OVA,
'Destination Options' = (absent or any valid value),
'Data Options' = ({ X'41' or a list of valid header options including Secure Path}),
'BACnet NPDU' =
    ReadProperty-ACK,
    'Object Identifier' = (the IUT's Device object),
    'Property Identifier' = Object_Name,
    'Property Value' = (the IUT's device object name)
    WAIT ½ of IUT's heartbeat interval
}

```

2. CHECK(that the IUT did not send a HeartBeat during step 1)

-- Since we already waited ½ of a heartbeat interval, only ½ of that interval is now given for the IUT to

-- generate a Heartbeat-Request

3. BEFORE ½ of IUT's heartbeat interval + 10s

```

RECEIVE Heartbeat-Request,
'Message ID' = M2,
-- 'Originating Virtual Address' absent
-- 'Destination Virtual Address' absent
'Destination Options' = (absent or any valid value),
-- 'Data Options' absent

```

4. TRANSMIT Heartbeat-ACK,

```

'Message ID' = M2,
-- 'Originating Virtual Address' absent
-- 'Destination Virtual Address' absent
'Destination Options' = (absent or any valid value),
-- 'Data Options' absent

```

### 12.YY.1.1.17 Configurable Reconnect Timeout Test

Purpose: To verify that a device adheres to its configurable reconnect timeout.

Test Concept: Turn on the IUT. When the IUT attempts to connect to the primary hub, the primary hub does not respond. Verify that the IUT waits at least the configured reconnect timeout, and no longer than 600 seconds before attempting to reconnect.

Configuration Requirements: The IUT is configured with the TD as the primary hub with no failover hub or as direct connection initiation peer of the TD. The IUT is configured with a tester selected reconnect timeout, RT, within the range supported by the IUT and within 2 .. 300 seconds. The IUT starts the test powered off. If the IUT has a fixed reconnect timeout, this test shall be skipped.

Test Steps:

1. MAKE(the IUT connect to the TD)
2. CHECK(that the IUT attempts to open a new WebSocket with the TD)
3. MAKE(place the TD in a mode where it will accept incoming connections)
4. WAIT RT seconds
5. BEFORE 600 - RT seconds
 

```

RECEIVE PORT (IUT-TD primary hub WebSocket)
Connect-Request,
'Message ID' = (M1: any valid value),
-- 'Originating Virtual Address' absent
-- 'Destination Virtual Address' absent
'Destination Options' = (absent or any valid value),
-- 'Data Options' absent
'VMAC Address' = (IUT's VMAC),
'Device UUID' = (IUT's UUID),
'Maximum BVLC Length' = (the IUT's maximum BVLC accepted length),
'Maximum NPDU Length' = (the IUT's maximum NPDU accepted length)

```
6. TRANSMIT PORT (IUT-TD primary hub WebSocket)
 

```

Connect-Accept,

```



'Message ID' = M1,  
 -- 'Originating Virtual Address' absent  
 -- 'Destination Virtual Address' absent  
 'Destination Options' (absent or any valid value),  
 -- 'Data Options' absent  
 'VMAC Address' = (TD's VMAC),  
 'Device UUID' = (TD's UUID),  
 'Maximum BVLC Length' = (the TD's maximum BVLC accepted length),  
 'Maximum NPDU Length' = (the TD's maximum NPDU accepted length)

### 12.YY.1.1.18 Fixed Reconnect Timeout Test

Purpose: To verify that a device's fixed reconnect timeout is in the range 10 .. 30 seconds.

Test Concept: Turn on the IUT. When the IUT attempts to connect to the TD as the primary hub or as a direct connection peer of the TD, the TD does not respond. Verify that the IUT does not attempt to connect within 10 seconds. Then verify that the IUT does attempt to connect within the following 590 seconds.

Configuration Requirements: The IUT is configured with the TD as the primary hub with no failover hub, or as a direct connection peer of the TD. The IUT starts the test powered off. If the IUT does not have a fixed reconnect timeout this test shall be skipped.

Test Steps:

1. MAKE(the IUT connect to the TD)
2. CHECK(that the IUT attempts to open a new WebSocket with the TD)
3. MAKE(place the TD in a mode where it will accept incoming connections)
4. WAIT 10 seconds
5. BEFORE 590 seconds
  - RECEIVE PORT (IUT-TD primary hub WebSocket)
    - Connect-Request,
    - 'Message ID' = (M1: any valid value),
    - 'Originating Virtual Address' absent
    - 'Destination Virtual Address' absent
    - 'Destination Options' (absent or any valid value),
    - 'Data Options' absent
    - 'VMAC Address' = (IUT's VMAC),
    - 'Device UUID' = (IUT's UUID),
    - 'Maximum BVLC Length' = (the IUT's maximum BVLC accepted length),
    - 'Maximum NPDU Length' = (the IUT's maximum NPDU accepted length)
6. TRANSMIT PORT (IUT-TD primary hub WebSocket)
  - Connect-Accept,
  - 'Message ID' = M1,
  - 'Originating Virtual Address' absent
  - 'Destination Virtual Address' absent
  - 'Destination Options' (absent or any valid value),
  - 'Data Options' absent
  - 'VMAC Address' = (TD's VMAC),
  - 'Device UUID' = (TD's UUID),
  - 'Maximum BVLC Length' = (the TD's maximum BVLC accepted length),
  - 'Maximum NPDU Length' = (the TD's maximum NPDU accepted length)

### 12.YY.1.2 Basic Node Negative Tests

#### 12.YY.1.2.1 Direct Connect Not Supported - NAK Address Resolution Test

Reference YY.3.3

Purpose: To verify that devices configured to not accept direct connect requests will NAK an Address-Resolution request.

Test Concept: With the IUT configured to not accept direct connections, and with it connected to the BACnet/SC network, have device D3 send an Address-Resolution message to the IUT. Verify that the IUT NAKs the request.

Configuration Requirements: The IUT is configured to not accept direct connections.

Test Steps:

1. TRANSMIT Address-Resolution,
  - 'Message ID' = (M1: any valid value)
  - 'Originating Virtual Address' = D3,
  - 'Destination Virtual Address' absent
  - 'Destination Options' = (absent or a valid list of options),
  - 'Data Options' absent
2. RECEIVE BVLC-Result,
  - 'Message ID' = M1,
  - 'Originating Virtual Address' absent
  - 'Destination Virtual Address' = D3,
  - 'Destination Options' = (absent or a valid list of options),
  - 'Data Options' absent
  - 'Result for BVLC Function' = X'02', -- Address-Resolution
  - 'Result Code' = X'01', -- NAK
  - 'Error Header Marker' = X'00', -- not a header option problem
  - 'Error Class' = COMMUNICATION,
  - 'Error Code' = OPTIONAL\_FUNCTIONALITY\_NOT\_SUPPORTED

## 12.YY.1.2.2 Malformed BVLC Test

Reference YY.3.1.5

Purpose: Verify that device NAKs malformed / unknown unicast BVLC and ignores malformed / unknown broadcast BVLC.

Test Concept: With the IUT connected to the BACnet/SC network, send a sequence of malformed unicast and broadcast BVLCs to the IUT. Verify that the IUT responds with an appropriate NAK to each unicast one and does not process nor route the messages.

Configuration Requirements: The IUT is connected to the BACnet/SC network as a node or hub.

Test Steps:

-- Invalid BVLC function

1. TRANSMIT
  - 'BVLC Function' = (IV: an invalid 1-octet value),
  - 'Message ID' = (M1: any valid value),
  - 'Originating Virtual Address' absent
  - 'Destination Virtual Address' absent
  - 'Destination Options' absent
  - 'Data Options' absent
2. RECEIVE BVLC-Result,
  - 'Message ID' = M1,
  - 'Originating Virtual Address' absent
  - 'Destination Virtual Address' absent
  - 'Destination Options' = (absent or a valid list of options),
  - 'Data Options' absent
  - 'Result for BVLC Function' = IV, -- the supplied invalid BVLC function from the request
  - 'Result Code' = X'01', -- NAK
  - 'Error Header Marker' = X'00', -- not a header option problem
  - 'Error Class' = COMMUNICATION,
  - 'Error Code' = BVLC\_FUNCTION\_UNKNOWN
3. CHECK(that the IUT did not process nor forward the request)

-- Inclusion of an Originating Virtual Address when it is required to be absent

4. TRANSMIT Disconnect-Request,
  - 'Message ID' = (M2: any valid value),
  - 'Originating Virtual Address' = D3,
  - 'Destination Virtual Address' absent
  - 'Destination Options' = (absent or a valid list of options),

- 'Data Options' absent
- 5. RECEIVE BVLC-Result,
  - 'Message ID' = M2,
  - 'Originating Virtual Address' absent
  - 'Destination Virtual Address' = D3
  - 'Destination Options' = (absent or a valid list of options),
  - 'Data Options' absent
  - 'Result for BVLC Function' = X'08', -- Disconnect-Request
  - 'Result Code' = X'01', -- NAK
  - 'Error Header Marker' = X'00', -- not a header option problem
  - 'Error Class' = (COMMUNICATION or SERVICES),
  - 'Error Code' = (HEADER\_ENCODING\_ERROR, INCONSISTENT\_PARAMETER, PARAMETER\_OUT\_OF\_RANGE or OTHER)
- 6. CHECK(that the IUT did not process the request)
- Inclusion of a 'Destination Virtual Address when it is required to be absent
- 7. TRANSMIT Disconnect-Request,
  - 'Message ID' = (M3: any valid value),
  - 'Originating Virtual Address' absent,
  - 'Destination Virtual Address' = (IUT's VMAC),
  - 'Destination Options' = (absent or a valid list of options),
  - 'Data Options' absent
- 8. RECEIVE BVLC-Result,
  - 'Message ID' = M3,
  - 'Originating Virtual Address' absent
  - 'Destination Virtual Address' absent
  - 'Destination Options' = (absent or a valid list of options),
  - 'Data Options' absent
  - 'Result for BVLC Function' = X'08', -- Disconnect-Request
  - 'Result Code' = X'01', -- NAK
  - 'Error Header Marker' = X'00', -- not a header option problem
  - 'Error Class' = (COMMUNICATION or SERVICES),
  - 'Error Code' = (HEADER\_ENCODING\_ERROR, INCONSISTENT\_PARAMETER, PARAMETER\_OUT\_OF\_RANGE or OTHER)
- 9. CHECK(that the IUT did not process the request)
- A truncated message
- 10. TRANSMIT Encapsulated-NPDU,
  - 'Message ID' = (M4: any valid value),
  - 'Originating Virtual Address' = (OVA: absent, or D3 if IUT is configured as a hub),
  - 'Destination Virtual Address' = (IUT's VMAC),
  - 'Destination Options' absent
  - 'Data Options' absent
  - no NPDU included in the message
- 11. RECEIVE BVLC-Result,
  - 'Message ID' = M4,
  - 'Originating Virtual Address' absent
  - 'Destination Virtual Address' = OVA,
  - 'Destination Options' = (absent or a valid list of options),
  - 'Data Options' absent
  - 'Result for BVLC Function' = X'01', -- Encapsulated-NPDU
  - 'Result Code' = X'01', -- NAK
  - 'Error Header Marker' = X'00', -- not a header option problem
  - 'Error Class' = COMMUNICATION,
  - 'Error Code' = MESSAGE\_INCOMPLETE | PAYLOAD\_EXPECTED
- 12. CHECK(that the IUT did not process the request)
- A message with extra octets added on
- 13. TRANSMIT Disconnect-Request,

'Message ID' = (M5: any valid value),  
 -- 'Originating Virtual Address' absent  
 -- 'Destination Virtual Address' absent  
 -- 'Destination Options' absent  
 -- 'Data Options' absent  
 (extra octets) = ({ X'C1', --a bunch of octets that look like valid data options.  
 X'BF0003000012',  
 X'3F0003000034'})

#### 14. RECEIVE BVLC-Result,

'Message ID' = M5,  
 -- 'Originating Virtual Address' absent  
 -- 'Destination Virtual Address' absent  
 'Destination Options' = (absent or a valid list of options),  
 -- 'Data Options' absent  
 'Result for BVLC Function' = X'08', -- Disconnect-Request  
 'Result Code' = X'01', -- NAK  
 'Error Header Marker' = X'00', -- not a header option problem  
 'Error Class' = COMMUNICATION,  
 'Error Code' = UNEXPECTED\_DATA

#### 15. CHECK(that the IUT did not process the request)

-- A truncated broadcast message

#### 16. TRANSMIT Encapsulated-NPDU,

DESTINATION = GLOBAL\_BROADCAST,  
 'Message ID' = (M6: any valid value),  
 -- 'Originating Virtual Address' absent,  
 'Destination Virtual Address' = (local broadcast VMAC),  
 -- 'Destination Options' absent  
 'Data Options' = ({ X'E1' -- Secure Path,  
 more follows (but none are present)  
 })

#### 17. CHECK(that the IUT does not send a BVLC-Result, did not process the message, nor route the message)

### 12.YY.1.2.3 Discard BVLC with Wrong Address Test

Purpose: To verify that BVLCs with an incorrect VMAC are dropped.

Test Concept: With the IUT connected to the BACnet/SC network, send a ReadProperty to the IUT in an invalid BVLC message with a BVLC Destination Address that does not match the IUT. Verify that the IUT does not respond with a BVLC NAK nor with a ReadProperty-ACK.

Test Steps:

#### 1. TRANSMIT Encapsulated-NPDU,

'Message ID' = (M1: any valid value),  
 -- 'Originating Virtual Address' (absent or any valid VMAC)  
 'Destination Virtual Address' = (any non-broadcast VMAC other than the IUT's),  
 -- 'Destination Options' absent  
 'Data Options' = ({X'41'}), -- Secure Path  
 'BACnet NPDU' =  
 ReadProperty-Request,  
 'Object Identifier' = (O: any object in the IUT),  
 'Property Identifier' = Object\_Name

#### 2. CHECK(that the IUT does not response with a BVLC-Result nor a ReadProperty-ACK)

### 12.YY.1.2.4 Hub Connector Ignores Malformed Hub URIs Test

Purpose: Verify that the IUT's hub connector will not attempt to connect to malformed URIs or URIs with a scheme other than wss.

Test Concept: Configure the IUT with a primary hub URI with an invalid scheme or otherwise invalid URI. Verify that the IUT does not attempt to connect to the invalid URI.

Configuration Requirements: If the IUT cannot be configured with an invalid hub URI this test shall be skipped.

Test Steps:

1. MAKE(configure the primary hub URI in the IUT with a URI having a scheme other than wss)
2. CHECK(that the IUT does not attempt to connect to the URI)

#### 12.YY.1.2.5 Connect-Request Response Wait Time Test

Purpose: To verify that the IUT will close the WebSocket if a response to a Connect-Request is not received before the connection wait timer expires.

Test Concept: Turn on the IUT. When the IUT attempts to connect to the TD as the primary hub or as a direct connection peer, the TD will accept the WebSocket connection but will not send a response to the connect request. It is verified that the IUT closes the WebSocket when the connection wait timer expires.

Configuration Requirements: The IUT is configured with the TD as the primary hub, or as a direct connect peer. The TD is configured to accept WebSocket connections but to not respond to Connect-Requests.

Test Steps:

1. MAKE(the IUT connect to the TD)
2. CHECK(that the IUT attempts to open a new WebSocket with the TD)
3. RECEIVE Connect-Request,
  - 'Message ID' = (M1: any valid value),
  - 'Originating Virtual Address' absent
  - 'Destination Virtual Address' absent
  - 'Destination Options' (absent or any valid value),
  - 'Data Options' absent
  - 'VMAC Address' = (IUT's VMAC),
  - 'Device UUID' = (IUT's UUID),
  - 'Maximum BVLC Length' = (the IUT's maximum BVLC accepted length),
  - 'Maximum NPDU Length' = (the IUT's maximum NPDU accepted length)
4. WAIT connect wait timeout
5. CHECK(that the IUT closed the WebSocket)

#### 12.YY.1.2.6 HTTP 1.1 Fallback Test

Purpose: To verify that the IUT is able to fallback to HTTP 1.1.

Test Concept: With the IUT configured to connect to the TD as the primary hub, turn on the IUT and when it attempts to connect, have the TD indicate that it only supports HTTP 1.1 as specified in RFC 6455. Verify that the IUT falls back to using HTTP 1.1.

Configuration Requirements: The IUT is configured to use HTTP 2 or later. If the IUT cannot be configured to do so, this test shall be skipped. The TD is configured to only support HTTP 1.1.

Test Steps:

1. MAKE(the IUT connect to the TD)
2. CHECK(that the IUT attempts to connect with HTTP 2 or later)
3. CHECK(that the IUT successfully connects to the TD using HTTP 1.1)
4. RECEIVE Connect-Request,
  - 'Message ID' = (M1: any valid value),
  - 'Originating Virtual Address' absent
  - 'Destination Virtual Address' absent
  - 'Destination Options' (absent or any valid value),
  - 'Data Options' absent
  - 'VMAC Address' = (IUT's VMAC),
  - 'Device UUID' = (IUT's UUID),
  - 'Maximum BVLC Length' = (the IUT's maximum BVLC accepted length),

- 'Maximum NPDU Length' = (the IUT's maximum NPDU accepted length)
5. TRANSMIT Connect-Accept,
    - 'Message ID' = M1,
    - 'Originating Virtual Address' absent
    - 'Destination Virtual Address' absent
    - 'Destination Options' absent
    - 'Data Options' absent
    - 'VMAC Address' = (TD's VMAC),
    - 'Device UUID' = (TD's UUID),
    - 'Maximum BVLC Length' = (the TD's maximum BVLC accepted length),
    - 'Maximum NPDU Length' = (the TD's maximum NPDU accepted length)

### 12.YY.1.2.7 Rejection of Invalid Certificate Outgoing Connection Test

Purpose: To verify that the IUT will drop initiated connection attempts if the peer's certificate is invalid.

Test Concept: With the IUT configured to connect to the TD as the primary hub or as a peer via a direct connection, allow the IUT to attempt the connection. The TD presents an invalid certificate during the connection. Verify that the WebSocket is not established.

Configuration Requirements: The TD is configured with an invalid certificate, or a certificate signed by a Certificate Authority which is not one recognized by the IUT. The TD shall be configured to accept the IUT's certificate.

Test Steps:

1. MAKE(the IUT connect to the TD)
2. CHECK(that the IUT initiated a WebSocket connection)
3. CHECK(that the WebSocket connection was failed by the IUT)

### 12.YY.1.2.8 No Additional Certificate Checks Performed Test On Outgoing Connections

Purpose: Verify that the IUT does not apply checks beyond the 4 listed in YY.7.4 when not configured to do so.

Test Concept: With the IUT configured to connect to the TD as the primary hub, or as a peer via a direct connection, allow the IUT to attempt to connect. During the connection process the TD presents a valid certificate with field that would be caught if the IUT applied validation steps outside of those listed in YY.7.4.

Configuration Requirements: The IUT is configured to not perform any additional certificate checks beyond those identified in YY.7.4. The TD is configured with a certificate with a Common Name, Distinguished Name, or Subject Alternate Name which does not match the TD device.

Test Steps:

1. MAKE(the IUT connect to the TD)
2. CHECK(that the IUT attempts to connect a WebSocket)
4. RECEIVE Connect-Request,
  - 'Message ID' = (M1: any valid value),
  - 'Originating Virtual Address' absent
  - 'Destination Virtual Address' absent
  - 'Destination Options' (absent or any valid value),
  - 'Data Options' absent
  - 'VMAC Address' = (IUT's VMAC),
  - 'Device UUID' = (IUT's UUID),
  - 'Maximum BVLC Length' = (the IUT's maximum BVLC accepted length),
  - 'Maximum NPDU Length' = (the IUT's maximum NPDU accepted length)
5. TRANSMIT Connect-Accept,
  - 'Message ID' = M1,
  - 'Originating Virtual Address' absent
  - 'Destination Virtual Address' absent
  - 'Destination Options' absent
  - 'Data Options' absent
  - 'VMAC Address' = (TD's VMAC),
  - 'Device UUID' = (TD's UUID),



### 12.YY.1.3.3 Factory Defaults Test

Purpose: To verify that the IUT can be reset to factory defaults and that operational certificates and sensitive data are removed.

Test Concept: The IUT is configured to connect to the BACnet/SC network. Verify that the IUT can connect. Using the IUT's documented method for returning it to factory defaults, reset the IUT to factory defaults. Reset the IUT. Using the IUT's configuration tool or another vendor identified method, verify that the IUT no longer has its private data. Reset the IUT and verify that it does not attempt to connect to the previously configured hub. If the IUT accepts direct connections, attempt to connect to the previously configured IUT direct connect URL. Verify that the connection does not succeed. If the IUT was configured as a hub, verify that it does not accept connections using a previously acceptable certificate.

Test Steps:

1. MAKE(the IUT connect to the TD)
2. READ ON = (Device, IUT), Object\_Name
3. MAKE(the IUT reset to factory defaults)
4. CHECK(that any private data viewable through the configuration tool has been discarded, including the hub URIs)
5. MAKE(reset the IUT)
6. MAKE(set the primary hub URI to the TD's hub URI)
7. MAKE(if the BACnet/SC Network Port has been disabled, enable it)
8. IF (the BACnet/SC port works when enabled without providing it with a certificate) THEN {  
CHECK(that the IUT is not able to connect to the TD due to not having a certificate signed by a mutually agreed upon CA)  
}

### 12.YY.2 Hub Tests

This clause contains test for BACnet/SC hub functions, both primary and failover.

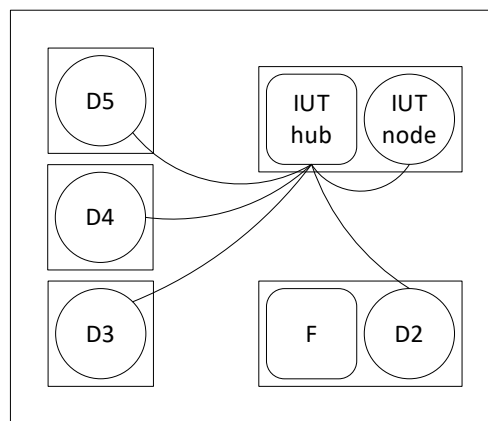


Figure X3: Network setup for hub function tests when the IUT is playing the role of the primary hub.



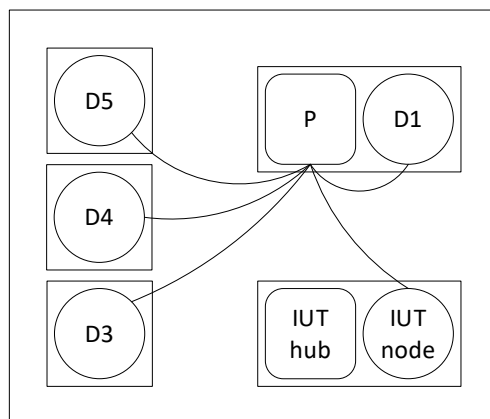


Figure X4: Network setup for hub function tests when the IUT is playing the role of the failover hub.

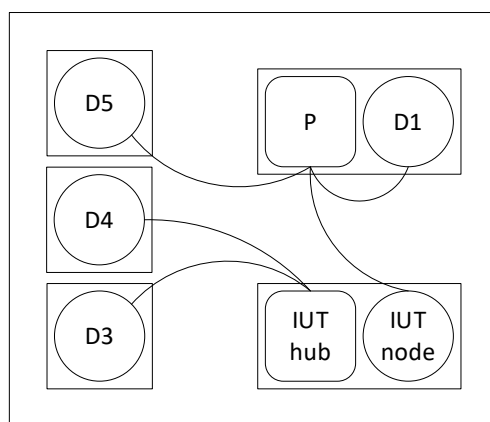


Figure X5: Network setup for hub function tests when the IUT is playing the role of the failover hub and not all nodes have reconnected to the primary.

## 12.YY.2.1 Hub Positive Tests

### 12.YY.2.1.1 Local Broadcast Initiation Test

Purpose: To verify that the IUT, as a hub, correctly initiates broadcast messages.

Test Concept: With IUT operating as a hub, make the IUT's node generate a broadcast message. Verify that the broadcast is sent correctly to all nodes attached to the hub.

Configuration Requirements: Configure the IUT as a hub. If the IUT does not support initiation of broadcast messages, this test shall be skipped.

Test Steps:

1. MAKE(the IUT send a broadcast NPDU)
2. REPEAT  $D_x = (D2, D3, D4)$  {
 

RECEIVE PORT ( $D_x$ -IUT hub WebSocket),  
 Encapsulated-NPDU,  
 'Originating Virtual Address' = (IUT's VMAC),  
 'Destination Virtual Address' = X'FFFFFFF' -- the local broadcast VMAC  
 'Destination Options' = (absent or any valid value),  
 'Data Options' = (secure path, plus 0 more data options),  
 'Payload' = (a well formed NPDU)
3. CHECK(that all of the NPDUs received in the previous step are the same)

### 12.YY.2.1.2 Local Broadcast Execution Test

Purpose: To verify that IUT, as a hub, correctly accepts and processes broadcast messages.

Test Concept: With the IUT operating as a hub, send a broadcast to the hub. Verify that the message is forwarded to all hub connectors except the one that originated it. Also verify that the hub's local node processes the broadcast.

Configuration Requirements: The IUT is operating as a hub and devices D2, D3, and D4 are connected to it.

Notes to Tester: The order of the broadcasts sent by the hub and the I-Am response can be sent in any order.

Test Steps:

1. TRANSMIT PORT (D4-IUT hub WebSocket),  
     Encapsulated-NPDU,  
     -- 'Originating Virtual Address' absent  
     'Destination Virtual Address' = X'FFFFFFF', -- the local broadcast VMAC  
     -- 'Destination Options' absent  
     'Data Options' = ({X'41'}), -- Secure Path  
     'Payload' Who-Is-Request
2. REPEAT Dx = (D2, D3) {  
     RECEIVE PORT (Dx-IUT hub WebSocket),  
     Encapsulated-NPDU,  
     'Originating Virtual Address' = (D4's VMAC),  
     'Destination Virtual Address' = X'FFFFFFF', -- the local broadcast VMAC  
     -- 'Destination Options' absent  
     'Data Options' = ({X'41'}), -- Secure Path  
     'Payload' Who-Is-Request  
   }
3. RECEIVE PORT (D4-IUT hub WebSocket),  
     Encapsulated-NPDU,  
     'Originating Virtual Address' = (IUT's VMAC)  
     'Destination Virtual Address' = (D4's VMAC or X'FFFFFFF', the local broadcast VMAC)  
     -- 'Destination Options' absent  
     'Data Options' = ({X'41'}), -- Secure Path  
     'Payload' I-Am-Request,  
     'I Am Device Identifier' = (the IUT's Device object),  
     'Max APDU Length Accepted' = (the value specified in the EPICS),  
     'Segmentation Supported' = (the value specified in the EPICS),  
     'Vendor Identifier' = (the identifier registered for this vendor)

### 12.YY.2.1.3 Minimum NPDU Forwarding Size Test

Purpose: To verify that the hub can forward BVLC messages of length 1497 with 4192 octets of data and destination options.

Test Concept: With the IUT operating as the primary hub, connect devices D3 and D4 to the IUT. D3 sends a BVLC of length 1497 octets with 4192 octets of data options to D4 via the hub. Verify that the BVLC is correctly forwarded to D4.

Configuration Requirements: The IUT is configured as a primary or failover hub and the test devices D3, D4, and D5 are connected to it.

Test Steps:

1. TRANSMIT PORT (D3-IUT hub WebSocket),  
     Encapsulated-NPDU,  
     -- 'Originating Virtual Address' absent  
     'Destination Virtual Address' = (D4's VMAC),  
     -- 'Destination Options' absent  
     'Data Options' = ({X'C1',  
                           X'3F105A000034...'  
                           }), -- replace ... with 4186 octets of any value  
     'Payload'

WriteProperty-Request,  
 'Object Identifier' = (O: any object identifier),  
 'Property Identifier' = (P: any property identifier),  
 'Property Value' = (V: any data value with an encoded length which makes the PayLoad 1497 octets)

2. RECEIVE PORT (D4-IUT hub WebSocket),

Encapsulated-NPDU,  
 'Originating Virtual Address' = (D3's VMAC),  
 -- 'Destination Virtual Address' absent  
 -- 'Destination Options' absent  
 'Data Options' = ({X'C1',  
 X'3F105A000034...'  
 }), -- replace ... with 4186 octets of any value  
 'Payload'  
 WriteProperty-Request,  
 'Object-Identifier' = O,  
 'Property-Identifier' = P,  
 'Property Value' = V

#### 12.YY.2.1.4 Failover Hub Connects to Primary Hub Test

Purpose: To verify that when the IUT is operating as a failover hub, the IUT's node connects to the primary hub.

Test Concept: The IUT is configured as a failover hub, and the IUT's node has another hub set as its primary hub. Verify that the IUT connects to its primary hub.

Configuration Requirements: The IUT is configured as a failover hub with the TD set as its primary hub.

Test Steps:

1. MAKE(the IUT connect to the primary hub)
2. CHECK(that the IUT opens a WebSocket with the TD)
3. RECEIVE PORT (IUT-TD hub WebSocket),  
 Connect-Request,  
 'Message ID' = (M1: any valid value),  
 -- 'Originating Virtual Address' absent  
 -- 'Destination Virtual Address' absent  
 'Destination Options' (absent or any valid value),  
 -- 'Data Options' absent  
 'VMAC Address' = (IUT's VMAC),  
 'Device UUID' = (IUT's UUID),  
 'Maximum BVLC Length' = (the IUT's maximum BVLC accepted length),  
 'Maximum NPDU Length' = (the IUT's maximum NPDU accepted length)
4. TRANSMIT PORT (IUT-TD hub WebSocket),  
 Connect-Accept,  
 'Message ID' = M1,  
 -- 'Originating Virtual Address' absent  
 -- 'Destination Virtual Address' absent  
 'Destination Options' (absent or any valid value),  
 -- 'Data Options' absent  
 'VMAC Address' = (TD's VMAC),  
 'Device UUID' = (TD's UUID),  
 'Maximum BVLC Length' = (the TD's maximum BVLC accepted length),  
 'Maximum NPDU Length' = (the TD's maximum NPDU accepted length)

#### 12.YY.2.1.5 Failover Hub's Local Node Connects to Failover Hub Test

Purpose: To verify that, when operating as a failover hub, the IUT's local node connects to the local hub connection when the primary is offline.

Test Concept: The IUT is configured as the IUT's local node's failover hub, and the IUT's local node has the TD set as its primary hub. Verify that the IUT connects to its primary hub. Disconnect the primary from the network. Wait for the IUT to

recognize the primary is offline. Connect D3 to the IUT's hub connection. Make the IUT's local node send a message to D3. Verify that the message is sent from the IUT's hub connection.

Configuration Requirements: The IUT is configured as a failover hub with the TD set as its local node's primary hub. The TD is configured to be a hub.

Test Steps:

1. MAKE(the IUT connect to the primary hub)
2. CHECK(that the IUT opens a WebSocket with the TD)
3. RECEIVE PORT (IUT-TD hub WebSocket),
  - Connect-Request,
  - 'Message ID' = (M1: any valid value),
  - 'Originating Virtual Address' absent
  - 'Destination Virtual Address' absent
  - 'Destination Options' (absent or any valid value),
  - 'Data Options' absent
  - 'VMAC Address' = (IUT's VMAC),
  - 'Device UUID' = (IUT's UUID),
  - 'Maximum BVLC Length' = (the IUT's maximum BVLC accepted length),
  - 'Maximum NPDU Length' = (the IUT's maximum NPDU accepted length)
4. TRANSMIT PORT (IUT-TD hub WebSocket),
  - Connect-Accept,
  - 'Message ID' = M1,
  - 'Originating Virtual Address' absent
  - 'Destination Virtual Address' absent
  - 'Destination Options' (absent or any valid value),
  - 'Data Options' absent
  - 'VMAC Address' = (TD's VMAC),
  - 'Device UUID' = (TD's UUID),
  - 'Maximum BVLC Length' = (the TD's maximum BVLC accepted length),
  - 'Maximum NPDU Length' = (the TD's maximum NPDU accepted length)
5. WAIT a tester selected amount of time
6. MAKE(the TD's hub function stop)
7. WAIT 2 times the IUT's HeartBeat timeout
8. MAKE(D3 open a WebSocket with IUT's hub URI)
9. TRANSMIT PORT (D3-IUT hub WebSocket),
  - Connect-Request,
  - 'Message ID' = (M2: any valid value),
  - 'Originating Virtual Address' absent
  - 'Destination Virtual Address' absent
  - 'Destination Options' absent
  - 'Data Options' absent
  - 'VMAC Address' = (D3's VMAC),
  - 'Device UUID' = (D3's UUID),
  - 'Maximum BVLC Length' = (the D3's maximum BVLC accepted length),
  - 'Maximum NPDU Length' = (the D3's maximum NPDU accepted length)
10. RECEIVE PORT (D3-IUT hub WebSocket),
  - Connect-Accept,
  - 'Message ID' = M2,
  - 'Originating Virtual Address' absent
  - 'Destination Virtual Address' absent
  - 'Destination Options' (absent or any valid value),
  - 'Data Options' absent
  - 'VMAC Address' = (IUT's VMAC),
  - 'Device UUID' = (IUT's UUID),
  - 'Maximum BVLC Length' = (the IUT's maximum BVLC accepted length),
  - 'Maximum NPDU Length' = (the IUT's maximum NPDU accepted length)
11. MAKE(the IUT's node send an NPDU to D3)
12. RECEIVE PORT (D3-IUT hub WebSocket),
  - Encapsulated-NPDU,
  - 'Message ID' = (M3: any valid value),

'Originating Virtual Address' = (IUT's VMAC),  
 'Destination Virtual Address' = (absent or local broadcast)  
 'Destination Options' = (absent or any valid value),  
 'Data Options' = (Secure Path + 0 or more valid data options),  
 'Payload' = (any valid NPDU)

#### 12.YY.2.1.6 Failover Hub Split Horizon Test

Purpose: To verify that the IUT, operating as a failover hub, continues to perform hub functions when the IUT node is connected to the primary hub.

Test Concept: The IUT is configured as a failover hub and is connected to the TD acting as the primary hub. Connect device D3 to the IUT's hub URI. Verify that the IUT accepts the connection. Connect device D4 to the IUT's hub function. Verify that the IUT accepts the connection. D3 sends a broadcast to the IUT's hub function. Verify that the broadcast is properly forwarded to D4 and no other nodes. Verify that the IUT's node does not receive the broadcast. Connect D5 to the primary hub. D5 sends an Advertisement-Solicitation message to ensure that the IUT is aware of its existence. D3 sends a unicast destined for D5 to the hub for forwarding. Verify that the IUT does not forward the request.

Configuration Requirements: The IUT is configured as a failover hub with the TD set as its primary hub. The TD is configured to be a primary hub. The IUT's node is connected to the primary hub.

Test Steps:

1. CHECK(that the IUT's node is connected to the TD's hub function)
2. MAKE(D3 open a WebSocket with IUT's hub URI)
3. TRANSMIT PORT (D3-IUT hub WebSocket),  
 Connect-Request,  
 'Message ID' = (M1: any valid value),  
 -- 'Originating Virtual Address' absent  
 -- 'Destination Virtual Address' absent  
 -- 'Destination Options' absent  
 -- 'Data Options' absent  
 'VMAC Address' = (D3's VMAC),  
 'Device UUID' = (D3's UUID),  
 'Maximum BVLC Length' = (the D3's maximum BVLC accepted length),  
 'Maximum NPDU Length' = (the D3's maximum NPDU accepted length)
4. RECEIVE PORT (D3-IUT hub WebSocket),  
 Connect-Accept,  
 'Message ID' = M1,  
 -- 'Originating Virtual Address' absent  
 -- 'Destination Virtual Address' absent  
 'Destination Options' = (absent or any valid value),  
 -- 'Data Options' absent  
 'VMAC Address' = (IUT's VMAC),  
 'Device UUID' = (IUT's UUID),  
 'Maximum BVLC Length' = (the IUT's maximum BVLC accepted length),  
 'Maximum NPDU Length' = (the IUT's maximum NPDU accepted length)
5. TRANSMIT PORT (D4-IUT hub WebSocket),  
 Connect-Request,  
 'Message ID' = (M2: any valid value),  
 -- 'Originating Virtual Address' absent  
 -- 'Destination Virtual Address' absent  
 -- 'Destination Options' absent  
 -- 'Data Options' absent  
 'VMAC Address' = (D4's VMAC),  
 'Device UUID' = (D4's UUID),  
 'Maximum BVLC Length' = (the D4's maximum BVLC accepted length),  
 'Maximum NPDU Length' = (the D4's maximum NPDU accepted length)
6. RECEIVE PORT (D4-IUT hub WebSocket),  
 Connect-Accept,  
 'Message ID' = M2,  
 -- 'Originating Virtual Address' absent

```

-- 'Destination Virtual Address' absent
'Destination Options'          (absent or any valid value),
-- 'Data Options' absent
'VMAC Address' =               (IUT's VMAC),
'Device UUID' =               (IUT's UUID),
'Maximum BVLC Length' =       (the IUT's maximum BVLC accepted length),
'Maximum NPDU Length' =       (the IUT's maximum NPDU accepted length)

-- verify that D3 and D4 can communicate with each other and that the IUT's local node does not receive
-- broadcasts generated by D3
7.  TRANSMIT PORT (D3-IUT hub WebSocket),
    Encapsulated-NPDU,
    'Message ID' =              (M3: any valid value),
    -- 'Originating Virtual Address' absent
    'Destination Virtual Address' = (X'FFFFFFFF' local broadcast VMAC),
    -- 'Destination Options' absent
    'Data Options' =            ({X'41'}),      -- Secure Path
    'Payload' =
        DNET =                  GLOBAL_BROADCAST,
        Who-Is-Request
8.  RECEIVE PORT (D4-IUT hub WebSocket),
    Encapsulated-NPDU,
    'Message ID' =              (M3: any valid value),
    'Originating Virtual Address' = (D3's VMAC),
    'Destination Virtual Address' = (X'FFFFFFFF' local broadcast VMAC),
    -- 'Destination Options' absent
    'Data Options' =            ({X'41'}),      -- Secure Path
    'Payload' =
        DNET =                  GLOBAL_BROADCAST,
        Who-Is-Request
9.  CHECK(that the IUT does not generate an I-Am for its local node)

-- verify that D5, connected to the primary, and the IUT's node can communicate
10. TRANSMIT PORT (IUT-TD hub WebSocket),
    Advertisement-Solicitation,
    'Message ID' =              (M4: any valid value),
    'Originating Virtual Address' = (D5's VMAC),
    -- 'Destination Virtual Address' absent
    -- 'Destination Options' absent
    -- 'Data Options' absent

11. RECEIVE PORT (IUT-TD hub WebSocket),
    Advertisement,
    'Message ID' =              (M5: any valid value),
    -- 'Originating Virtual Address' absent
    'Destination Virtual Address' = (D5's VMAC),
    'Destination Options'        (absent, or a valid list of options),
    -- 'Data Options' absent
    'Payload'
        'Hub Connection Status' = X'01',      -- connected to the primary hub
        'Accept Direct Connections' = (as per the IUT's configuration),
        'Maximum BVLC Length' = (as per the IUT's configuration),
        'Maximum NPDU Length' = (as per the IUT's configuration)

-- verify that D3, connected to the failover, and D5, connected to the primary, cannot communicate
12. TRANSMIT PORT (D3-IUT hub WebSocket),
    Advertisement-Solicitation,
    'Message ID' =              (M6: any valid value),
    -- 'Originating Virtual Address' absent
    'Destination Virtual Address' = (D5's VMAC),

```

- 'Destination Options' absent
- 'Data Options' absent

13. CHECK(that the IUT does not forward the Advertisement-Solicitation)

-- verify that D3, connected to the failover, and the IUT's node, connected to the primary, cannot communicate

14. TRANSMIT PORT(D3-IUT hub WebSocket),  
 Advertisement-Solicitation,  
 'Message ID' = (M7: any valid value),  
 -- 'Originating Virtual Address' absent  
 -- 'Destination Virtual Address' absent  
 -- 'Destination Options' absent  
 -- 'Data Options' absent

15. CHECK(that the IUT does not generate an Advertisement)

### 12.YY.2.1.7 Hub Forwards Unicast BVLCs Test

Purpose: To verify that a hub correctly forwards unicast BVLCs received on any current hub connection.

Test Concept: The IUT is operating as the primary hub. D3 sends a unicast to D4 via the hub. Verify that the IUT correctly forwards the message to D4.

Configuration Requirements: The IUT is configured as the primary hub. D3 and D4 are connected to the IUT's hub function.

Test Steps:

-- verify that D3 and D4 can communicate with each other

1. TRANSMIT PORT (D3-IUT hub WebSocket),  
 Encapsulated-NPDU,  
 'Message ID' = (M1: any valid value),  
 -- 'Originating Virtual Address' absent  
 'Destination Virtual Address' = (D4's VMAC),  
 -- 'Destination Options' absent  
 'Data Options' = ({X'41'}), -- Secure Path  
 'Payload' =  
 Who-Is-Request
2. RECEIVE PORT (D4-IUT hub WebSocket),  
 Encapsulated-NPDU,  
 'Message ID' = (M1: any valid value),  
 'Originating Virtual Address' = (D3's VMAC),  
 -- 'Destination Virtual Address' absent  
 -- 'Destination Options' absent  
 'Data Options' = ({X'41'}), -- Secure Path  
 'Payload' =  
 Who-Is-Request
3. CHECK(that the Encapsulated-NPDU is not forwarded to any other devices, is not routed, and that the IUT does not generate an I-Am)

### 12.YY.2.1.8 No Additional Certificate Checks Performed Test On Incoming Connections

Purpose: Verify that the IUT does not apply checks beyond the 4 listed in YY.7.4 when not configured to do so.

Test Concept: With the IUT configured to operate as a hub. The TD attempts to connect to the IUT with a valid certificate with field that would be caught if the IUT applied validation steps outside of those listed in YY.7.4.

Configuration Requirements: The IUT is configured to operate as a hub. The TD is configured with a certificate with a Common Name, Distinguished Name, or Subject Alternate Name which does not match the TD device.

Test Steps:

1. MAKE(the TD open a WebSocket to the IUT's hub function)
2. TRANSMIT PORT (TD-IUT hub WebSocket)

Connect-Request,  
'Message ID' = (M1: any valid value),  
-- 'Originating Virtual Address' absent  
-- 'Destination Virtual Address' absent  
-- 'Destination Options' absent  
-- 'Data Options' absent  
'VMAC Address' = (TD's VMAC),  
'Device UUID' = (TD's UUID),  
'Maximum BVLC Length' = (the TD's maximum BVLC accepted length),  
'Maximum NPDU Length' = (the TD's maximum NPDU accepted length)

3. RECEIVE PORT (TD-IUT hub WebSocket)

Connect-Accept,  
'Message ID' = M1,  
-- 'Originating Virtual Address' absent  
-- 'Destination Virtual Address' absent  
'VMAC Address' = (IUT's VMAC),  
'Device UUID' = (IUT's UUID),  
'Maximum BVLC Length' = (the IUT's maximum BVLC accepted length),  
'Maximum NPDU Length' = (the IUT's maximum NPDU accepted length)

### 12.YY.2.1.9 Duplicate Connection Test

Purpose: To verify that duplicate hub connection requests result in the original connection being dropped.

Test Concept: With the IUT operating as hub, connect device D3 to the IUT's hub URI. When the connection is complete, D3 attempts to bring up a second connection to the IUT's hub URI. Verify that the IUT accepts the second connect request and closes the first connection. Repeat the reconnection, but with a new VMAC for D3 and ensure that the new request is accepted and the existing one dropped.

Test Steps:

1. MAKE(D3 connect to the IUT's hub function)
2. TRANSMIT PORT (D3-IUT hub first WebSocket),  
Connect-Request,  
'Message ID' = (M1: any valid value),  
-- 'Originating Virtual Address' absent  
-- 'Destination Virtual Address' absent  
-- 'Destination Options' absent  
-- 'Data Options' absent  
'VMAC Address' = (D3's VMAC),  
'Device UUID' = (D3's UUID),  
'Maximum BVLC Length' = (the D3's maximum BVLC accepted length),  
'Maximum NPDU Length' = (the D3's maximum NPDU accepted length)
3. RECEIVE PORT (D3-IUT hub first WebSocket),  
Connect-Accept,  
'Message ID' = M1,  
-- 'Originating Virtual Address' absent  
-- 'Destination Virtual Address' absent  
'VMAC Address' = (IUT's VMAC),  
'Device UUID' = (IUT's UUID),  
'Maximum BVLC Length' = (the IUT's maximum BVLC accepted length),  
'Maximum NPDU Length' = (the IUT's maximum NPDU accepted length)
4. MAKE(D1 connect a second WebSocket to the IUT's hub function)
5. TRANSMIT PORT (D3-IUT hub second WebSocket),  
Connect-Request,  
'Message ID' = (M2: any valid value),  
-- 'Originating Virtual Address' absent  
-- 'Destination Virtual Address' absent  
-- 'Destination Options' absent  
-- 'Data Options' absent  
'VMAC Address' = (D3's VMAC),



- 'Device UUID' = (D3's UUID),
- 'Maximum BVLC Length' = (the D3's maximum BVLC accepted length),
- 'Maximum NPDU Length' = (the D3's maximum NPDU accepted length)
- 6. RECEIVE PORT (D3-IUT hub second WebSocket),
  - Connect-Accept,
  - 'Message ID' = M2,
  - 'Originating Virtual Address' absent
  - 'Destination Virtual Address' absent
  - 'Destination Options' = (absent or a list of valid options),
  - 'Data Options' absent
  - 'VMAC Address' = (IUT's VMAC),
  - 'Device UUID' = (IUT's UUID),
  - 'Maximum BVLC Length' = (the IUT's maximum BVLC accepted length),
  - 'Maximum NPDU Length' = (the IUT's maximum NPDU accepted length)
- 7. RECEIVE PORT (D3-IUT hub first WebSocket),
  - Disconnect-Request,
  - 'Message ID' = M3,
  - 'Originating Virtual Address' absent
  - 'Destination Virtual Address' absent
  - 'Destination Options' = (absent or a list of valid options),
  - 'Data Options' absent
- 8. TRANSMIT PORT (D3-IUT hub first WebSocket),
  - Disconnect-ACK,
  - 'Message ID' = M3,
  - 'Originating Virtual Address' absent
  - 'Destination Virtual Address' absent
  - 'Destination Options' absent
  - 'Data Options' absent
- 9. CHECK(that the IUT closed D3's initial WebSocket)
- 10. MAKE(D3 connect a third WebSocket to the IUT's hub function)
- 11. TRANSMIT PORT (D3-IUT hub second WebSocket),
  - Connect-Request,
  - 'Message ID' = (M4: any valid value),
  - 'Originating Virtual Address' absent
  - 'Destination Virtual Address' absent
  - 'Destination Options' absent
  - 'Data Options' absent
  - 'VMAC Address' = (a new VMAC for D3 which does not conflict with any other VMACs),
  - 'Device UUID' = (D3's UUID),
  - 'Maximum BVLC Length' = (the D3's maximum BVLC accepted length),
  - 'Maximum NPDU Length' = (the D3's maximum NPDU accepted length)
- 12. RECEIVE PORT (D3-IUT hub third WebSocket),
  - Connect-Accept,
  - 'Message ID' = M4,
  - 'Originating Virtual Address' absent
  - 'Destination Virtual Address' absent
  - 'Destination Options' = (absent or a list of valid options),
  - 'Data Options' absent
  - 'VMAC Address' = (IUT's VMAC),
  - 'Device UUID' = (IUT's UUID),
  - 'Maximum BVLC Length' = (the IUT's maximum BVLC accepted length),
  - 'Maximum NPDU Length' = (the IUT's maximum NPDU accepted length)
- 13. RECEIVE PORT (D3-IUT hub second WebSocket),
  - Disconnect-Request,
  - 'Message ID' = M5,
  - 'Originating Virtual Address' absent
  - 'Destination Virtual Address' absent
  - 'Destination Options' = (absent or a list of valid options),
  - 'Data Options' absent
- 14. TRANSMIT PORT (D3-IUT hub second WebSocket),

Disconnect-ACK,  
'Message ID' = M5,  
-- 'Originating Virtual Address' absent  
-- 'Destination Virtual Address' absent  
-- 'Destination Options' absent  
-- 'Data Options' absent

15. CHECK(that the IUT closed D3's second WebSocket)

#### 12.YY.2.1.10 Heartbeat-Request Execution Test

Purpose: To verify that the IUT accepts and responds to Heartbeat-Requests.

Test Concept: With the IUT operating as a hub, the TD connects to the IUT. The TD sends a Heartbeat-Request to the IUT. Verify that the IUT responds with a Heartbeat-ACK.

Configuration Requirements: The IUT is configured as a hub, and the TD is connected to it.

Test Steps:

1. TRANSMIT PORT (TD-IUT hub WebSocket),  
Heartbeat-Request,  
'Message ID' = (M1: any valid value),  
-- 'Originating Virtual Address' absent  
-- 'Destination Virtual Address' absent  
-- 'Destination Options' absent  
-- 'Data Options' absent
2. RECEIVE PORT (TD-IUT hub WebSocket),  
Heartbeat-ACK,  
'Message ID' = M1,  
-- 'Originating Virtual Address' absent  
-- 'Destination Virtual Address' absent  
'Destination Options' = (absent or a valid list of options),  
-- 'Data Options' absent

#### 12.YY.2.2 Hub Negative Tests

##### 12.YY.2.2.1 Hub Discards BVLCs with Non-connected VMAC Test

Reference YY.5.1, YY.5.3.3

Purpose: To verify that a hub will drop received BVLCs with a destination VMAC not connected to the hub.

Test Concept: With the IUT operating as the primary hub, connect to the IUT and send a message to a VMAC which is not connected to the IUT. Verify that the BVLC is silently dropped.

Configuration Requirements: The IUT is configured as a hub and device D3 is connected but device D4 is not.

Test Steps:

1. TRANSMIT PORT (D3-IUT hub WebSocket),  
Advertisement-Solicitation,  
'Message ID' = (M1: any valid value),  
-- 'Originating Virtual Address' absent  
'Destination Virtual Address' = (D4's VMAC)  
-- 'Destination Options' absent  
-- 'Data Options' absent
2. CHECK(that the Advertisement-Solicitation is silently dropped)

##### 12.YY.2.2.2 Connect-Request Wait Time Test

Purpose: To verify that the hub will close the WebSocket if the Connect-Request is not received before the connection wait timer expires.

Test Concept: With the IUT connected to the BACnet/SC network. Open a WebSocket connection with the IUT's hub port, but do not send a connect-request. Verify that the IUT closes the WebSocket after the connection wait timer expires.

Configuration Requirements: The IUT is configured to be a BACnet/SC hub.

Test Steps:

1. MAKE(a WebSocket connection to the IUT's hub function)
2. WAIT the connection wait timer expiration time
3. CHECK(that the IUT closed the WebSocket and did not send any messages on the WebSocket)

### 12.YY.2.2.3 VMAC Collision Detection Test

Purpose: To verify that connections to devices with a duplicate VMAC are rejected.

Test Concept: With the IUT operating as the primary hub, connect device D3 to the IUT's hub function. Connect D4 to the IUT's hub function and provide the IUT's VMAC in the Connect-Request. Verify that the IUT NAKs the connect request with an error code of NODE\_DUPLICATE\_VMAC. Retry the Connect-Request with D3's VMAC. Verify that the IUT NAK's the connect request.

Configuration Requirements: The IUT is configured as a hub. D3 is connected to the IUT and D4 is not.

Test Steps:

1. MAKE(D4 connect a WebSocket to the IUT's hub function)
2. TRANSMIT PORT (D4-IUT hub WebSocket),
  - Connect-Request,
  - 'Message ID' = (M1: any valid value),
  - 'Originating Virtual Address' absent
  - 'Destination Virtual Address' absent
  - 'Destination Options' absent
  - 'Data Options' absent
  - 'VMAC Address' = (IUT's VMAC),
  - 'Device UUID' = (D4's UUID),
  - 'Maximum BVLC Length' = (the D4's maximum BVLC accepted length),
  - 'Maximum NPDU Length' = (the D4's maximum NPDU accepted length)
3. RECEIVE PORT (D4-IUT hub WebSocket),
  - BVLC-Result,
  - 'Message ID' = M1,
  - 'Originating Virtual Address' absent
  - 'Destination Virtual Address' absent
  - 'Destination Options' = (absent or a list of valid options),
  - 'Data Options' absent
  - 'Result for BVLC Function' = X'06', -- Connect-Request
  - 'Result Code' = X'01', -- NAK
  - 'Error Header Marker' = X'00', -- not a header option problem
  - 'Error Class' = COMMUNICATION,
  - 'Error Code' = NODE\_DUPLICATE\_VMAC
4. TRANSMIT PORT (D4-IUT hub WebSocket),
  - Connect-Request,
  - 'Message ID' = (M2: any valid value),
  - 'Originating Virtual Address' absent
  - 'Destination Virtual Address' absent
  - 'Destination Options' absent
  - 'Data Options' absent
  - 'VMAC Address' = (D3's VMAC),
  - 'Device UUID' = (D4's UUID),
  - 'Maximum BVLC Length' = (the D4's maximum BVLC accepted length),
  - 'Maximum NPDU Length' = (the D4's maximum NPDU accepted length)
5. RECEIVE PORT (D4-IUT hub WebSocket),
  - BVLC-Result,
  - 'Message ID' = M2,
  - 'Originating Virtual Address' absent

```
-- 'Destination Virtual Address' absent
'Destination Options' =          (absent or a list of valid options),
-- 'Data Options' absent
'Result for BVLC Function' =      X'06',  -- Connect-Request
'Result Code' =                  X'01',  -- NAK
'Error Header Marker' =          X'00',  -- not a header option problem
'Error Class' =                  COMMUNICATION,
'Error Code' =                  NODE_DUPLICATE_VMAC
```

6. TRANSMIT PORT (D4-IUT hub WebSocket),
 

```
Connect-Request,
'Message ID' =                  (M3: any valid value),
-- 'Originating Virtual Address' absent
-- 'Destination Virtual Address' absent
-- 'Destination Options' absent
-- 'Data Options' absent
'VMAC Address' =                (D4's VMAC),
'Device UUID' =                 (D4's UUID),
'Maximum BVLC Length' =         (the D4's maximum BVLC accepted length),
'Maximum NPDU Length' =         (the D4's maximum NPDU accepted length)
```
7. RECEIVE PORT (D4-IUT hub WebSocket),
 

```
Connect-Accept,
'Message ID' =                  M3,
-- 'Originating Virtual Address' absent
-- 'Destination Virtual Address' absent
'Destination Options' =          (absent or a list of valid options),
-- 'Data Options' absent
'VMAC Address' =                (IUT's VMAC),
'Device UUID' =                 (IUT's UUID),
'Maximum BVLC Length' =         (the IUT's maximum BVLC accepted length),
'Maximum NPDU Length' =         (the IUT's maximum NPDU accepted length)
```

#### 12.YY.2.2.4 Rejection of Invalid Certificate Incoming Connection Test

Purpose: To verify that the IUT will not accept incoming connections if the peer's certificate is invalid.

Test Concept: With the IUT operating as a hub, D3 attempts to open a WebSocket to the IUT's hub URI. D3 presents an invalid certificate during the connection process. Verify that the WebSocket is not established.

Configuration Requirements: The IUT is configured to be a hub. The TD is configured with an invalid certificate, or a certificate signed by a Certificate Authority which is not one recognized by the IUT. The TD shall be configured to accept the IUT's certificate.

Test Steps:

1. MAKE(D3 attempt to connect to the IUT's hub function with an invalid certificate)
2. CHECK(that the IUT refused the connection)

#### 12.YY.3 Direct Connect Tests

This group of tests verifies secure connect devices using direct connections. The logical configuration of the network used for these tests is shown in Figure X6. The test descriptions in this clause assume that the TD plays the role of all of the other

devices in the network configuration. For the tests in this section, unless specified through a PORT parameter, messages specified by the test are on the IUT to peer device direct connection WebSocket.

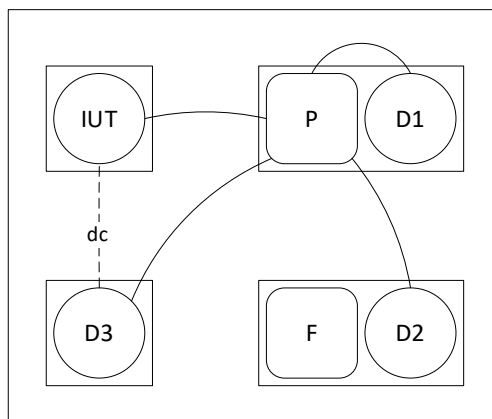


Figure X6: Network setup for basic node tests showing connections when the primary hub is active.

### 12.YY.3.1 Direction Connect Basic Tests

#### 12.YY.3.1.1 Direction Connect Basic Positive Tests

##### 12.YY.3.1.1.1 Unicast Through Direct Connect Test

Purpose: Verify that the IUT correctly composes unicasts aimed at a device through a direct connect.

Test Concept: With the IUT connected to D3 via a direct connection, have D3 send a request to the IUT which requires a response and verify that the response is sent back through the direct connection. If the IUT initiates APDU requests, make the IUT initiate a request and verify that it is transmitted through the direct connection.

Configuration Requirements: The IUT is connected to D3 via direct connect.

Test Steps:

1. TRANSMIT PORT (IUT-D3 direct connect WebSocket),  
Encapsulated-NPDU,  
'Message ID' = M1,  
-- 'Originating Virtual Address' absent  
-- 'Destination Virtual Address' absent  
-- 'Destination Options' absent  
'Data Options' = ({ X'41' }), -- Secure Path  
'Payload'  
ReadProperty-Request,  
'Object Identifier' = (O: any object in the IUT),  
'Property Identifier' = (P: any readable property in O)
2. RECEIVE PORT (IUT-D3 direct connect WebSocket),  
Encapsulated-NPDU,  
'Message ID' = (M2: any valid value),  
-- 'Originating Virtual Address' absent  
-- 'Destination Virtual Address' absent  
-- 'Destination Options' absent  
'Data Options' = ({ X'41' }), -- Secure Path  
'Payload'  
ReadProperty-ACK,  
'Object Identifier' = (O: any object in the IUT),  
'Property Identifier' = (P: any readable property in O),  
'Property Value' = (V: any valid value)

### 12.YY.3.1.1.2 Direct Connect Disconnect Test

Purpose: To verify that the IUT is able to correctly disconnect a direct connection with a non-hub peer BACnet/SC node.

Test Concept: With the IUT connected to another device via a direct connection, make the IUT close the connection. Verify that the IUT correctly requests and performs a disconnect.

Configuration Requirements: The IUT is directly connected to D3. If the IUT never closes established direct connections, this test shall be skipped.

Test Steps:

1. MAKE(the IUT drop the direct connection to D3)
2. RECEIVE PORT (IUT-D3 direct connect WebSocket)
  - Disconnect-Request,
  - 'Message ID' = (M1: any valid value),
  - 'Originating Virtual Address' absent
  - 'Destination Virtual Address' absent
  - 'Destination Options' = (absent or a list of valid options),
  - 'Data Options' absent
3. TRANSMIT PORT (IUT-D3 direct connect WebSocket)
  - Disconnect-ACK,
  - 'Message ID' = M1,
  - 'Originating Virtual Address' absent
  - 'Destination Virtual Address' absent
  - 'Destination Options' absent
  - 'Data Options' absent
4. CHECK(that the WebSocket is closed)

### 12.YY.3.1.1.3 Direct Connect Establishment Failover Test

Purpose: To verify that the IUT is able to continue to communicate with an arbitrary peer BACnet/SC node via the hub after a direct connect request to the peer is rejected.

Test Concept: With IUT connected to the network, and with a direct connection between the IUT and D3, make the IUT communicate with D3 to verify that the connection is working. Make D3 drop the direct connection. Make the IUT communicate with D3, having the IUT initiate the conversation if possible. Verify that the IUT is able to communicate with D3 through the hub.

Configuration Requirements: The IUT is connected to the primary hub and has a direct connection to D3. D3 shall be configured such that it will reject any attempts to re-establish the direct connection after the initial direct connection is dropped.

Test Steps:

1. TRANSMIT PORT (IUT-D3 direct connect WebSocket),
  - Advertisement,
  - 'Message ID' = M1,
  - 'Originating Virtual Address' absent
  - 'Destination Virtual Address' absent
  - 'Destination Options' absent
  - 'Data Options' absent
  - 'Payload'
  - 'Hub Connection Status' = X'01', -- connected to the primary
  - 'Accept Direct Connections' = X'00', -- does not accept incoming direct connections.
  - 'Maximum BVLC Length' = (any valid value),
  - 'Maximum NPDU Length' = (any valid value)
2. TRANSMIT PORT (IUT-D3 direct connect WebSocket),
  - Disconnect-Request,
  - 'Message ID' = (M2: any valid value),
  - 'Originating Virtual Address' absent
  - 'Destination Virtual Address' absent
  - 'Destination Options' absent
  - 'Data Options' absent
3. RECEIVE PORT IUT to D3 direct connect WebSocket),

```

Disconnect-ACK,
'Message ID' = M2,
-- 'Originating Virtual Address' absent
-- 'Destination Virtual Address' absent
-- 'Destination Options' absent
-- 'Data Options' absent
4. IF the IUT can initiate an NPDU request to D3 THEN {
    MAKE(IUT send an NPDU to D3)
    -- note that the IUT might attempt a reconnect at this point, which will be rejected by D3, and that such an
    -- attempt is acceptable
    RECEIVE PORT (IUT-TD hub WebSocket),
        Encapsulated-NPDU,
        'Message ID' = (M3: any valid value),
        -- 'Originating Virtual Address' absent
        'Destination Virtual Address' = D3,
        'Destination Options' = (absent or a valid list of options),
        'Data Options' = (Secure Path plus 0 or more valid data options),
        'Payload' = (a valid NPDU)
    } ELSE {
        TRANSMIT PORT (IUT-TD hub WebSocket),
            Encapsulated-NPDU,
            'Message ID' = (M4: any valid value),
            'Originating Virtual Address' = D3,
            -- 'Destination Virtual Address' absent
            -- 'Destination Options' absent
            'Data Options' = ({ X'41' } ), -- Secure Path
            'Payload' =
                ReadProperty-Request,
                'Object Identifier' = (O: any object in the IUT),
                'Property Identifier' = (P: any property in O)
        RECEIVE PORT (IUT-TD hub WebSocket),
            Encapsulated-NPDU,
            'Message ID' = (M5: any valid value),
            -- 'Originating Virtual Address' absent
            'Destination Virtual Address' = D3,
            'Destination Options' = (absent or a valid list of options),
            'Data Options' = (Secure Path plus 0 or more valid data options),
            'Payload' =
                ReadProperty-ACK,
                'Object Identifier' = (O: any object in the IUT),
                'Property Identifier' = (P: any property in O),
                'Property Value' = (any valid value)
    }

```

### 12.YY.3.1.2 Direction Connect Basic Negative Tests

#### 12.YY.3.1.2.1 Discard Broadcast BVLC Received on Direct Connect Test

Purpose: To verify that broadcast BVLCs received on a direct connection are discarded.

Test Concept: With the IUT connected to another device, D3, via a direct connection, have D3 send a broadcast to the IUT. Verify that the IUT does not process the broadcast.

Configuration Requirements: The IUT is connected to the SC network and has a direct connection to D3.

Test Steps:

1. TRANSMIT PORT (IUT-D3 direct connect WebSocket)
 

```

Encapsulated-NPDU,
'Message ID' = (M1: any valid value),
-- 'Originating Virtual Address' absent
'Destination Virtual Address' = X'FFFFFFFF', -- the local broadcast VMAC

```

'Destination Options' = (absent or a valid list of options),  
 'Data Options' = (Secure Path plus 0 or more valid data options),  
 'Payload' =  
 Who-Is-Request

2. CHECK(that the IUT does not generate a BVLC-Response nor an I-Am-Request)

## 12.YY.3.2 Accepting Direct Connect Tests

### 12.YY.3.2.1 Accepting Direct Connect Positive Tests

#### 12.YY.3.2.1.1 Direct Connect Acceptance Test

Purpose: To verify that the IUT correctly accepts direct connections.

Test Concept: With IUT connected to the network, have peer node D3 establish a direct connection with the IUT. Verify that the IUT correctly accepts the connection.

Configuration Requirements: The IUT is configured with a single WebSocket-URI at which it accepts direct connections. The IUT is connected to the primary hub.

Test Steps:

1. TRANSMIT PORT (IUT-TD hub WebSocket)  
 Address-Resolution,  
 'Message ID' = (M1: any valid value),  
 'Originating Virtual Address' = D3,  
 -- 'Destination Virtual Address' absent  
 -- 'Destination Options' absent  
 -- 'Data Options' absent
2. RECEIVE PORT (IUT-TD hub WebSocket)  
 Address-Resolution-ACK,  
 'Message ID' = M1,  
 -- 'Originating Virtual Address' absent  
 'Destination Virtual Address' = D3,  
 'Destination Options' = (absent or a valid list of options),  
 -- 'Data Options' absent  
 'Payload'  
 'WebSocket-URIs' (W: the configured WebSocket URI for the IUT)
- }
3. MAKE(D3 connect a WebSocket to the IUT)
4. TRANSMIT PORT (D3-IUT direct connect WebSocket),  
 Connect-Request,  
 'Message ID' = (M2: any valid value),  
 -- 'Originating Virtual Address' absent  
 -- 'Destination Virtual Address' absent  
 -- 'Destination Options' absent  
 -- 'Data Options' absent  
 'VMAC Address' = (D3's VMAC),  
 'Device UUID' = (D3's UUID),  
 'Maximum BVLC Length' = (the D1's maximum BVLC accepted length),  
 'Maximum NPDU Length' = (the D1's maximum NPDU accepted length)
6. RECEIVE PORT (D3-IUT direct connect WebSocket),  
 Connect-Accept,  
 'Message ID' = M2,  
 -- 'Originating Virtual Address' absent  
 -- 'Destination Virtual Address' absent  
 'Destination Options' (absent or any valid value),  
 -- 'Data Options' absent  
 'VMAC Address' = (IUT's VMAC),  
 'Device UUID' = (IUT's UUID),  
 'Maximum BVLC Length' = (the IUT's maximum BVLC accepted length),  
 'Maximum NPDU Length' = (the IUT's maximum NPDU accepted length)



### 12.YY.3.2.1.2 No Additional Certificate Checks Performed Test On Incoming Connections

Purpose: Verify that the IUT does not apply checks beyond the 4 listed in YY.7.4 when not configured to do so.

Test Concept: With the IUT configured to accept direct connections. The TD attempts to connect to the IUT with a valid certificate with field that would be caught if the IUT applied validation steps outside of those listed in YY.7.4.

Configuration Requirements: The IUT is configured to accept direct connections. The TD is configured with a certificate with a Common Name, Distinguished Name, or Subject Alternate Name which does not match the TD device.

Test Steps:

1. MAKE(the TD open a WebSocket to the IUT's direct connect URI)
2. TRANSMIT PORT (TD-IUT direct connect WebSocket)
  - Connect-Request,
  - 'Message ID' = (M1: any valid value),
  - 'Originating Virtual Address' absent
  - 'Destination Virtual Address' absent
  - 'Destination Options' absent
  - 'Data Options' absent
  - 'VMAC Address' = (TD's VMAC),
  - 'Device UUID' = (TD's UUID),
  - 'Maximum BVLC Length' = (the TD's maximum BVLC accepted length),
  - 'Maximum NPDU Length' = (the TD's maximum NPDU accepted length)
3. RECEIVE PORT (TD-IUT direct connect WebSocket)
  - Connect-Accept,
  - 'Message ID' = M1,
  - 'Originating Virtual Address' absent
  - 'Destination Virtual Address' absent
  - 'VMAC Address' = (IUT's VMAC),
  - 'Device UUID' = (IUT's UUID),
  - 'Maximum BVLC Length' = (the IUT's maximum BVLC accepted length),
  - 'Maximum NPDU Length' = (the IUT's maximum NPDU accepted length)

### 12.YY.3.2.2 Accepting Direct Connect Negative Tests

#### 12.YY.3.2.2.1 Connect-Request Wait Time Test

Purpose: To verify that the IUT will close the WebSocket if the Connect-Request is not received before the connection wait timer expires.

Test Concept: With the IUT connected to the BACnet/SC network, open a WebSocket connection to the IUT's direct connect URI, but do not send a connect-request. Verify that the IUT closes the WebSocket after the connection wait timer expires.

Configuration Requirements: The IUT is configured to accept direct connections.

Test Steps:

1. MAKE(a WebSocket connection to the IUT's direct connect WebSocket-URI)
2. WAIT the connection wait timer expiration time
3. CHECK(that the IUT closed the WebSocket and did not send any messages on the WebSocket)

#### 12.YY.3.2.2.2 Direct-Connect Duplicate Connection for IUT Accepted Connections Test

Purpose: To verify that duplicate direct connect connection requests result in the original connection being dropped for connections initiated by the IUT's peer.

Test Concept: With the IUT connected to the BACnet/SC network, D3 connects to the IUT's direct connect URI. When the connection is complete, D3 attempts to bring up a second connection to the IUT's direct connect URI. Verify that the IUT accepts the second connect request and closes the first connection. Repeat the reconnection, but with a new VMAC for D3 and ensure that the new request is accepted and the existing one dropped.

Configuration Requirements: The IUT is configured to accept direct connections.

# Test Steps:

1. MAKE(D3 connect to the IUT's direct connect WebSocket URI)
2. TRANSMIT PORT (D3-IUT direct connect first WebSocket),
  - Connect-Request,
  - 'Message ID' = (M1: any valid value),
  - 'Originating Virtual Address' absent
  - 'Destination Virtual Address' absent
  - 'Destination Options' absent
  - 'Data Options' absent
  - 'VMAC Address' = (D3's VMAC),
  - 'Device UUID' = (D3's UUID),
  - 'Maximum BVLC Length' = (the D3's maximum BVLC accepted length),
  - 'Maximum NPDU Length' = (the D3's maximum NPDU accepted length)
3. RECEIVE PORT (D3-IUT direct connect first WebSocket),
  - Connect-Accept,
  - 'Message ID' = M1,
  - 'Originating Virtual Address' absent
  - 'Destination Virtual Address' absent
  - 'VMAC Address' = (IUT's VMAC),
  - 'Device UUID' = (IUT's UUID),
  - 'Maximum BVLC Length' = (the IUT's maximum BVLC accepted length),
  - 'Maximum NPDU Length' = (the IUT's maximum NPDU accepted length)
4. MAKE(D3 connect a second WebSocket to the IUT's direct connect WebSocket-URI)
5. TRANSMIT PORT (D3-IUT direct connect second WebSocket),
  - Connect-Request,
  - 'Message ID' = (M2: any valid value),
  - 'Originating Virtual Address' absent
  - 'Destination Virtual Address' absent
  - 'Destination Options' absent
  - 'Data Options' absent
  - 'VMAC Address' = (D3's VMAC),
  - 'Device UUID' = (D3's UUID),
  - 'Maximum BVLC Length' = (the D3's maximum BVLC accepted length),
  - 'Maximum NPDU Length' = (the D3's maximum NPDU accepted length)
6. RECEIVE PORT (D3-IUT direct connect second WebSocket),
  - Connect-Accept,
  - 'Message ID' = M2,
  - 'Originating Virtual Address' absent
  - 'Destination Virtual Address' absent
  - 'Destination Options' = (absent or a list of valid options),
  - 'Data Options' absent
  - 'VMAC Address' = (IUT's VMAC),
  - 'Device UUID' = (IUT's UUID),
  - 'Maximum BVLC Length' = (the IUT's maximum BVLC accepted length),
  - 'Maximum NPDU Length' = (the IUT's maximum NPDU accepted length)
7. RECEIVE PORT (D3-IUT direct connect first WebSocket),
  - Disconnect-Request,
  - 'Message ID' = M3,
  - 'Originating Virtual Address' absent
  - 'Destination Virtual Address' absent
  - 'Destination Options' = (absent or a list of valid options),
  - 'Data Options' absent
8. TRANSMIT PORT (D3-IUT direct connect first WebSocket),
  - Disconnect-ACK,
  - 'Message ID' = M3,
  - 'Originating Virtual Address' absent
  - 'Destination Virtual Address' absent
  - 'Destination Options' absent
  - 'Data Options' absent
9. CHECK(that the IUT closed D3's initial WebSocket)

10. MAKE(D3 connect a third WebSocket to the IUT's direct connect WebSocket URI)
11. TRANSMIT PORT (D3-IUT direct connect third WebSocket),
  - Connect-Request,
  - 'Message ID' = (M4: any valid value),
  - 'Originating Virtual Address' absent
  - 'Destination Virtual Address' absent
  - 'Destination Options' absent
  - 'Data Options' absent
  - 'VMAC Address' = (a new VMAC for D3 which does not conflict with any other VMACs),
  - 'Device UUID' = (D3's UUID),
  - 'Maximum BVLC Length' = (the D3's maximum BVLC accepted length),
  - 'Maximum NPDU Length' = (the D3's maximum NPDU accepted length)
12. RECEIVE PORT (D3-IUT direct connect third WebSocket),
  - Connect-Accept,
  - 'Message ID' = M4,
  - 'Originating Virtual Address' absent
  - 'Destination Virtual Address' absent
  - 'Destination Options' = (absent or a list of valid options),
  - 'Data Options' absent
  - 'VMAC Address' = (IUT's VMAC),
  - 'Device UUID' = (IUT's UUID),
  - 'Maximum BVLC Length' = (the IUT's maximum BVLC accepted length),
  - 'Maximum NPDU Length' = (the IUT's maximum NPDU accepted length)
13. RECEIVE PORT (D3-IUT direct connect second WebSocket),
  - Disconnect-Request,
  - 'Message ID' = M5,
  - 'Originating Virtual Address' absent
  - 'Destination Virtual Address' absent
  - 'Destination Options' = (absent or a list of valid options),
  - 'Data Options' absent
14. TRANSMIT PORT (D3-IUT direct connect second WebSocket),
  - Disconnect-ACK,
  - 'Message ID' = M5,
  - 'Originating Virtual Address' absent
  - 'Destination Virtual Address' absent
  - 'Destination Options' absent
  - 'Data Options' absent
15. CHECK(that the IUT closed D3's second direct connect WebSocket)

### 12.YY.3.2.2.3 Direct-Connect Duplicate Connection for IUT Initiated Connections Test

Purpose: To verify that duplicate direct connect connection requests result in the original connection being dropped when the original connection was initiated by the IUT.

Test Concept: With the IUT connected to the BACnet/SC network, the IUT connects to the D3's direct connect URI. When the connection is complete, D3 attempts to bring up a second connection to the IUT's direct connect URI. Verify that the IUT accepts the connect request and closes the existing connection. D3 attempts to bring up a second connection to the IUT's direct connect URI. Verify that the IUT accepts the connect request and closes the existing connection.

Configuration Requirements: The IUT is configured to initiate and accept direct connections. If the IUT does not support both initiating and accepting direct connections, this test shall be skipped. If the IUT does not support 2 or more simultaneous direct connect WebSocket connections, this test shall be skipped.

Test Steps:

1. MAKE(IUT connect to D3's WebSocket URI)
2. RECEIVE PORT (IUT-D3 direct connect first WebSocket),
  - Connect-Request,
  - 'Message ID' = (M1: any valid value),
  - 'Originating Virtual Address' absent
  - 'Destination Virtual Address' absent
  - 'Destination Options' absent

- 'Data Options' absent
- 'VMAC Address' = (IUT's VMAC),
- 'Device UUID' = (IUT's UUID),
- 'Maximum BVLC Length' = (the IUT's maximum BVLC accepted length),
- 'Maximum NPDU Length' = (the IUT's maximum NPDU accepted length)
- 3. TRANSMIT PORT (IUT-D3 direct connect first WebSocket),
  - Connect-Accept,
  - 'Message ID' = M1,
  - 'Originating Virtual Address' absent
  - 'Destination Virtual Address' absent
  - 'VMAC Address' = (D3's VMAC),
  - 'Device UUID' = (D3's UUID),
  - 'Maximum BVLC Length' = (the D3's maximum BVLC accepted length),
  - 'Maximum NPDU Length' = (the D3's maximum NPDU accepted length)
- 4. MAKE(D3 connect a second WebSocket to the IUT's direct connect WebSocket-URI)
- 5. TRANSMIT PORT (D3-IUT direct connect second WebSocket),
  - Connect-Request,
  - 'Message ID' = (M2: any valid value),
  - 'Originating Virtual Address' absent
  - 'Destination Virtual Address' absent
  - 'Destination Options' absent
  - 'Data Options' absent
  - 'VMAC Address' = (D3's VMAC),
  - 'Device UUID' = (D3's UUID),
  - 'Maximum BVLC Length' = (the D3's maximum BVLC accepted length),
  - 'Maximum NPDU Length' = (the D3's maximum NPDU accepted length)
- 6. RECEIVE PORT (D3-IUT direct connect second WebSocket),
  - Connect-Accept,
  - 'Message ID' = M2,
  - 'Originating Virtual Address' absent
  - 'Destination Virtual Address' absent
  - 'Destination Options' = (absent or a list of valid options),
  - 'Data Options' absent
  - 'VMAC Address' = (IUT's VMAC),
  - 'Device UUID' = (IUT's UUID),
  - 'Maximum BVLC Length' = (the IUT's maximum BVLC accepted length),
  - 'Maximum NPDU Length' = (the IUT's maximum NPDU accepted length)
- 7. RECEIVE PORT (D3-IUT direct connect first WebSocket),
  - Disconnect-Request,
  - 'Message ID' = M3,
  - 'Originating Virtual Address' absent
  - 'Destination Virtual Address' absent
  - 'Destination Options' = (absent or a list of valid options),
  - 'Data Options' absent
- 8. TRANSMIT PORT (D3-IUT direct connect first WebSocket),
  - Disconnect-ACK,
  - 'Message ID' = M3,
  - 'Originating Virtual Address' absent
  - 'Destination Virtual Address' absent
  - 'Destination Options' absent
  - 'Data Options' absent
- 9. CHECK(that the IUT closed D3's initial WebSocket)
- 10. MAKE(D3 connect a third WebSocket to the IUT's direct connect WebSocket URI)
- 11. TRANSMIT PORT (D3-IUT direct connect third WebSocket),
  - Connect-Request,
  - 'Message ID' = (M4: any valid value),
  - 'Originating Virtual Address' absent
  - 'Destination Virtual Address' absent
  - 'Destination Options' absent
  - 'Data Options' absent

- 'VMAC Address' = (a new VMAC for D3 which does not conflict with any other VMACs),  
 'Device UUID' = (D3's UUID),  
 'Maximum BVLC Length' = (the D3's maximum BVLC accepted length),  
 'Maximum NPDU Length' = (the D3's maximum NPDU accepted length)
12. RECEIVE PORT (D3-IUT direct connect third WebSocket),  
 Connect-Accept,  
 'Message ID' = M4,  
 -- 'Originating Virtual Address' absent  
 -- 'Destination Virtual Address' absent  
 'Destination Options' = (absent or a list of valid options),  
 -- 'Data Options' absent  
 'VMAC Address' = (IUT's VMAC),  
 'Device UUID' = (IUT's UUID),  
 'Maximum BVLC Length' = (the IUT's maximum BVLC accepted length),  
 'Maximum NPDU Length' = (the IUT's maximum NPDU accepted length)
13. RECEIVE PORT (D3-IUT direct connect second WebSocket),  
 Disconnect-Request,  
 'Message ID' = M5,  
 -- 'Originating Virtual Address' absent  
 -- 'Destination Virtual Address' absent  
 'Destination Options' = (absent or a list of valid options),  
 -- 'Data Options' absent
14. TRANSMIT PORT (D3-IUT direct connect second WebSocket),  
 Disconnect-ACK,  
 'Message ID' = M5,  
 -- 'Originating Virtual Address' absent  
 -- 'Destination Virtual Address' absent  
 -- 'Destination Options' absent  
 -- 'Data Options' absent
15. CHECK(that the IUT closed D3's second direct connect WebSocket)

#### 12.YY.3.2.2.4 VMAC Collision Detection Test

Purpose: To verify that connections to devices with a duplicate VMAC are rejected.

Test Concept: With the IUT connected to the network, have D3 open a WebSocket to the IUT's direct connect URI and provide the IUT's VMAC in the connect request. Verify that the IUT NAKs the connect request with an error code of NODE\_DUPLICATE\_VMAC.

Test Steps:

1. MAKE(D3 connect a WebSocket to the IUT's direct connect WebSocket URI)
2. TRANSMIT PORT (D3-IUT direct connect WebSocket),  
 Connect-Request,  
 'Message ID' = (M1: any valid value),  
 -- 'Originating Virtual Address' absent  
 -- 'Destination Virtual Address' absent  
 -- 'Destination Options' absent  
 -- 'Data Options' absent  
 'VMAC Address' = (IUT's VMAC),  
 'Device UUID' = (D3's UUID),  
 'Maximum BVLC Length' = (the D3's maximum BVLC accepted length),  
 'Maximum NPDU Length' = (the D3's maximum NPDU accepted length)
3. RECEIVE PORT (D3-IUT direct connect WebSocket),  
 BVLC-Result,  
 'Message ID' = M1,  
 -- 'Originating Virtual Address' absent  
 -- 'Destination Virtual Address' absent  
 'Destination Options' = (absent or a list of valid options),  
 -- 'Data Options' absent  
 'Result for BVLC Function' = X'06', -- Connect-Request  
 'Result Code' = X'01', -- NAK

- 'Error Header Marker' = X'00', -- not a header option problem  
 'Error Class' = COMMUNICATION,  
 'Error Code' = NODE\_DUPLICATE\_VMAC
4. TRANSMIT PORT (D3-IUT direct connect WebSocket),  
 Connect-Request,  
 'Message ID' = (M2: any valid value),  
 -- 'Originating Virtual Address' absent  
 -- 'Destination Virtual Address' absent  
 -- 'Destination Options' absent  
 -- 'Data Options' absent  
 'VMAC Address' = (D3's VMAC),  
 'Device UUID' = (D3's UUID),  
 'Maximum BVLC Length' = (the D3's maximum BVLC accepted length),  
 'Maximum NPDU Length' = (the D3's maximum NPDU accepted length)
5. RECEIVE PORT (D3-IUT direct connect WebSocket),  
 Connect-Accept,  
 'Message ID' = M2,  
 -- 'Originating Virtual Address' absent  
 -- 'Destination Virtual Address' absent  
 'Destination Options' = (absent or a list of valid options),  
 -- 'Data Options' absent  
 'VMAC Address' = (IUT's VMAC),  
 'Device UUID' = (IUT's UUID),  
 'Maximum BVLC Length' = (the IUT's maximum BVLC accepted length),  
 'Maximum NPDU Length' = (the IUT's maximum NPDU accepted length)

## 12.YY.3.2.2.5 Rejection of Invalid Certificate Incoming Connection Test

Purpose: To verify that the IUT will not accept incoming connections if the peer's certificate is invalid.

Test Concept: With the IUT connected to the network, D3 attempts to open a WebSocket to the IUT's direct connect URI. D3 presents an invalid certificate during the connection process. Verify that the WebSocket is not established.

Configuration Requirements: The IUT is configured to accept direct connections. D3 is configured with an invalid certificate. D3 is configured to accept the IUT's certificate.

Test Steps:

1. MAKE(D3 attempt to connect to the IUT's direct connect URI with an invalid certificate)
2. CHECK(that the IUT refused the connection)

## 12.YY.3.3 Initiating Direct Connect Tests

### 12.YY.3.3.1 Initiating Direct Connect Positive Tests

#### 12.YY.3.3.1.1 Direct Connect Establishment Test

Purpose: To verify that the IUT is able to correctly establish a direct connection with a non-hub peer BACnet/SC node.

Test Concept: With IUT connected to the network, make the IUT establish a direct connection to another node on the network. Verify that the direct connection is correctly established.

Configuration Requirements: The IUT is configured to support establishing direct connections. The IUT is connected to the primary hub. D3 is configured to support accepting direct connections. The IUT is configured to use dynamic discovery of WebSocket-URIs if supported, otherwise the WebSocket-URI for D3 is configured into the IUT.

Test Steps:

1. MAKE(the IUT establish a direct connection to D3)
2. IF (the IUT supports discovering direct connection URIs) {  
 RECEIVE PORT (IUT-TD hub WebSocket)  
 Address-Resolution,  
 'Message ID' = (M1: any valid value),  
 -- 'Originating Virtual Address' absent  
 'Destination Virtual Address' = D3,  
 'Destination Options' = (absent or a list of valid options),

```

-- 'Data Options' absent
TRANSMIT PORT (IUT-TD hub WebSocket)
  Address-Resolution-ACK,
  'Message ID' = M1,
  'Originating Virtual Address' = D3
  -- 'Destination Virtual Address' absent
  -- 'Destination Options' absent
  -- 'Data Options' absent
  'Payload'
  'WebSocket-URIs' (W: a WebSocket URI which D3 can be reached at)
}
4. CHECK(that the IUT opens a WebSocket to D3's WebSocket-URI)
5. RECEIVE PORT (IUT-D3 direct connect WebSocket),
  Connect-Request,
  'Message ID' = (M2: any valid value),
  -- 'Originating Virtual Address' absent
  -- 'Destination Virtual Address' absent
  'Destination Options' (absent or any valid value),
  -- 'Data Options' absent
  'VMAC Address' = (IUT's VMAC),
  'Device UUID' = (IUT's UUID),
  'Maximum BVLC Length' = (the IUT's maximum BVLC accepted length),
  'Maximum NPDU Length' = (the IUT's maximum NPDU accepted length)
6. TRANSMIT Connect-Accept,
  'Message ID' = M2,
  -- 'Originating Virtual Address' absent
  -- 'Destination Virtual Address' absent
  'Destination Options' (absent or any valid value),
  -- 'Data Options' absent
  'VMAC Address' = (D3's VMAC),
  'Device UUID' = (D3's UUID),
  'Maximum BVLC Length' = (the D3's maximum BVLC accepted length),
  'Maximum NPDU Length' = (the D3's maximum NPDU accepted length)

```

### 12.YY.3.3.1.2 Direct Connect Multiple URI Test

Purpose: To verify that the IUT is able to correctly connect to a peer with multiple URIs where only 1 of the URIs is valid for IUT.

Test Concept: With IUT connected to the network, make the IUT establish a direct connection to D3 with D3 configured to advertise multiple URIs of which only 1 is valid for the IUT. Verify that the direct connection is correctly established.

Configuration Requirements: The IUT is configured to support establishing direct connections. The IUT is connected to the primary hub. D3 is configured to support accepting direct connections and is configured with multiple WebSocket-URIs. Only one of the URIs configured into D3 shall be reachable by the IUT, and that URI shall not be the first or last URI in the D3's list of URIs. The IUT is configured to use dynamic discovery of WebSocket-URIs. If the IUT does not support dynamic discovery of WebSocket-URIs, this test shall be skipped.

Test Steps:

```

1. MAKE(the IUT attempt a direct connection to D3)
2. RECEIVE PORT (IUT-TD hub WebSocket)
  Address-Resolution,
  'Message ID' = (M1: any valid value),
  -- 'Originating Virtual Address' absent
  'Destination Virtual Address' = D3,
  'Destination Options' = (absent or a list of valid options),
  -- 'Data Options' absent
TRANSMIT PORT (IUT-TD hub WebSocket)
  Address-Resolution-ACK,
  'Message ID' = M1,
  'Originating Virtual Address' = D3

```

- 'Destination Virtual Address' absent
- 'Destination Options' absent
- 'Data Options' absent
- 'Payload'
- 'WebSocket-URIs' (W1, ..., Wi, ..., Wn) -- only Wi is a WebSocket-URI that  
-- will connect to D3
- }
- 4. CHECK(that the IUT repeatedly attempts to connect to the WebSocket-URIs until Wi is used)
- 5. RECEIVE PORT (IUT-D3 direct connect WebSocket),  
Connect-Request,  
'Message ID' = (M2: any valid value),  
-- 'Originating Virtual Address' absent  
-- 'Destination Virtual Address' absent  
'Destination Options' (absent or any valid value),  
-- 'Data Options' absent  
'VMAC Address' = (IUT's VMAC),  
'Device UUID' = (IUT's UUID),  
'Maximum BVLC Length' = (the IUT's maximum BVLC accepted length),  
'Maximum NPDU Length' = (the IUT's maximum NPDU accepted length)
- 6. TRANSMIT PORT (IUT-D3 direct connect WebSocket),  
Connect-Accept,  
'Message ID' = M2,  
-- 'Originating Virtual Address' absent  
-- 'Destination Virtual Address' absent  
'Destination Options' (absent or any valid value),  
-- 'Data Options' absent  
'VMAC Address' = (D3's VMAC),  
'Device UUID' = (D3's UUID),  
'Maximum BVLC Length' = (the D3's maximum BVLC accepted length),  
'Maximum NPDU Length' = (the D3's maximum NPDU accepted length)

### 12.YY.3.3.2 Initiating Direct Connect Negatives Tests

#### 12.YY.3.3.2.1 Invalid Web Socket Scheme In Configured Direct Connect URI Test

Purpose: To verify that the IUT does not attempt to connect to configured direct connect URIs with invalid schemes.

Test Concept: The IUT is configured with the direct connect URI for device D3 and is connected to the BACnet/SC network. Take the steps that would normally cause the IUT to initiate a direct connect to D3. Verify that the IUT does not attempt to connect to the invalid URI.

Configuration Requirements: The IUT is configured to support establishing direct connections to D3 and D3's URI is configured into the IUT so it does not have to dynamically discover it. The configured URI shall have an invalid websocket scheme. The IUT starts the test already connected to the primary hub. D3 is configured to support accepting direct connections. If the IUT does not support configured peer URIs or does not accept invalid schemes in configured URIs, this test shall be skipped.

Test Steps:

1. MAKE(the IUT initiate the establishment of a direct connection to D3)
2. CHECK(that the IUT does not attempt to open a WebSocket to the invalid WebSocket-URI)

#### 12.YY.3.3.2.2 Invalid Web Socket Scheme in Discovered Direct Connect URI Test

Purpose: To verify that the IUT does not attempt to connect to discovered URIs containing invalid websocket schemes.

Test Concept: With the IUT connected to the BACnet/SC network, make the IUT initiate a direct connect to D3. D3 advertises its direct connect URL with an invalid scheme. Verify that the IUT does not attempt to connect to the invalid URI.

Configuration Requirements: The IUT is configured to support establishing direct connections and is connected to the primary hub. D3 is configured to support accepting direct connections. If the IUT does not support dynamically determining direct connect URIs, this test shall be skipped.



#### Test Steps:

1. MAKE(the IUT initiate the establishment of a direct connection to D3)
2. IF (the IUT supports discovering direct connection URIs) {
  - RECEIVE PORT (IUT-TD hub WebSocket)
    - Address-Resolution,
    - 'Message ID' = (M1: any valid value),
    - 'Originating Virtual Address' absent
    - 'Destination Virtual Address' = D3,
    - 'Destination Options' = (absent or a list of valid options),
    - 'Data Options' absent
  - TRANSMIT PORT (IUT-TD hub WebSocket)
    - Address-Resolution-ACK,
    - 'Message ID' = M1,
    - 'Originating Virtual Address' = D3,
    - 'Destination Virtual Address' absent
    - 'Destination Options' absent
    - 'Data Options' absent
    - 'Payload'
    - 'WebSocket-URIs' (W: a WebSocket URI with an invalid scheme)
3. CHECK(that the IUT does not attempt to open a WebSocket to the invalid WebSocket-URI)

#### 12.YY.3.3.2.3 Rejection of Invalid Certificate Outgoing Connection Test

Purpose: To verify that the IUT will drop initiated connection attempts if the peer's certificate is invalid.

Test Concept: With the IUT configured to initiate direct connections. Make the IUT attempt to connect to D3 via a direct connection. D3 presents an invalid certificate during the connection. Verify that the WebSocket is not established.

Configuration Requirements: The IUT is configured to initiate direct connections. D3 is configured with an invalid certificate. D3 shall be configured to accept the IUT's certificate.

#### Test Steps:

1. MAKE(the IUT attempt to establish a direct connection to D3)
2. CHECK(that the IUT initiated a WebSocket connection)
3. CHECK(that the WebSocket connection was failed by the IUT)

## 10 NETWORK LAYER PROTOCOL TESTS

[Add into **Clause 10.2**, tests **10.2.X3**, **10.2.X4**, **10.2.X5**, **10.2.X6**, p. 483]

### 10.2 Router Functionality Tests

#### 10.2.X3 Data Attributes Forwarding Test

Purpose: To verify that routers which connect multiple network supporting data attributes, correctly route data attributes.

Test Concept: With the IUT configured as a router between two networks which supports data attributes (such as BACnet/SC), send to the router a message which needs to be routed to the next network, which contains data attributes. Verify that the message is correctly routed and the data attributes are included in the routed message.

Configuration Requirements: The IUT shall be configured as a router between 2 networks which support data attributes. If the IUT does not support this configuration this test shall be skipped.

Test Steps:

1. TRANSMIT PORT A,  
DA = LOCAL BROADCAST,  
SOURCE = D1A,  
'Data Options' = (DATA\_OPTS: a valid list of options),  
DNET = GLOBAL BROADCAST,  
DLEN = 0,  
Hop Count = 255,  
BACnet-Unconfirmed-Request-PDU,  
'Service Choice' = Who-Is
2. RECEIVE PORT B,  
DA = LOCAL BROADCAST,  
SA = IUT,  
'Data Options' = DATA\_OPTS,  
DNET = GLOBAL BROADCAST,  
DLEN = 0,  
SNET = 1,  
SADR = D1A,  
Hop Count = (any integer x:  $0 < x < 255$ ),  
BACnet-Unconfirmed-Request-PDU,  
'Service Choice' = Who-Is

#### 10.2.X4 Data Attributes Dropping Test

Purpose: To verify that the IUT drops data\_attributes from a received message before routing it to a network which does not support data\_attributes.

Test Concept: With the IUT configured as a router from a network which support data\_attributes (such as BACnet/SC) to a network which does not support data\_attributes (such as BACnet/IP), send to the router a message which needs to be routed to the next network, and which contains data\_attributes. Verify that message is correctly routed and the data\_attributes are silently dropped.

Test Steps:

1. TRANSMIT PORT A,  
DA = LOCAL BROADCAST,  
SOURCE = D1A,  
'Data Options' = (DATA\_OPTS: a valid list of options),  
DNET = GLOBAL BROADCAST,  
DLEN = 0,  
Hop Count = 255,  
BACnet-Unconfirmed-Request-PDU,  
'Service Choice' = Who-Is
2. RECEIVE PORT B,  
DA = LOCAL BROADCAST,

SA = IUT,  
DNET = GLOBAL BROADCAST,  
DLEN = 0,  
SNET = 1,  
SADR = D1A,  
Hop Count = (any integer x:  $0 < x < 255$ ),  
BACnet-Unconfirmed-Request-PDU,  
'Service Choice' = Who-Is

### 10.2.X5 Secure Path Test

**Purpose:** To verify that a BACnet/SC to BACnet/SC router correctly conveys the 'Secure Path' header option on routed messages.

**Test Concept:** With the IUT configured as a router from BACnet/SC to BACnet/SC or another datalink which supports the 'Secure Path' header option, send to the router a message which needs to be routed to the next network, and which contains the Secure Path header option. Verify that Secure Path option is included in the routed message.

**Configuration Requirements:** The IUT is connected to 2 networks which support data attributes and are secure. If the IUT does not support this configuration then this test shall be skipped.

**Test Steps:**

1. TRANSMIT PORT A,  
DA = LOCAL BROADCAST,  
SOURCE = D1A,  
'Data Options' = (DATA\_OPTS: Secure Path),  
DNET = GLOBAL BROADCAST,  
DLEN = 0,  
Hop Count = 255,  
BACnet-Unconfirmed-Request-PDU,  
'Service Choice' = Who-Is
2. RECEIVE PORT B,  
DA = LOCAL BROADCAST,  
SA = IUT,  
'Data Options' = (DATA\_OPTS: Secure Path),  
DNET = GLOBAL BROADCAST,  
DLEN = 0,  
SNET = 1,  
SADR = D1A,  
Hop Count = (any integer x:  $0 < x < 255$ ),  
BACnet-Unconfirmed-Request-PDU,  
'Service Choice' = Who-Is

### 10.2.X6 Insecure Path Test

**Purpose:** To verify that a non-BACnet/SC to BACnet/SC router does not include the 'Secure Path' header option on routed messages from the non-BACnet/SC network.

**Test Concept:** With the IUT configured as a router from BACnet/SC to a non-BACnet/SC which does not have data options, send to the router a message on the non-BACnet/SC network which needs to be routed to the BACnet/SC network. Verify that the Secure Path option is not included in the routed message.

**Configuration Requirements:** The IUT is connected to 2 networks with PORT A connected to a network type which does not support secure path, and PORT B connected to a network type which does. If the IUT does not support this configuration then this test shall be skipped.

**Test Steps:**

1. TRANSMIT PORT A,  
DA = LOCAL BROADCAST,  
SOURCE = D1A,

DNET = GLOBAL BROADCAST,  
DLEN = 0,  
Hop Count = 255,  
BACnet-Unconfirmed-Request-PDU,  
'Service Choice' = Who-Is

2. RECEIVE PORT B,  
DA = LOCAL BROADCAST,  
SA = IUT,  
-- 'Data Options' absent  
DNET = GLOBAL BROADCAST,  
DLEN = 0,  
SNET = 1,  
SADR = D1A,  
Hop Count = (any integer x:  $0 < x < 255$ ),  
BACnet-Unconfirmed-Request-PDU,  
'Service Choice' = Who-Is

## 135.1-2019s-4 Add tests for IPv6

### Rationale

Addendum 135-2012aj introduced support for IPv6. Addendum 135.1-2019s adds new tests to meet the requirements of addendum aj.

[Add new **Clause 12.X**, p. 625]

### 12.X BACnet/IPv6 Functionality Tests

This clause defines the tests necessary to demonstrate BACnet/IPv6 functionality, as defined in Annex U of the BACnet standard. For each test case, a sequence of one or more messages that are to be exchanged are described. A passing result occurs when the IUT and TD exchange messages as described in the test case. Any other combinations of messages constitute a failure of the test. Some test cases are not valid unless some other test defined in this standard has already been executed and the IUT passed this test. These dependencies are noted in the test case description.

For the tests in this clause, references to the virtual address mean the 3-octet virtual address. For example, Source-Virtual-Address = TD means Source-Virtual-Address = (the 3-octet VMAC of TD).

#### 12.X.1 Common Tests

This group of tests verifies that a B/IPv6 device will respond correctly to incoming B/IPv6 messages. All B/IPv6 devices shall execute these tests.

Configuration Requirements: The IUT's Network Port object that represents the B/IPv6 port under test shall be configured as follows:

- BACnet\_IPv6\_Multicast\_Address is FF02::BAC0 (Link Local Multicast Address)

#### 12.X.1.1 Execute Original-Unicast-NPDU

Purpose: To verify that an IUT will process an Original-Unicast-NPDU message.

Test Steps:

1. TRANSMIT DA = IUT, SA = TD,  
Original-Unicast-NPDU,  
Source-Virtual-Address = TD,  
Destination-Virtual-Address = IUT,  
ReadProperty-Request,  
'Object Identifier' = X,  
'Property Identifier' = Y
2. RECEIVE DA = TD, SA = IUT  
Original-Unicast-NPDU,  
Source-Virtual-Address = IUT,  
Destination-Virtual-Address = TD,  
ReadProperty-ACK,  
'Object Identifier' = X,  
'Property Identifier' = Y

#### 12.X.1.2 Execute Virtual-Address-Resolution

Purpose: To verify that an IUT will process a Virtual-Address-Resolution message.

Test Steps:

1. TRANSMIT DA = IUT, SA = TD,  
Virtual-Address-Resolution, Source-Virtual-Address = TD
2. RECEIVE DA = TD,  
Virtual-Address-Resolution-ACK,

Source-Virtual-Address = IUT,  
Destination-Virtual-Address = TD

## 12.X.2 IPv6 Normal Mode Tests

This group of tests verifies that a B/IPv6 device that is operating in normal mode (not a BACnet Broadcast Management Device (BBMD), and not a Foreign Device) will respond correctly to incoming B/IPv6 messages.

Configuration Requirements: The IUT's Network Port object that represents the B/IPv6 port under test shall be configured as follows:

- BACnet\_IPv6\_Mode is NORMAL
- BACnet\_IPv6\_Multicast\_Address is FF02::BAC0 (Link Local Multicast Address)

### 12.X.2.1 Positive Tests

#### 12.X.2.1.1 Initiate Original-Broadcast-NPDU

Purpose: To verify that an IUT, operating in normal IPv6 mode, will correctly initiate an Original-Broadcast-NPDU message.

Test Steps:

1. MAKE(the IUT send a broadcast)
2. RECEIVE DA=Link Local Multicast Address, SA = IUT  
Original-Broadcast-NPDU,  
Source-Virtual-Address = IUT,  
(any valid BACnet-Unconfirmed-Request-PDU, with any valid broadcast network options)

#### 12.X.2.1.2 Execute Original-Broadcast-NPDU

Purpose: To verify that an IUT, operating in normal IPv6 mode, will process an Original-Broadcast-NPDU message.

Test Steps:

1. TRANSMIT DA = B/IPv6 Link Local Multicast Address, SA = TD,  
Original-Broadcast-NPDU,  
Source-Virtual-Address = TD,  
Who-Is-Request
2. IF (the IUT responds with Unicast I-Am) THEN  
RECEIVE DA = TD, SA = IUT,  
Original-Unicast-NPDU,  
Source-Virtual-Address = IUT,  
Destination-Virtual-Address = TD,  
I-Am-Request  
ELSE  
RECEIVE DA=Link Local Multicast Address, SA = IUT  
Original-Broadcast-NPDU,  
Source-Virtual-Address = IUT,  
I-Am-Request
3. CHECK (The IUT does not issue any Forwarded-NPDUs)

#### 12.X.2.1.3 Execute Forwarded-NPDU

Purpose: To verify that an IUT, operating in normal IPv6 mode, will process a Forwarded-NPDU.

Test Steps:

1. TRANSMIT DA = Link Local Multicast Address, SA = TD,  
Forwarded-NPDU,  
Original-Source-Virtual-Address = D2,  
Original-Source-B/IPv6-Address = D2,

#### Who-Is-Request

2. IF (the IUT responds with Unicast I-Am) THEN
  - RECEIVE DA = D2, SA = IUT,
  - Original-Unicast-NPDU,
  - Source-Virtual-Address = IUT,
  - Destination-Virtual-Address = D2,
  - I-Am-Request
- ELSE
  - RECEIVE DA=Link Local Multicast Address, SA = IUT
  - Original-Broadcast-NPDU,
  - Source-Virtual-Address = IUT,
  - I-Am-Request
3. CHECK (The IUT does not issue any Forwarded-NPDU BVLCs)

#### 12.X.2.1.4 Execute Address-Resolution

Purpose: To verify that an IUT, operating in normal IPv6 mode, will process an Address-Resolution message.

Test Steps:

1. TRANSMIT DA = B/IPv6 Link Local Multicast Address, SA = TD,
  - Address-Resolution,
  - Source-Virtual-Address = TD,
  - Target-Virtual-Address = IUT
2. RECEIVE DA = TD,
  - Address-Resolution-ACK,
  - Source-Virtual-Address = IUT,
  - Destination-Virtual-Address = TD
3. CHECK (The IUT does not issue any Forwarded-Address-Resolution BVLCs)

#### 12.X.2.1.5 Execute Forwarded-Address-Resolution

Purpose: To verify that an IUT, operating in normal IPv6 mode, will process a Forwarded-Address-Resolution message.

Test Concept: The TD, acting as a BBMD, sends a Forwarded-Address-Resolution message to the IUT on behalf of device D2. It is verified that the IUT responds to D2 with an Address-Resolution message.

1. TRANSMIT DA = IUT, SA = TD,
  - Forwarded-Address-Resolution,
  - Original-Source-Virtual-Address = D2,
  - Target-Virtual-Address = IUT
  - Original-Source-B/IPv6-Address = D2
2. RECEIVE DA = D2, SA = IUT
  - Address-Resolution-ACK,
  - Source-Virtual-Address = IUT,
  - Destination-Virtual-Address = D2
3. CHECK (The IUT does not issue any Forwarded-Address-Resolution BVLCs)

#### 12.X.2.2 Negative Tests

##### 12.X.2.2.1 Reject Register-Foreign-Device

Purpose: To verify that an IUT, operating in normal IPv6 mode, will reject a Register-Foreign-Device request.

Test Steps:

1. TRANSMIT DESTINATION = IUT, SA = TD,
  - Register-Foreign-Device,
  - Source-Virtual-Address = TD
  - Time-To-Live = 60

2. RECEIVE DESTINATION = TD,  
BVLC-Result,  
Source-Virtual-Address = IUT  
'Result Code' = Register-Foreign-Device NAK

#### **12.X.2.2.2 Reject Delete-Foreign-Device-Table-Entry**

Purpose: To verify that an IUT, operating in normal IPv6 mode, will reject a Delete-Foreign-Device-Table-Entry request.

Test Steps:

1. TRANSMIT DESTINATION = IUT, SA = TD,  
Delete-Foreign-Device-Table-Entry,  
Source-Virtual-Address = TD  
FDT Entry = TD
2. RECEIVE DESTINATION = TD,  
BVLC-Result,  
Source-Virtual-Address = IUT  
'Result Code' = Delete-Foreign-Device-Table-Entry NAK

#### **12.X.2.2.3 Reject Distribute-Broadcast-To-Network**

Purpose: To verify that an IUT, operating in normal IPv6 mode, will reject a Distribute-Broadcast-To-Network request.

Test Steps:

1. TRANSMIT DESTINATION = IUT, SA = TD,  
Distribute-Broadcast-To-Network,  
Original-Source-Virtual-Address = TD  
Who-Is-Request
2. RECEIVE DESTINATION = TD,  
BVLC-Result,  
Source-Virtual-Address = IUT  
'Result Code' = Distribute-Broadcast-To-Network NAK

### **12.X.3 Foreign Device Tests**

This group of tests verifies that a B/IPv6 device that is configured as a Foreign Device is able to register with a BBMD and send and receive broadcast messages through the BBMD.

Configuration Requirements: The IUT's Network Port object that represents the B/IPv6 port under test shall be configured as follows:

- BACnet\_IPv6\_Mode is FOREIGN
- BACnet\_IPv6\_Multicast\_Address is FF02::BAC0 (Link Local Multicast Address)

#### **12.X.3.1 Positive Tests**

##### **12.X.3.1.1 Initiate Distribute-Broadcast-To-Network-NPDU**

Purpose: To verify that an IUT, configured as a Foreign Device, will correctly initiate a Distribute-Broadcast-To-Network - NPDU message.

Configuration Requirements: The TD is operating as a BBMD, and the IUT has registered as a foreign device with it.

Test Steps:

1. MAKE(the IUT send a broadcast)
2. RECEIVE DA=IUT, SA = IUT  
Distribute-Broadcast-To-Network-NPDU,



Source-Virtual-Address = IUT,  
(any valid BACnet-Unconfirmed-Request-PDU, with any valid broadcast network options)

#### 12.X.3.1.2 Execute Forwarded-NPDU

Purpose: To verify that an IUT, operating as a foreign device, will process a Forwarded-NPDU.

Configuration Requirements: The TD is operating as a BBMD, and the IUT has registered as a foreign device with it.

Test Steps:

1. TRANSMIT DA = IUT, SA = TD,  
Forwarded-NPDU,  
Original-Source-Virtual-Address = D2,  
Original-Source-B/IPv6-Address = D2,  
Who-Is-Request
2. IF (the IUT responds with Unicast I-Am) THEN  
RECEIVE DA = D2, SA = IUT,  
Original-Unicast-NPDU,  
Source-Virtual-Address = IUT,  
Destination-Virtual-Address = D2,  
I-Am-Request  
ELSE  
RECEIVE DA=TD, SA = IUT  
Distribute-Broadcast-To-Network-NPDU,  
Source-Virtual-Address = IUT,  
I-Am-Request
3. CHECK (The IUT does not issue any Forwarded-NPDU BVLCs)

#### 12.X.3.1.3 Execute Forwarded-Address-Resolution

Purpose: To verify that an IUT, operating as a foreign device, will process a Forwarded-Address-Resolution message.

Test Concept: The TD, acting as a BBMD, sends a Forwarded-Address-Resolution message to the IUT on behalf of device D2. It is verified that the IUT responds to D2 with an Address-Resolution message.

Test Steps:

1. TRANSMIT DA = IUT, SA = TD,  
Forwarded-Address-Resolution,  
Original-Source-Virtual-Address = D2,  
Target-Virtual-Address = IUT  
Original-Source-B/IPv6-Address = D2
2. RECEIVE  
DA = D2, SA = IUT  
Address-Resolution-ACK,  
Source-Virtual-Address = IUT,  
Destination-Virtual-Address = D2
3. CHECK (The IUT does not issue any Forwarded-Address-Resolution BVLCs)

#### 12.X.3.2 Negative Tests

##### 12.X.3.2.1 Ignores Original-Broadcast-NPDU

Purpose: To verify that an IUT, operating as a foreign device, will not process an Original-Broadcast-NPDU message.

Test Steps:

1. TRANSMIT DA = B/IPv6 Link Local Multicast Address, SA = D2,  
Original-Broadcast-NPDU,  
Source-Virtual-Address = D2,  
Who-Is-Request

2. CHECK (The IUT does not issue any I-Am-Requests in response)

#### **12.X.3.2.2 Ignore Address-Resolution**

Purpose: To verify that an IUT, operating as a foreign device, will ignore multicast Address-Resolution messages.

Test Steps:

1. TRANSMIT DA = B/IPv6 Link Local Multicast Address, SA = D2,  
Address-Resolution,  
Source-Virtual-Address = D2,  
Target-Virtual-Address = IUT
2. CHECK (The IUT does not issue any Address-Resolution-ACK BVLCs)

#### **12.X.3.2.3 Reject Register-Foreign-Device**

Purpose: To verify that an IUT, operating as a foreign device, will reject a Register-Foreign-Device request.

Test Steps:

1. TRANSMIT DESTINATION = IUT, SA = TD,  
Register-Foreign-Device,  
Source-Virtual-Address = TD  
Time-To-Live = 60
2. RECEIVE DESTINATION = TD,  
BVLC-Result,  
Source-Virtual-Address = IUT  
'Result Code' = Register-Foreign-Device NAK

#### **12.X.3.2.4 Reject Delete-Foreign-Device-Table-Entry**

Purpose: To verify that an IUT, operating as a foreign device, will reject a Delete-Foreign-Device-Table-Entry request.

Test Steps:

1. TRANSMIT DESTINATION = IUT, SA = TD,  
Delete-Foreign-Device-Table-Entry,  
Source-Virtual-Address = TD  
FDT Entry = TD
2. RECEIVE DESTINATION = TD,  
BVLC-Result,  
Source-Virtual-Address = IUT  
'Result Code' = Delete-Foreign-Device-Table-Entry NAK

#### **12.X.3.2.5 Reject Distribute-Broadcast-To-Network**

Purpose: To verify that an IUT, operating as a foreign device, will reject a Distribute-Broadcast-To-Network request.

Test Steps:

1. TRANSMIT DESTINATION = IUT, SA = TD,  
Distribute-Broadcast-To-Network,  
Original-Source-Virtual-Address = TD  
Who-Is-Request
2. RECEIVE DESTINATION = TD,  
BVLC-Result,  
Source-Virtual-Address = IUT  
'Result Code' = Distribute-Broadcast-To-Network NAK

## 12.X.4 BBMD Tests

### 12.X.4.1 Positive Tests

This group of tests verifies that a B/IPv6 device that is configured as a BACnet Broadcast Management Device (BBMD) will correctly process incoming B/IPv6 messages that pertain to BBMDs. Only devices that are configured to support BBMD functionality shall execute these tests.

Configuration Requirements: The IUT's Network Port object that represents the B/IPv6 port under test shall be configured as follows:

- BACnet\_IPv6\_Mode is BBMD
- BACnet\_IPv6\_Multicast\_Address is FF02::BAC0 (Link Local Multicast Address)
- BBMD\_Broadcast\_Distribution\_Table shall contain:

bbmd-address
BBMD1
BBMD2
BBMD3

For purposes of these tests, TD shall be operating as BBMD1.

#### 12.X.4.1.1 Original-Broadcast-NPDU

Purpose: To verify that the IUT, configured as a BBMD, will forward an Original-Broadcast-NPDU request.

Test Steps:

##### 1. TRANSMIT

DA = B/IPv6 Link Local Multicast Address,  
SA = TD,  
Source-Virtual-Address = TD,  
Original-Broadcast-NPDU,  
Who-Is-Request

##### 2. RECEIVE

DA = BBMD1,  
SA = IUT,  
Forwarded-NPDU,  
Original-Source-Virtual-Address = TD  
Original-Source-B/IPv6-Address = TD  
Who-Is-Request

##### 3. RECEIVE

DA = BBMD2,  
SA = IUT,  
Forwarded-NPDU,  
Original-Source-Virtual-Address = TD  
Original-Source-B/IPv6-Address = TD  
Who-Is-Request

##### 4. RECEIVE

DA = BBMD3,  
SA = IUT,  
Forwarded-NPDU,  
Original-Source-Virtual-Address = TD  
Original-Source-B/IPv6-Address = TD  
Who-Is-Request

#### 12.X.4.1.2 Forwarded-NPDU

Purpose: To verify that the IUT, configured as a BBMD, will forward a Forwarded-NPDU request.

Configuration Requirements: Register FD1 as a foreign device with the IUT. FD3 is a registered foreign device with BBMD1.

Notes to Tester: The order of the messages transmitted by the IUT is not significant.

Test Steps:

1. TRANSMIT

DA = IUT,  
SA = BBMD1,  
Forwarded-NPDU,  
Source-Virtual-Address = FD3,  
Original-Source-B/IPv6-Address = FD3  
I-Am-Request

2. RECEIVE

DA = B/IPv6 Link Local Multicast Address,  
SA = IUT  
Forwarded-NPDU,  
Source-Virtual-Address = FD3,  
Original-Source-B/IPv6-Address = FD3  
I-Am-Request

3. RECEIVE

DA = FD1,  
SA = IUT  
Forwarded-NPDU,  
Source-Virtual-Address = FD3,  
Original-Source-B/IPv6-Address = FD3  
I-Am-Request

### 12.X.4.1.3 Address-Resolution

Purpose: To verify that the IUT, configured as a BBMD, will process an Address-Resolution request when the target virtual address is not the virtual address of the IUT.

Configuration Requirements: TD shall be a registered foreign device (FD1) with the IUT.

Notes to Tester: The execution of step 7 is not significant but is shown here in order to demonstrate the completion of the BVLC.

Test Steps:

1. TRANSMIT

DA = IUT,  
SA = TD,  
Address-Resolution,  
Source-Virtual-Address = TD,  
Target-Virtual-Address = FD2

2. RECEIVE

DA = B/IPv6 Link Local Multicast Address  
SA = IUT,  
Forwarded-Address-Resolution,  
Original-Source-Virtual-Address = TD,  
Target-Virtual-Address = FD2,  
Original-Source-B/IPv6-Address = TD

3. RECEIVE

DA = BBMD1,  
SA = IUT,  
Forwarded-Address-Resolution,  
Original-Source-Virtual-Address = TD,  
Target-Virtual-Address = FD2,  
Original-Source-B/IPv6-Address = TD

4. RECEIVE

- DA = BBMD2,
- SA = IUT,
- Forwarded-Address-Resolution,
- Original-Source-Virtual-Address = TD,
- Target-Virtual-Address = FD2,
- Original-Source-B/IPv6-Address = TD
- 5. RECEIVE
  - DA = BBMD3,
  - SA = IUT,
  - Forwarded-Address-Resolution,
  - Original-Source-Virtual-Address = TD,
  - Target-Virtual-Address = FD2,
  - Original-Source-B/IPv6-Address = TD
- 6. RECEIVE
  - DA = FD2,
  - SA = IUT,
  - Forwarded-Address-Resolution,
  - Original-Source-Virtual-Address = TD,
  - Target-Virtual-Address = FD2,
  - Original-Source-B/IPv6-Address = TD
- 7. TRANSMIT
  - DA = TD,
  - SA = FD2,
  - Address-Resolution-ACK,
  - Source-Virtual-Address = FD2,
  - Destination-Virtual-Address = TD

#### 12.X.4.1.4 Forwarded-Address-Resolution

Purpose: To verify that the IUT, configured as a BBMD, will process a Forwarded-Address-Resolution request when the target virtual address is not the virtual address of the IUT.

Configuration Requirements: TD shall operate as BBMD1 and listed in the IUTs Broadcast Distribution Table.

Notes to Tester: The execution of step 7 is not significant but is shown here in order to demonstrate the completion of the BVLC. The order of the messages transmitted by the IUT is not significant.

Test Steps:

- 1. TRANSMIT
  - DA = IUT,
  - SA = TD,
  - Forwarded-Address-Resolution,
  - Original-Source-Virtual-Address = FD1,
  - Target-Virtual-Address = FD2
  - Original-Source-B/IPv6-Address = FD1
- 2. RECEIVE
  - DA = B/IPv6 Link Local Multicast Address,
  - SA = IUT,
  - Forwarded-Address-Resolution,
  - Original-Source-Virtual-Address = FD1,
  - Target-Virtual-Address = FD2,
  - Original-Source-B/IPv6-Address = FD1
- 3. RECEIVE
  - DA = FD2,
  - SA = IUT,
  - Forwarded-Address-Resolution,
  - Original-Source-Virtual-Address = FD1,
  - Target-Virtual-Address = FD2,

Original-Source-B/IPv6-Address = FD1

#### 5. TRANSMIT

DA = TD,  
SA = FD2,  
Address-Resolution-ACK,  
Source-Virtual-Address = FD2,  
Destination-Virtual-Address = TD

#### 12.X.4.1.5 Distribute-Broadcast-To-Network

Purpose: To verify that the IUT, configured as a BBMD, will process a Distribute-Broadcast-To-Network request.

Configuration Requirements: Register FD1 as a foreign device with the IUT. FD2 is a registered foreign device with BBMD1. For purposes of this test, TD is acting as FD1.

Notes to Tester: Steps 1-6 are the processing of the Distributed-Broadcast-To-Network, Step 7 and on is the processing of the APDU service by the IUT. The order of the messages transmitted by the IUT is not significant.

Test Steps:

#### 1. TRANSMIT

DA = IUT,  
SA = FD1,  
Distribute-Broadcast-To-Network,  
Who-Is-Request

#### 2. RECEIVE

DA = B/IPv6 Link Local Multicast Address,  
SA = IUT,  
Forwarded-NPDU,  
Source-Virtual-Address = FD1,  
Original-Source-Virtual-Address = FD1,  
Who-Is-Request

#### 3. RECEIVE

DA = BBMD1,  
SA = IUT,  
Forwarded-NPDU,  
Source-Virtual-Address = FD1,  
Original-Source-Virtual-Address = FD1,  
Who-Is-Request

#### 4. RECEIVE

DA = BBMD2,  
SA = IUT,  
Forwarded-NPDU,  
Source-Virtual-Address = FD1,  
Original-Source-Virtual-Address = FD1,  
Who-Is-Request

#### 5. RECEIVE

DA = BBMD3,  
SA = IUT,  
Forwarded-NPDU,  
Source-Virtual-Address = FD1,  
Original-Source-Virtual-Address = FD1,  
Who-Is-Request

#### 6. RECEIVE

DA = FD2,  
SA = IUT,  
Forwarded-NPDU,  
Source-Virtual-Address = FD1,  
Original-Source-Virtual-Address = FD1,  
Who-Is-Request

7. RECEIVE

DA = B/IPv6 Link Local Multicast Address,  
SA = IUT,  
Original-Broadcast-NPDU,  
Original-Source-Virtual-Address = IUT,  
I-Am-Request

8. RECEIVE

DA = BBMD1,  
SA = IUT,  
Forwarded-NPDU,  
Source-Virtual-Address = IUT,  
Original-Source-Virtual-Address = IUT,  
I-Am-Request

9. RECEIVE DA = BBMD2,

SA = IUT,  
Forwarded-NPDU,  
Source-Virtual-Address = IUT,  
Original-Source-Virtual-Address = IUT,  
I-Am-Request

10. RECEIVE DA = BBMD3,

SA = IUT,  
Forwarded-NPDU,  
Source-Virtual-Address = IUT,  
Original-Source-Virtual-Address = IUT,  
I-Am-Request

11. RECEIVE

DA = FD1,  
SA = IUT,  
Forwarded-NPDU,  
Source-Virtual-Address = IUT,  
Original-Source-Virtual-Address = IUT,  
I-Am-Request

12. RECEIVE

DA = FD2,  
SA = IUT  
Forwarded-NPDU,  
Source-Virtual-Address = IUT,  
Original-Source-Virtual-Address = IUT,  
I-Am-Request

## 12.X.4.2 Negative Tests

### 12.X.4.2.1 Reject Forwarded-NPDU

Purpose: To verify that the IUT, configured as a BBMD, will drop a Forwarded-NPDU request from a BBMD that's not in the IUT's BDT.

Configuration Requirements: Empty the IUT's BDT. FD3 is a foreign device registered with the IUT.

Test Steps:

1. TRANSMIT

DA = IUT,  
SA = BBMD1,  
Forwarded-NPDU,  
Source-Virtual-Address = FD3,  
Original-Source-B/IPv6-Address = FD3  
I-Am-Request

2. CHECK (The IUT does not issue any Forwarded-NPDU BVLCs)

#### 12.X.4.2.2 Reject Address-Resolution

Purpose: To verify that the IUT, configured as a BBMD, will not process an Address-Resolution request when the target virtual address is not the virtual address of the IUT and the SA is not from a device registered with the IUT.

Configuration Requirements: TD shall not be a registered foreign device (FD1) with the IUT.

Test Steps:

1. TRANSMIT  
DA = IUT,  
SA = TD,  
Address-Resolution,  
Source-Virtual-Address = TD,  
Target-Virtual-Address = FD2
2. RECEIVE  
DA = TD,  
SA = IUT  
BVLC-Result  
Address-Resolution NAK
2. CHECK (The IUT does not issue any Forwarded-Address-Resolution BVLCs)

#### 12.X.4.2.3 Reject Forwarded-Address-Resolution

Purpose: To verify that the IUT, configured as a BBMD, will not process a Forwarded-Address-Resolution request from a BBMD that is not present in the IUT's BDT.

Configuration Requirements: Empty the IUT's BDT.

Test Steps:

1. TRANSMIT  
DA = IUT,  
SA = TD,  
Forwarded-Address-Resolution,  
Original-Source-Virtual-Address = FD1,  
Target-Virtual-Address = FD2  
Original-Source-B/IPv6-Address = FD1
2. CHECK (The IUT does not issue any Forwarded-Address-Resolution BVLCs)

#### 12.X.4.2.4 Reject Distribute-Broadcast-To-Network

Purpose: To verify that the IUT, configured as a BBMD, will not process a Distribute-Broadcast-To-Network request from a device that is not registered as a foreign device with the IUT.

Configuration Requirements: Ensure the TD is not registered as a foreign device with the IUT and that the TD is not listed in the IUTs FDT.

Test Steps:

1. TRANSMIT  
DA = IUT,  
SA = TD,  
Distribute-Broadcast-To-Network,  
Who-Is-Request
2. RECEIVE  
DA = TD  
SA = IUT  
BVLC-Result  
Distribute-Broadcast-To-Network-NAK



### 12.X.4.3 Broadcast Distribution Table Operations

This group of tests verifies that a BACnet Broadcast Management Device will correctly perform BDT operations.

Configuration Requirements: The IUT's Network Port object that represents the B/IPv6 port under test shall be configured as follows:

- BACnet\_IPv6\_Mode is BBMD

#### 12.X.4.3.1 Verify writability of the BDT

Purpose: To verify the contents of the broadcast distribution table.

Test Steps:

##### 1. TRANSMIT

WriteProperty-Request,  
'Object Identifier' = (Network Port Object that represents this port),  
'Property Identifier' = BBMD\_Broadcast\_Distribution\_Table  
'Property Value' = (WrittenBDT: a list of valid BACnetBDTEntry)

##### 2. RECEIVE

BACnet-SimpleACK-PDU,

##### 3. ReadBDT = READ NP, BBMD\_Broadcast\_Distribution\_Table

##### 4. CHECK(ReadBDT contains the same entries as WrittenBDT, but not necessarily in the same order)

### 12.X.5 Foreign Device Management Tests

This group of tests verifies that a BBMD with an FDT will correctly perform FDT operations.

Configuration Requirements: The IUT's Network Port object, NP, that represents the B/IPv6 port under test shall be configured as follows:

- BACnet\_IPv6\_Mode is BBMD
- BACnet\_IPv6\_Multicast\_Address is FF02::BAC0 (Link Local Multicast Address)
- BBMD\_Accept\_FD\_Registrations is TRUE.

The TD's Network Port object that represents the B/IPv6 port being used shall be configured as follows:

- BACnet\_IPv6\_Mode is FOREIGN
- BACnet\_IPv6\_Multicast\_Address is FF02::BAC0 (Link Local Multicast Address)

#### 12.X.5.1 Execute Register-Foreign-Device

Purpose: To verify that the IUT will handle a Register-Foreign-Device request.

Test Steps:

##### 1. TRANSMIT

DA = IUT,  
SA = TD,  
Source-Virtual-Address = TD,  
Register-Foreign-Device,  
'Time-To-Live' = 60

##### 2. RECEIVE

DA = TD,  
SA = IUT,  
Source-Virtual-Address = IUT,  
BVLC-Result,  
'Result Code' = 0

##### 3. VERIFY NP, BBMD\_Foreign\_Device\_Table = ( (B/IPv6 address of FD2, 60, 90-execution time) )

### 12.X.5.2 Execute Delete-Foreign-Device-Table-Entry

Purpose: To verify that the IUT will handle a Delete-Foreign-Device-Table-Entry message when a valid FDT entry is supplied.

Configuration Requirements: The TD shall take the role of foreign device FD1. The IUT's FDT must be empty.

Test Steps:

1. TRANSMIT
  - DA = IUT,
  - SA = FD1,
  - Source-Virtual-Address = FD1,
  - Register-Foreign-Device,
  - 'Time-To-Live' = 60
2. RECEIVE
  - DA = FD1,
  - SA = IUT,
  - Source-Virtual-Address = IUT,
  - BVLC-Result,
  - 'Result Code' = 0
3. VERIFY NP, BBMD\_Foreign\_Device\_Table = ( (B/IPv6 address of FD1, 60, 90-execution time) )
  - 'Property Value' = ( (B/IPv6 address of FD1, 60, 90-execution time) )
4. TRANSMIT
  - DA = IUT,
  - SA = FD1,
  - Source-Virtual-Address = FD1,
  - Delete-Foreign-Device-Table-Entry,
  - 'FDT Entry' = FD1
5. RECEIVE
  - DA = FD1,
  - SA = IUT,
  - Source-Virtual-Address = IUT,
  - BVLC-Result,
  - 'Result Code' = Successful completion
6. VERIFY NP, BBMD\_Foreign\_Device\_Table = ()

### 12.X.5.3 Foreign Device Table Timer Operations

#### 12.X.5.3.1 Non-Zero-Duration Foreign Device Table Timer Operations

Purpose: To verify that the IUT will handle FDT timer operations: finite time Foreign Device registration, re-registration, adding grace period to the supplied Time-To-Live parameter, and FDT entry clearing upon timer expiration.

Configuration Requirements: The TD shall take the role of foreign device FD2. The value of the IUT's BBMD\_Foreign\_Device Table must be empty.

Test Steps:

1. TRANSMIT
  - DA = IUT,
  - SA = FD2,
  - Register-Foreign-Device,
  - 'Time-To-Live' = 60
2. RECEIVE
  - DA = FD2,
  - SA = IUT,
  - BVLC-Result,
  - 'Result Code' = 0
3. VERIFY NP, BBMD\_Foreign\_Device\_Table = ( (B/IPv6 address of FD2, 60, 90-execution time) )
4. TRANSMIT

- DA = IUT,
- SA = FD2,
- Register-Foreign-Device,
- 'Time-To-Live' = 40
- 5. RECEIVE
  - DA = FD2,
  - SA = IUT,
  - BVLC-Result,
  - 'Result Code' = 0
- 6. WAIT (30 seconds)
- 7. VERIFY NP, BBMD\_Foreign\_Device\_Table = ( (B/IPv6 address of FD2, 40, 40-execution time) )
- 8. WAIT (50 seconds)
- 9. VERIFY NP, BBMD\_Foreign\_Device\_Table = ( )

### 12.X.5.3.2 Zero-Duration Foreign Device Timer Operations

Purpose: To verify that the IUT will handle Foreign Device registration with Time-To-Live parameter equal to zero and clears FDT entry upon timer expiration.

Configuration Requirements: The TD shall take the role of foreign device FD2. The IUTs FDT must be empty.

Test Steps:

- 1. TRANSMIT
  - DA = IUT,
  - SA = FD2,
  - Register-Foreign-Device-Table,
  - 'Time-To-Live' = 0
- 2. RECEIVE
  - DA = FD2,
  - SA = IUT,
  - BVLC-Result,
  - 'Result Code' = 0
- 3. WAIT (10 seconds)
- 4. VERIFY NP, BBMD\_Foreign\_Device\_Table = ( (B/IPv6 address of FD2, 0, 20-execution time) )
- 5. WAIT (30 seconds)
- 6. VERIFY NP, BBMD\_Foreign\_Device\_Table = ( )

### 12.X.5.4 Delete-Foreign-Device-Table-Entry For A Non-existent Entry

Purpose: To verify that the IUT will handle a Delete-Foreign-Device-Table-Entry message when a non-existent FDT entry is supplied.

Test Concept: The IUT starts with a Foreign Device Table without an entry for FD1. The TD, acting as FD1, attempts to delete its entry from the IUT's Foreign Device Table. It is verified that the IUT returns a NAK to the request.

Configuration Requirements: The IUT's Foreign Device Table does not contain an entry for FD1.

Test Steps:

- 1. VERIFY NP, BBMD\_Foreign\_Device\_Table = (a list of entries without an entry for FD1)
- 2. TRANSMIT
  - DA = IUT,
  - SA = FD1,
  - Source-Virtual-Address = FD1,
  - Delete-Foreign-Device-Table-Entry,
  - 'FDT Entry' = FD1
- 3. RECEIVE
  - DA = FD1,

SA = IUT,  
Source-Virtual-Address = IUT  
BVLC-Result,  
'Result Code' = Delete-Foreign-Device-Table-Entry NAK

4. VERIFY NP, BBMD\_Foreign\_Device\_Table = ( the previously read list but with updated lifetimes )

## 135.1-2019s-5 Add new and correct existing BACnet/IP Functionality Tests

### Rationale

Errors have been identified in a number of BACnet/IP functionality tests in ANSI/ASHRAE Standard 135.1-2019. In addition, the test coverage is increased by the addition of new tests.

Also, all BACnet/IP functionality tests are moved to a subclause of DATA LINK LAYER PROTOCOL TESTS (Clause 12).

[Renumber existing **Clause 14** and **ALL** subclauses, p.676]

### 14.12.3. BACnet/IP FUNCTIONALITY TESTS

#### 14.1.12.3.1 Non-BBMD B/IP Device

[Reviewer Note: This makes changes to all tests in section 14.1 that use the I-Am service, to allow the unicast form of the response.]

[Change **Clause 14.1.7**, p. 625]

#### 14.1.712.3.1.7 Forwarded-NPDU (One-hop Distribution)

Purpose: To verify that an IUT, not configured as a BBMD, will process a Forwarded-NPDU message.

Configuration Requirements: The IUT shall not be configured as a BBMD. The TD shall be on a different IP subnet than that of the IUT.

Test Steps:

1. TRANSMIT DA = Directed IP Broadcast to IUT's IP Subnet, ~~SOURCE~~SA = TD,  
Forwarded-NPDU,  
Originating-Device = TD,  
NPDU = Who-Is
2. *IF (the IUT responds with an unicast I-Am) THEN*  
RECEIVE DA = TD,  
Original-Unicast-NPDU,  
NPDU = I-Am  
*ELSE*  
RECEIVE DA = Local IP Broadcast, ~~SOURCE = IUT,~~  
Original-Broadcast-NPDU,  
NPDU = I-Am
3. CHECK (The IUT shall not issue any Forwarded-NPDUs)

[Change **Clause 14.1.8**, p.625]

#### 14.1.812.3.1.8 Original-Broadcast-NPDU

Purpose: To verify that an IUT, not configured as a BBMD, will process an Original-Broadcast-NPDU message.

Test Steps:

1. TRANSMIT DA = Local IP Broadcast, ~~SOURCE~~SA = TD,  
Original-Broadcast-NPDU,  
NPDU = Who-Is
2. *IF (the IUT responds with Unicast I-Am) THEN*  
RECEIVE DA = TD,  
Original-Unicast-NPDU,  
NPDU = I-Am  
*ELSE*  
RECEIVE DA = Local IP Broadcast, ~~SOURCE = IUT,~~  
Original-Broadcast-NPDU,  
NPDU = I-Am
3. CHECK (The IUT shall not issue any Forwarded-NPDUs)

[Change **Clause 14.1.10**, p.625]

#### ~~14.1.10~~**12.3.1.10 Forwarded-NPDU (Two-hop Distribution)**

Purpose: To verify that an IUT, not configured as a BBMD, will process a Forwarded-NPDU message.

Configuration Requirements: The IUT should not be configured as a BBMD. The TD shall be on the same subnet as the IUT. D1 is a device on a different IP subnet than the TD.

Test Steps:

1. TRANSMIT DA = Local IP Broadcast, ~~SOURCE~~SA = TD,  
Forwarded-NPDU,  
Originating-Device = D1,  
NPDU = Who-Is
2. *IF (the IUT responds with Unicast I-Am) THEN*  
RECEIVE DA = D1,  
Original-Unicast-NPDU,  
NPDU = I-Am  
*ELSE*  
RECEIVE DA = Local IP Broadcast, ~~SOURCE = IUT,~~  
Original-Broadcast-NPDU,  
NPDU = I-Am
3. CHECK (The IUT shall not issue any Forwarded-NPDUs)

[Add new **Clause 12.3.1.X11**, p. 625]

#### **12.3.1.X11 Processing Forwarded-NPDU request initiated from different port**

Purpose: To verify that an IUT will correctly process a Forwarded-NPDU message received from a device located at an address where it has a different UDP port number from those in the source and destination of a Forwarded-NPDU.

Test Concept: The IUT and the TD (acting as a BBMD) are configured such that they have the same UDP port number (P1). The originating device (D2) is selected having different UDP port number (P2) than the IUT and TD. The behavior of the IUT is verified when it correctly responds to the Forwarded-NPDU message from the device having different UDP number.

Configuration Requirements: The IUT is on the same subnet as the TD and on the same port number (P1). D2 is a device on a different subnet and has an address using port P2.

Test Steps:

1. TRANSMIT Forwarded-NPDU,

```

    Originating-Device = D2 -- (with UDP port P2)
    NPDU = Who-Is
2. IF (the IUT responds with Unicast I-Am) THEN
    RECEIVE Original-Unicast-NPDU,
        DESTINATION = D2, -- (with UDP port P2)
        NPDU = I-Am
ELSE
    RECEIVE Original-Broadcast-NPDU -- (with UDP port P1.)
    NPDU = I-Am

```

[Add new **Clause 12.3.1.X12**, p.625]

### **12.3.1.X12 Processing Forwarded-NPDU request initiated from different port when registered as a Foreign Device into a BBMD**

Purpose: To verify that an IUT when configured as a Foreign Device, will correctly process a Forwarded-NPDU message received from a device using a different UDP port number from those in the source and destination of the Forwarded-NPDU.

Test Concept: The IUT and the TD, acting as the BBMD are configured such that they have the same UDP port number (P1). The IUT must be on a different IP subnet than the BBMD. The IUT is registered as a Foreign Device with the BBMD. The originating device (D2) is selected having different UDP port number (P2) than the IUT and BBMD. The behavior of the IUT is verified when it correctly responds to the Forwarded-NPDU message from the device having different UDP number.

Configuration Requirements: TD is acting as a BBMD with port P1. D2 is a device at an address using a different port P2.

Test Steps:

```

1. TRANSMIT Forwarded-NPDU,
    Originating-Device = D2 -- (with UDP port P2)
    NPDU = Who-Is
2. IF (the IUT responds with Unicast I-Am) THEN
    RECEIVE Original-Unicast-NPDU,
        DESTINATION = D2, -- (with UDP port P2)
        NPDU = I-Am
ELSE
    RECEIVE Distribute-Broadcast-to-Network
        DESTINATION = TD, -- (with UDP port P1)
        NPDU = I-Am

```

### **~~14.2~~12.3.2 BBMD B/IP Device with a Server Application**

[Reviewer Note: This makes changes to all tests in Section 14.2 that use the I-Am service, to allow the unicast form of the response.]

#### **~~14.2.1~~12.3.2.1 Execute Forwarded-NPDU**

[Change **Clause 14.2.1.1**, p.625]

##### **~~14.2.1.1~~12.3.2.1.1 Execute Forwarded-NPDU (One-hop Distribution)**

Configuration Requirements: The IUT shall be configured with a BDT that contains:

B/IP Address	Broadcast Distribution Mask
IUT	IP Subnet 1 subnet mask
BBMD1	IP Subnet 2 subnet mask

*Notes to Tester: The order of the messages transmitted by the IUT is not significant.*

Test Steps:

1. TRANSMIT
  - DA = Directed IP Broadcast to IP Subnet 1,
  - SA = BBMD1,
  - Forwarded-NPDU,
  - Originating-Device = BBMD1,
  - NPDU = Who-Is
2. IF (the IUT responds with Unicast I-Am) THEN
  - RECEIVE DESTINATION = BBMD1,
  - Original-Unicast-NPDU,
  - NPDU = I-Am
- ELSE
  - RECEIVE
    - DA = Local IP Broadcast on IP Subnet 1,
    - ~~SA = IUT,~~
    - Original-Broadcast-NPDU,
    - NPDU = I-Am
- ~~3.~~ RECEIVE
  - DA = Directed IP Broadcast to IP Subnet 2,
  - ~~SA = IUT~~
  - Forwarded-NPDU,
  - Originating-Device = IUT,
  - NPDU = I-Am
43. CHECK (The IUT does not forward or resend the Who-Is packet out the port on which it was received)

~~Notes to Tester: The order of the messages transmitted by the IUT is not significant.~~

[Change **Clause 14.2.1.2**, p. 625]

#### ~~14.2.1.2~~ 12.3.2.1.2 Execute Forwarded-NPDU (Two-hop Distribution)

Configuration Requirements: The IUT shall be configured with a BDT that contains:

B/IP Address	Broadcast Distribution Mask
IUT	255.255.255.255
BBMD1	255.255.255.255

~~When the IUT is configured for NAT, the Originating Device in Forwarded NPDU's that originate at the IUT, OD, is equal to the Global IP Address and Port of the IUT's Internet Router. When the IUT is not configured for NAT operation, OD is equal to the IUT.~~

*Notes to Tester: The order of the messages transmitted by the IUT is not significant.*

Test Steps:

1. TRANSMIT
  - ~~DA = IUT,~~
  - SOURCE = BBMD1,
  - Forwarded-NPDU,
  - Originating-Device = BBMD1,
  - NPDU = Who-Is
2. RECEIVE



DA = Local IP Broadcast on IP Subnet 1,  
~~SA = IUT,~~  
 Forwarded-NPDU,  
 Originating-Device = BBMD1,  
 NPDU = Who-Is

3. IF (the IUT responds with Unicast I-Am) THEN  
 RECEIVE DESTINATION = BBMD1,  
 Original-Unicast-NPDU,  
 NPDU = I-Am  
 ELSE  
 RECEIVE  
 DA = Local IP Broadcast on IP Subnet 1,  
~~SA = IUT,~~  
 Original-Broadcast-NPDU,  
 NPDU = I-Am

4. ~~RECEIVE~~  
 RECEIVE  
 DA = BBMD1,  
~~SA = IUT,~~  
 Forwarded-NPDU,  
 Originating-Device = ~~OD~~IUT,  
 NPDU = I-Am

Notes to Tester: The order of the messages transmitted by the IUT is not significant

#### ~~14.2.2.1~~12.3.2.2 Execute Original-Broadcast-NPDU

[Change Clause 14.2.2.1, p. 625]

##### ~~14.2.2.1~~12.3.2.2.1 Execute Original-Broadcast-NPDU (One-hop Distribution)

Configuration Requirements: The IUT shall be configured with a BDT that contains:

B/IP Address	Broadcast Distribution Mask
IUT	IP Subnet 1 subnet mask
BBMD1	IP Subnet 2 subnet mask

Notes to Tester: The order of the messages transmitted by the IUT is not significant.

Test Steps:

- TRANSMIT  
 DA = Local IP Broadcast,  
 SA = D1,  
 Original-Broadcast-NPDU,  
 NPDU = Who-Is
- RECEIVE  
 DA = Directed IP Broadcast to IP Subnet 2,  
~~SA = IUT~~  
 Forwarded-NPDU,  
 Originating-Device = D1,  
 NPDU = Who-Is
- RECEIVE  
 DA = Local IP Broadcast,  
~~SA = IUT,~~  
 Original-Broadcast-NPDU,  
 NPDU = I-Am
- IF (the IUT responds with Unicast I-Am) THEN  
 RECEIVE DESTINATION = D1,

*Original-Unicast-NPDU*,  
*NPDU = I-Am*  
**ELSE**  
**RECEIVE**  
 DA = Directed IP Broadcast to IP Subnet 2,  
~~SA = IUT~~  
 Forwarded-NPDU,  
 Originating-Device = IUT,  
 NPDU = I-Am

Notes to Tester: The order of the messages transmitted by the IUT is not significant.

[Change **Clause 14.2.2.2**, p. 625]

#### ~~14.2.2.2~~12.3.2.2.2 Execute Original-Broadcast-NPDU (Two-hop Distribution)

Configuration Requirements: The IUT shall be configured with a BDT that contains:

B/IP Address	Broadcast Distribution Mask
IUT	255.255.255.255
BBMD1	255.255.255.255

~~When the IUT is configured for NAT, the Originating Device in Forwarded NPDUs that originate at the IUT, OD, is equal to the Global IP Address and Port of the IUT's Internet Router. When the IUT is not configured for NAT operation, OD is equal to the IUT.~~

Notes to Tester: The order of the messages transmitted by the IUT is not significant.

Test Steps:

- TRANSMIT  
 DA = Local IP Broadcast,  
 SA = D1,  
 Original-Broadcast-NPDU,  
 NPDU = Who-Is
- RECEIVE  
 DA = BBMD1,  
~~SA = IUT,~~  
 Forwarded-NPDU,  
 Originating-Device = D1,  
 NPDU = Who-Is
- IF (the IUT responds with Unicast I-Am) THEN  
 RECEIVE DESTINATION = D1,  
 Original-Unicast-NPDU,  
 NPDU = I-Am  
**ELSE**  
**RECEIVE**  
 DA = Local IP Broadcast,  
~~SA = IUT,~~  
 Original-Broadcast-NPDU,  
 NPDU = I-Am
- ~~RECEIVE~~  
 RECEIVE  
 DA=BBMD1,  
~~SA=IUT,~~  
 Forwarded-NPDU,  
 Originating-Device = ~~OD~~IUT,  
 NPDU = I-Am

~~Notes to Tester: The order of the messages transmitted by the IUT is not significant.~~

### 14.3.3.3 Broadcast Distribution Table Operations

[Change **Clause 14.3.3**, p. 625]

[Reviewer Note: IUTs claiming PR16 and less need to accept Write-Broadcast-Distribution-Table message.]

#### ~~14.3.3.3.3~~ 12.3.3.3 Verify Broadcast Distribution Table Created from the Configuration Saved During the Previous Session

Purpose: To verify that a BBMD will update the BDT in the local configuration database and initialize it at startup.

Configuration Requirements: The IUT's BDT does not consist of the same entries *that are as are* written in step 1.

Test Steps:

1. TRANSMIT
  - DA = IUT,
  - SA = D1,
  - Write-Broadcast-Distribution-Table,
  - (List of BDT entries ~~consisting of three entries~~ *at least one of which is different from what it has*

<del>IUT</del>	<del>255.255.255.255</del>
<del>BBMD1</del>	<del>255.255.255.255</del>
<del>BBMD2</del>	<del>255.255.255.255</del>
2. RECEIVE
  - DA = D1,
  - SA = IUT,
  - BVLC-Result,
  - 'Result Code' = Successful completion
3. WAIT (-Vendor specified period for BDT to be saved in non-volatile memory)
4. MAKE (the IUT reset)
5. TRANSMIT
  - DA = IUT,
  - SA = D1,
  - Read-Broadcast-Distribution-Table
6. RECEIVE
  - DA = D1,
  - SA = IUT,
  - Read-Broadcast-Distribution-Table-Ack,
  - List of BDT Entries
7. CHECK (*IUT's BDT holds the entries which it was configured in step 1*) ~~List of BDT Entries consisting of three entries (order unspecified)~~

<del>IUT</del>	<del>255.255.255.255</del>
<del>BBMD1</del>	<del>255.255.255.255</del>
<del>BBMD2</del>	<del>255.255.255.255</del>

→

[Add new **Clause 12.3.3.X1**, p.625]

[Reviewer Note: Clause J.4.4.2 mandates a change and that all devices claiming Protocol\_Revision >= 17 shall behave in this changed way.]

### 12.3.3.X1 Write-BDT service is required to return Write-BDT-NAK

Purpose: To verify that any IUT with Protocol\_Revision claimed as 17 or higher, will return Write-Broadcast-Distribution-Table NAK to every Write-Broadcast-Distribution-Table request.

Configuration Requirements: If the Protocol\_Revision claimed is less than 17, this test shall be skipped.

Test Steps:

1. TRANSMIT Write-Broadcast-Distribution-Table
2. RECEIVE BVLC-Result,  
    'Result Code' = Write-Broadcast-Distribution-Table NAK

### ~~14.6.3.1~~ 12.3.6 Foreign Device Management

#### ~~14.6.3.1.2.3.6.3~~ 12.3.6.3 Foreign Device Table Timer Operations

[Change **Clause 14.6.3.1**, p. 625]

[Reviewer Note: Added in Network Port FDT checks.]

#### ~~14.6.3.1.2.3.6.3.1~~ 12.3.6.3.1 Non-Zero-Duration Foreign Device Table Timer Operations

Purpose: To verify that the IUT will handle FDT timer operations: finite time Foreign Device registration, re-registration, adding grace period to the supplied Time-To-Live parameter, and FDT entry clearing upon timer expiration.

Configuration Requirements: The TD shall take the role of foreign device FD2. The IUT's FDT must be empty. *The Network Port object for the BACnet/IP network is NP.*

*Notes to Tester: The accuracy of the FDT timer shall be specified by the vendor.*

Test Steps:

1. TRANSMIT  
    DA = IUT,  
    SA = FD2,  
    Register-Foreign-Device,  
    'Time-To-Live' = 60
2. RECEIVE  
    DA = FD2,  
    SA = IUT,  
    BVLC-Result,  
    'Result Code' = 0
3. WAIT (10 seconds)
4. TRANSMIT  
    DA = IUT,  
    SA = FD2,  
    Read-Foreign-Device-Table
5. RECEIVE  
    DA = FD2,  
    SA = IUT,  
    Read-Foreign-Device-Table-Ack,  
    B/IP address of FD2, Time-To-Live = 60, Remaining-Time = 80 minus test execution time.  
    -- (50 is also acceptable if Protocol\_Revision < 7)
6. *IF Protocol\_Revision >= 17 THEN*  
    *VERIFY NP, BBMD\_Foreign\_Device\_Table = ( (B/IP address of FD2, 60, 80 - execution time) )*
7. TRANSMIT  
    DA = IUT,  
    SA = FD2,  
    Register-Foreign-Device,

'Time-To-Live' = 40

8. RECEIVE

DA = FD2,  
SA = IUT,  
BVLC-Result,  
'Result Code' = 0

9. WAIT (30 seconds)

10. TRANSMIT

DA = IUT,  
SA = FD2,  
Read-Foreign-Device-Table

11. RECEIVE

DA = FD2,  
SA = IUT,  
Read-Foreign-Device-Table-Ack,  
B/IP address of FD2, Time-To-Live = 40, Remaining-Time = 40 minus test execution time  
-- (10 is also acceptable if Protocol\_Revision < 7)

12. IF Protocol\_Revision >= 17 THEN

VERIFY NP, BBMD\_Foreign\_Device\_Table = ( (B/IP address of FD2, 40, 40 - execution time) )

13. WAIT (50 seconds)

14. TRANSMIT

DA = IUT,  
SA = FD2,  
Read-Foreign-Device-Table

15. RECEIVE

DA = FD2,  
SA = IUT,  
Read-Foreign-Device-Table-Ack,  
(No FDT entries)

16. IF Protocol\_Revision >= 17 THEN

VERIFY NP, BBMD\_Foreign\_Device\_Table = ( )

~~Notes to tester: the accuracy of the FDT timer shall be specified by the vendor.~~

[Change **Clause 14.6.3.2**, p. 625]

[Reviewer Note: Added in Network Port FDT checks.]

#### ~~14.6.3.2~~ 12.3.6.3.2 Zero-Duration Foreign Device Timer Operations

Purpose: To verify that the IUT will handle Foreign Device registration with Time-To-Live parameter equal to zero and clears FDT entry upon timer expiration.

Configuration Requirements: The TD shall take the role of foreign device FD2. The IUTs FDT must be empty. *The Network Port object for the BACnet/IP network is NP.*

*Notes to Tester: The accuracy of the FDT timer shall be specified by the vendor.*

Test Steps:

1. TRANSMIT

DA = IUT,  
SA = FD2,  
Register-Foreign-Device-Table,  
'Time-To-Live' = 0

2. RECEIVE

DA = FD2,  
SA = IUT,  
BVLC-Result,

- 'Result Code' = 0
3. WAIT (10 seconds)
  4. TRANSMIT
    - DA = IUT,
    - SA = FD2,
    - Read-Foreign-Device-Table
  5. RECEIVE
    - DA = FD2,
    - SA = IUT,
    - Read-Foreign-Device-Table-Ack,
    - B/IP address of FD2, Time-To-Live = 0, Remaining-Time = 20 minus test execution time  
-- (0 is also acceptable if Protocol\_Revision < 7)
  6. *IF Protocol\_Revision >= 17 THEN*  
*VERIFY NP, BBMD\_Foreign\_Device\_Table = ( (B/IP address of FD2, 0, 20 - execution time) )*
  7. WAIT (30 seconds)
  8. TRANSMIT
    - DA = IUT,
    - SA = FD2,
    - Read-Foreign-Device-Table
  9. RECEIVE
    - DA = FD2,
    - SA = IUT,
    - Read-Foreign-Device-Table-Ack,
    - (No FDT entries)
  10. *IF Protocol\_Revision >= 17 THEN*  
*VERIFY NP, BBMD\_Foreign\_Device\_Table = ( )*

~~Note to tester: The accuracy of the FDT timer shall be specified by the vendor.~~

### ~~14.7.12.3.7~~ Broadcast Management (BBMD, Foreign Devices, Local Application)

[Reviewer Note: This makes changes to all tests in Section 14.7 that use the I-Am service, to allow the unicast form of the response.]

#### ~~14.7.12.3.7.1~~ Broadcast Message from Directly Connected IP Subnet

[Change **Clause 14.7.1.1**, p. 625]

##### ~~14.7.12.3.7.1.1~~ Broadcast Message from Directly Connected IP Subnet (One-hop Distribution)

Configuration Requirements: The IUT's BDT shall contain the following three entries:

B/IP Address	Broadcast Distribution Mask
IUT	IP Subnet 1 subnet mask
BBMD1	IP Subnet 2 subnet mask
BBMD2	IP Subnet 3 subnet mask

*The TD shall be on the same subnet as the IUT. D1 is a device on a different IP subnet than the TD.*

*Notes to Tester: Steps 2-5 are the distribution of the Who-Is request to the devices considered to be members of the BACnet network, step 6 is steps 6-10 are the distribution of the I-Am response from the local application. The order of the messages transmitted by the IUT is not significant.*

Test Steps:

1. TRANSMIT
  - DA = Local IP Broadcast,
  - SA = D1,
  - Original-Broadcast-NPDU,
  - NPDU = Who-Is

2. RECEIVE  
 DA = Directed IP Broadcast to IP Subnet 2,  
~~SA = IUT,~~  
 Forwarded-NPDU,  
 Originating-Device = D1,  
 NPDU = Who-Is
3. RECEIVE  
 DA = Directed IP Broadcast to IP Subnet 3,  
~~SA = IUT,~~  
 Forwarded-NPDU,  
 Originating-Device = D1,  
 NPDU = Who-Is
4. RECEIVE  
 DA = FD1,  
~~SA = IUT,~~  
 Forwarded-NPDU,  
 Originating-Device = D1,  
 NPDU = Who-Is
5. RECEIVE  
 DA = FD2,  
~~SA = IUT,~~  
 Forwarded-NPDU,  
 Originating-Device = D1,  
 NPDU = Who-Is
6. IF (the IUT responds with Unicast I-Am) THEN  
 RECEIVE DA = D1, ~~SA = IUT,~~  
 Original-Unicast-NPDU,  
 NPDU = I-Am  
 ELSE  
 RECEIVE DA = Local IP Broadcast, ~~SA = IUT,~~  
 Original-Broadcast-NPDU,  
 NPDU = I-Am  
 RECEIVE DA = Directed IP Broadcast to IP Subnet 2, ~~SA = IUT,~~  
 Forwarded-NPDU,  
 Originating-Device = IUT,  
 NPDU = I-Am  
 RECEIVE DA = Directed IP Broadcast to IP Subnet 3, ~~SA = IUT,~~  
 Forwarded-NPDU,  
 Originating-Device = IUT,  
 NPDU = I-Am  
 RECEIVE DA = FD1, ~~SA = IUT,~~  
 Forwarded-NPDU,  
 Originating-Device = IUT,  
 NPDU = I-Am  
 RECEIVE DA = FD2, ~~SA = IUT,~~  
 Forwarded-NPDU,  
 Originating-Device = IUT,  
 NPDU = I-Am

Notes to Tester: The order of the messages transmitted by the IUT is not significant.

[Change **Clause 14.7.1.2**, p. 625]

#### **14.7.1.2/2.3.7.1.2 Broadcast Message from Directly Connected IP Subnet (Two-hop Distribution)**

Configuration Requirements: The BDT shall contain the following three entries:

B/IP Address	Broadcast Distribution Mask
IUT	255.255.255.255
BBMD1	255.255.255.255

BBMD2	255.255.255.255
-------	-----------------

*Notes to Tester:* Steps 2-5 are the distribution of the Who-Is request to the devices considered to be members of the BACnet network, ~~step 6 is~~ ~~steps 6-10 are~~ the distribution of the I-Am response from the local application. *The order of the messages transmitted by the IUT is not significant.*

Test Steps:

1. TRANSMIT DA = Local IP Broadcast, SA = D1,  
Original-Broadcast-NPDU,  
NPDU = Who-Is
2. RECEIVE DA = BBMD1, ~~SA = IUT,~~  
Forwarded-NPDU,  
Originating-Device = D1,  
NPDU = Who-Is
3. RECEIVE DA = BBMD2, ~~SA = IUT,~~  
Forwarded-NPDU,  
Originating-Device = D1,  
NPDU = Who-Is
4. RECEIVE DA = FD1, ~~SA = IUT,~~  
Forwarded-NPDU,  
Originating-Device = D1,  
NPDU = Who-Is
5. RECEIVE DA = FD2, ~~SA = IUT,~~  
Forwarded-NPDU,  
Originating-Device = D1,  
NPDU = Who-Is
6. *IF (the IUT responds with Unicast I-Am) THEN*  
RECEIVE DA = D1, ~~SA = IUT,~~  
Original-Unicast-NPDU,  
NPDU = I-Am  
*ELSE*  
RECEIVE DA = Local IP Broadcast, ~~SA = IUT,~~  
Original-Broadcast-NPDU,  
NPDU = I-Am  
RECEIVE DA = BBMD1, ~~SA = IUT,~~  
Forwarded-NPDU,  
Originating-Device = IUT,  
NPDU = I-Am  
RECEIVE DA = BBMD2, ~~SA = IUT,~~  
Forwarded-NPDU,  
Originating-Device = IUT,  
NPDU = I-Am  
RECEIVE DA = FD1, ~~SA = IUT,~~  
Forwarded-NPDU,  
Originating-Device = IUT,  
NPDU = I-Am  
RECEIVE DA = FD2, ~~SA = IUT,~~  
Forwarded-NPDU,  
Originating-Device = IUT,  
NPDU = I-Am

*Notes to Tester:* The order of the messages transmitted by the IUT is not significant.

#### **14.7.2.12.3.7.2 Broadcast Message Forwarded by a Peer BBMD**

[Change **Clause 14.7.2.1**, p. 625]

##### **14.7.2.12.3.7.2.1 Broadcast Message Forwarded by a Peer BBMD (One-hop Distribution)**



Configuration Requirements: The BDT shall be configured as in test 14.7.1.1.

*Notes to Tester:* Steps 2-3 are the distribution of the Who-Is request to the devices considered to be members of the BACnet network, *step 4 is steps 4-8 are* the distribution of the I-Am response from the local application. *The order of the messages transmitted by the IUT is not significant.*

Test Steps:

1. TRANSMIT DA = Directed IP Broadcast to IP Subnet 1, SA = BBMD1,  
Forwarded-NPDU,  
Originating-Device = D2,  
NPDU = Who-Is
2. RECEIVE DA = FD1, ~~SA = IUT,~~  
Forwarded-NPDU,  
Originating-Device = D2,  
NPDU = Who-Is
3. RECEIVE DA = FD2, ~~SA = IUT,~~  
Forwarded-NPDU,  
Originating-Device = D2,  
NPDU = Who-Is
4. IF (the IUT responds with Unicast I-Am) THEN  
RECEIVE DESTINATION = D2, ~~SA = IUT,~~  
Original-Unicast-NPDU,  
NPDU = I-Am  
ELSE  
RECEIVE DA = Local IP Broadcast, ~~SA = IUT,~~  
Original-Broadcast-NPDU,  
NPDU = I-Am  
RECEIVE DA = Directed IP Broadcast to IP Subnet 2, ~~SA = IUT,~~  
Forwarded-NPDU,  
Originating-Device = IUT,  
NPDU = I-Am  
RECEIVE DA = Directed IP Broadcast to IP Subnet 3, ~~SA = IUT,~~  
Forwarded-NPDU,  
Originating-Device = IUT,  
NPDU = I-Am  
RECEIVE DA = FD1, ~~SA = IUT,~~  
Forwarded-NPDU,  
Originating-Device = IUT,  
NPDU = I-Am  
RECEIVE DA = FD2, ~~SA = IUT,~~  
Forwarded-NPDU,  
Originating-Device = IUT,  
NPDU = I-Am

~~Notes to Tester: The order of the messages transmitted by the IUT is not significant.~~

[Change **Clause 14.7.2.2**, p. 625]

#### **14.7.2.2/12.3.7.2.2 Broadcast Message Forwarded by a Peer BBMD (Two-hop Distribution)**

Configuration Requirements: The BDT shall be configured as in test 14.7.1.2

*Notes to Tester:* Steps 2-4 are the distribution of the Who-Is request to the devices considered to be members of the BACnet network, *step 5 is steps 5-9 are* the distribution of the I-Am response from the local application. *The order of the messages transmitted by the IUT is not significant.*

Test Steps:

1. TRANSMIT ~~DA = IUT,~~ SOURCEA = BBMD1,  
Forwarded-NPDU,  
Originating-Device = D2,  
NPDU = Who-Is

2. RECEIVE DA = Local IP Broadcast, ~~SA = IUT~~,  
Forwarded-NPDU,  
Originating-Device = D2,  
NPDU = Who-Is
3. RECEIVE DA = FD1, ~~SA = IUT~~,  
Forwarded-NPDU,  
Originating-Device = D2,  
NPDU = Who-Is
4. RECEIVE DA = FD2, ~~SA = IUT~~,  
Forwarded-NPDU,  
Originating-Device = D2,  
NPDU = Who-Is
5. *IF (the IUT responds with Unicast I-Am) THEN*  
RECEIVE DESTINATION = D2, ~~SA = IUT~~,  
Original-Unicast-NPDU,  
NPDU = I-Am  
*ELSE*  
RECEIVE DA = Local IP Broadcast, ~~SA = IUT~~,  
Original-Broadcast-NPDU,  
NPDU = I-Am  
RECEIVE DA = BBMD1, ~~SA = IUT~~,  
Forwarded-NPDU,  
Originating-Device = IUT,  
NPDU = I-Am  
RECEIVE DA = BBMD2, ~~SA = IUT~~,  
Forwarded-NPDU,  
Originating-Device = IUT,  
NPDU = I-Am  
RECEIVE DA = FD1, ~~SA = IUT~~,  
Forwarded-NPDU,  
Originating-Device = IUT,  
NPDU = I-Am  
RECEIVE DA = FD2, ~~SA = IUT~~,  
Forwarded-NPDU,  
Originating-Device = IUT,  
NPDU = I-Am

**Notes to Tester:** The order of the messages transmitted by the IUT is not significant.

### **14.7.3.12.3.7.3 Broadcast Message from a Foreign Device**

[Change **Clause 14.7.3.1**, p. 625]

#### **14.7.3.12.3.7.3.1 Broadcast Message From a Foreign Device (One-hop Distribution)**

Configuration Requirements: The BDT shall be configured as in test 14.7.1.1.

*Notes to Tester:* Steps 2-5 are the distribution of the Who-Is request to the devices considered to be members of the BACnet network, ~~step 6 is~~ ~~steps 6-10 are~~ the distribution of the I-Am response from the local application. *The order of the messages transmitted by the IUT is not significant.*

Test Steps:

1. TRANSMIT ~~DA = IUT~~, SA = FD1,  
Distribute-Broadcast-To-Network,  
NPDU = Who-Is
2. RECEIVE DA = Local IP Broadcast, ~~SA = IUT~~,  
Forwarded-NPDU,  
Originating-Device = FD1,  
NPDU = Who-Is
3. RECEIVE DA = BBMD1, ~~SA = IUT~~,  
Forwarded-NPDU,

- Originating-Device = FD1,  
NPDU = Who-Is
4. RECEIVE DA = BBMD2, ~~SA = IUT,~~  
Forwarded-NPDU,  
Originating-Device = FD1,  
NPDU = Who-Is
  5. RECEIVE DA = FD2, ~~SA = IUT,~~  
Forwarded-NPDU,  
Originating-Device = FD1,  
NPDU = Who-Is
  6. *IF (the IUT responds with Unicast I-Am) THEN*  
RECEIVE DA = FD1, ~~SA = IUT,~~  
Original-Unicast-NPDU,  
NPDU = I-Am  
*ELSE*  
RECEIVE DA = Local IP Broadcast, ~~SA = IUT,~~  
Original-Broadcast-NPDU,  
NPDU = I-Am  
RECEIVE DA = Directed IP Broadcast to IP Subnet 2, ~~SA = IUT,~~  
Forwarded-NPDU,  
Originating-Device = IUT,  
NPDU = I-Am  
RECEIVE DA = Directed IP Broadcast to IP Subnet 3, ~~SA = IUT,~~  
Forwarded-NPDU,  
Originating-Device = IUT,  
NPDU = I-Am  
RECEIVE DA = FD1, ~~SA = IUT,~~  
Forwarded-NPDU,  
Originating-Device = IUT,  
NPDU = I-Am  
RECEIVE DA = FD2, ~~SA = IUT,~~  
Forwarded-NPDU,  
Originating-Device = IUT,  
NPDU = I-Am

~~Notes to Tester: The order of the messages transmitted by the IUT is not significant.~~

[Change **Clause 14.7.3.2**, p. 625]

#### ~~14.7.3.2~~12.3.7.3.2 Broadcast Message From a Foreign Device (Two-hop Distribution)

Configuration Requirements: The BDT and FDT shall be configured as in test 14.7.1.2.

*Notes to Tester:* Steps 2-5 are the distribution of the Who-Is request to the devices considered to be members of the BACnet network, ~~step 6 is steps 6-10 are~~ the distribution of the I-Am response from the local application. *The order of the messages transmitted by the IUT is not significant.*

Test Steps:

1. TRANSMIT ~~DA = IUT,~~ SA = FD1,  
Distribute-Broadcast-To-Network,  
NPDU = Who-Is
2. RECEIVE DA = Local IP Broadcast, ~~SA = IUT,~~  
Forwarded-NPDU,  
Originating-Device = FD1,  
NPDU = Who-Is
3. RECEIVE DA = BBMD1, ~~SA = IUT,~~  
Forwarded-NPDU,  
Originating-Device = FD1,  
NPDU = Who-Is

4. RECEIVE DA = BBMD2, ~~SA = IUT~~,  
Forwarded-NPDU,  
Originating-Device = FD1,  
NPDU = Who-Is
5. RECEIVE DA = FD2, ~~SA = IUT~~,  
Forwarded-NPDU,  
Originating-Device = FD1,  
NPDU = Who-Is
6. IF (the IUT responds with Unicast I-Am) THEN  
RECEIVE DA = FD1, ~~SA = IUT~~,  
Original-Unicast-NPDU,  
NPDU = I-Am  
ELSE  
RECEIVE DA = Local IP Broadcast, ~~SA = IUT~~,  
Original-Broadcast-NPDU,  
NPDU = I-Am  
RECEIVE DA = BBMD1, ~~SA = IUT~~,  
Forwarded-NPDU,  
Originating-Device = IUT,  
NPDU = I-Am  
RECEIVE DA = BBMD2, ~~SA = IUT~~,  
Forwarded-NPDU,  
Originating-Device = IUT,  
NPDU = I-Am  
RECEIVE DA = FD1, ~~SA = IUT~~,  
Forwarded-NPDU,  
Originating-Device = IUT,  
NPDU = I-Am  
RECEIVE DA = FD2, ~~SA = IUT~~,  
Forwarded-NPDU,  
Originating-Device = IUT,  
NPDU = I-Am

Notes to Tester: The order of the messages transmitted by the IUT is not significant.

[Change **Title of Clause 14.8** and renumber, p. 625]

[Reviewer Note: Renumber to allow multiple foreign device tests. Add in check that lifetime is within the range required by NM-FDR-A.]

#### **~~14.8 Registering as a Foreign Device~~ 12.3.8 Foreign Device Tests**

[Change **Clause Number 14.8 to 12.3.8.1**, p. 625]

##### **~~14.8~~ 12.3.8.1 Registering as a Foreign Device**

~~Dependencies: None~~

~~BACnet Reference Clause: J.5.2~~

Purpose: This test case verifies that the IUT can register as a foreign device with a BBMD.

Test Concept: The IUT is caused to register as a foreign device with the TD.

Configuration Requirements: The IUT is configured to register as a foreign device with the TD.

Test Steps:

1. RECEIVE DESTINATION = TD, SOURCE = IUT,  
Register-Foreign-Device

*'Time-to-Live'* = (any value between 30 seconds and 9 hours)

2. TRANSMIT DESTINATION = IUT, SOURCE = TD,

BVLC-Result,

'Result Code' = Successful completion

[Add new **Clause 12.3.8.X1**, p. 625]

### **12.3.8.X1 Register-Foreign-Device Enable and Disable Test**

Purpose: Verify that the option to issue Register-Foreign-Device requests can be configured by the product end-user.

Test Concept: Using a product end-user interface, configure the mode for use of Register-Foreign-Device requests, and then configure the mode to cease use of Register-Foreign-Device requests.

Configuration Requirements: The means by which the product is here configured shall be part of the product's end-user interface. BBMD1 is the TD simulating a correctly functioning BBMD implementation.

Test Steps:

1. MAKE (IUT enter mode for use of Register-Foreign-Device requests)
2. RECEIVE DA = BBMD1,  
Register-Foreign-Device
3. TRANSMIT BVLC-Result,  
'Result Code' = Successful completion
4. MAKE (the IUT not in mode for use of Register-Foreign-Device requests)
5. WAIT (more than 31 seconds longer than the 'Time-to-Live' parameter used in Register-Foreign-Device requests)
6. CHECK (that the IUT did not send any Register-Foreign-Device requests)

[Add new **Clause 12.3.8.X2** p. 625]

[Reviewer Note: This tests in continuous manner what 12.3.9.1 tests just once.]

### **12.3.8.X2 Recurring Register-Foreign-Device Test**

Purpose: Verify that mode for use of Register-Foreign-Device and setting of 'BBMD Address' parameter are persistent across reset, and that the issuance of Register-Foreign-Device precedes the first issuance of any broadcast, when in that mode.

Test Concept: IUT is put in a mode to use Register-Foreign-Device requests, and it is observed that Register-Foreign-Device requests are sent sufficiently frequently to prevent expiration of the registration at the BBMD.

Configuration Requirements: The product's setting of 'BBMD Address' parameter is configured as BBMD1. BBMD1 is the TD simulating a correctly functioning BBMD implementation.

Notes to Tester: There is no need for the recurring request to be sent any more quickly than precisely the 'Time-to-Live' since the standard mandates that the BBMD preserve the registration for 30 seconds past the 'Time-to-Live'.

Test Steps:

1. MAKE (IUT enter mode for use of Register-Foreign-Device requests)
2. RECEIVE DA = BBMD1,  
Register-Foreign-Device
3. TRANSMIT BVLC-Result,  
'Result Code' = Successful completion
4. BEFORE (the time configured for the 'Time-to-Live' parameter used for Register-Foreign-Device requests)  
RECEIVE DA = BBMD1,  
Register-Foreign-Device
5. TRANSMIT BVLC-Result,  
'Result Code' = Successful completion
6. BEFORE (the time configured for the 'Time-to-Live' parameter used for Register-Foreign-Device requests)  
RECEIVE DA = BBMD1,

#### Register-Foreign-Device

7. TRANSMIT BVLC-Result,  
    'Result Code' = Successful completion

[Add new **Clause 12.3.8.X3** p. 625]

[Reviewer Note: This tests that the behavior in test 12.3.8 can be configured by the product end-user.]

### **12.3.8.X3 BBMD Address Configuration Test**

Purpose: Verify that the parameter in Register-Foreign-Device in test 12.3.8 can be configured by the product end-user.

Test Concept: Using a product end-user interface, configure the 'BBMD Address' parameter that is used in Register-Foreign-Device requests.

Configuration Requirements: The means by which the product is configured for a 'BBMD Address' can be anything in the product's end-user interface. BBMD1 is the TD simulating a correctly functioning BBMD implementation.

Test Steps:

1. MAKE (through the product's end-user interface, the setting of 'BBMD Address' parameter equal BBMD1)
2. MAKE (IUT enter mode for use of Register-Foreign-Device requests)
3. RECEIVE DA = BBMD1,  
    Register-Foreign-Device
4. TRANSMIT BVLC-Result,  
    'Result Code' = Successful completion

[Add new **Clause 12.3.8.X4** p. 625]

[Reviewer Note: This tests in the specific case of startup, what test 12.3.9.1 expects to observe during ordinary ongoing operation.]

### **12.3.8.X4 Transmits a Broadcast at Startup preceded by Register-Foreign-Device**

Purpose: Verify that mode for use of Register-Foreign-Device and setting of 'BBMD Address' parameter are persistent across reset, and that the issuance of Register-Foreign-Device precedes the first issuance of any broadcast, when in that mode.

Test Concept: IUT is put in a mode to use Register-Foreign-Device requests persistently so it will be re-established, then IUT is reset, and the timing of Register-Foreign-Device request to re-establish that precedes the first issuance of any broadcast.

Configuration Requirements: The product's setting of 'BBMD Address' parameter is configured as BBMD1. BBMD1 is the TD simulating a correctly functioning BBMD implementation.

Notes to Tester: For the I-Am, one can precede the Register-Foreign-Device command, as long as then after the Register-Foreign-Device occurs, it is followed by a Distribute-Broadcast-To-Network again, of that I-Am.

Test Steps:

1. MAKE (IUT enter mode for use of Register-Foreign-Device requests, persistently  
    so it will be re-established after any reset)
2. MAKE (IUT reset)
3. RECEIVE DA = BBMD1,  
    Register-Foreign-Device
4. TRANSMIT BVLC-Result,  
    'Result Code' = Successful completion
5. RECEIVE DA = BBMD1,  
    Distribute-Broadcast-To-Network,  
    NPDU = (any broadcast)
6. TRANSMIT BVLC-Result,

'Result Code' = Successful completion

[Add new **Clause 12.3.8.X5** p. 625]

[Reviewer Note: Adds verification that the behavior in test 12.3.8 can be configured by the product end-user.]

#### **12.3.8.X5 Time-to-Live Configuration Test**

Purpose: Verify that the parameter in Register-Foreign-Device in test 12.3.8 can be configured by the product end-user, through a reasonable range (120 through 28800 is sufficient; the absolute upper limit is 65535 seconds, approximately 17 hours).

Test Concept: Using a product end-user interface, configure the 'Time-to-Live' parameter that is used in Register-Foreign-Device requests.

Configuration Requirements: The means by which the product is configured can be anything in the product's end-user interface. BBMD1 is the TD simulating a correctly functioning BBMD implementation.

Test Steps:

1. MAKE (through the product's end-user interface, the setting of 'Time-to-Live' parameter equal 120, or any larger value supported by the implementation)
2. MAKE (IUT enter mode for use of Register-Foreign-Device requests)
3. RECEIVE DA = BBMD1,  
Register-Foreign-Device,  
'Time-to-Live' = (value configured in step 1)
4. TRANSMIT BVLC-Result,  
'Result Code' = Successful completion

#### **~~14.9~~12.3.9 Initiating BVLL Service Requests Conveying an NPDU**

[Change **Clause 14.9.1**, p. 625]

[Reviewer Note: Modified test to enforce the behavior specified in Clause J.2.12.]

#### **~~14.9.1~~12.3.9.1 Distribute-Broadcast-To-Network**

~~Dependencies: 14.8, "Registering as a Foreign Device"~~

~~BACnet Reference Clause: J.2.10~~

Purpose: This test case verifies that the IUT, registered as a foreign device, can issue a request to a BBMD to broadcast the message on all subnets in the BBMD's BDT.

Test Concept: The IUT is configured to register itself as a foreign device with the TD, then after registration is achieved it is caused to initiate a broadcast message to be conveyed to the BBMD for distribution. If the IUT does not support foreign device registration, or cannot initiate broadcast messages conveying a BACnet NPDU, then this test shall be omitted.

Test Steps:

1. RECEIVE DESTINATION = TD, SOURCE = IUT,  
Register-Foreign-Device
2. TRANSMIT DESTINATION = IUT, SOURCE = TD,  
BVLC-Result,  
'Result Code' = Successful completion
3. MAKE (*a condition that would make the IUT normally* initiate a broadcast)
4. RECEIVE DESTINATION = TD, SOURCE = IUT,  
Distribute-Broadcast-To-Network
5. CHECK (*that the IUT does not transmit an Original-Broadcast-NPDU on this port*)

[Add **Clause 12.3.X1**, p.625]

### **12.3.X1 BBMD Configuration Tests – A side**

#### **12.3.X1.1 Read-Broadcast-Distribution-Table Initiation**

Purpose: To verify that an IUT which configures BBMDs is able to query and present an arbitrary broadcast distribution table.

Test Steps:

1. RECEIVE Read-Broadcast-Distribution-Table
2. TRANSMIT Read-Broadcast-Distribution-Table-Ack,  
List of BDT Entries
3. CHECK (the IUT presents the table entries, in any order)

#### **12.3.X1.2 Write-Broadcast-Distribution-Table Initiation**

Purpose: To verify that an IUT which configures BBMDs is able to generate an arbitrary Write-Broadcast-Distribution-Table request.

Test Steps:

1. MAKE (the IUT generate a Write-Broadcast-Distribution-Table to configure the TD with a tester selected BDT, B)
2. RECEIVE Write-Broadcast-Distribution-Table,  
(B: a valid list of BDT entries)
2. TRANSMIT BVLC-Result,  
'Result Code' = Successful completion

#### **12.3.X1.3 Read-Foreign-Device-Table Initiation**

Purpose: To verify that an IUT which configures BBMDs is able to query and present an arbitrary foreign device table.

Test Steps:

1. RECEIVE Read-Foreign-Device-Table
2. TRANSMIT Read-Foreign-Device-Table-Ack,  
List of FDT Entries
3. CHECK (the IUT presents the table entries, in any order)

#### **12.3.X1.4 Delete-Foreign-Device-Table-Entry Initiation**

Purpose: To verify that an IUT which configures BBMDs is able to generate an arbitrary Delete-Foreign-Device-Table-Entry request.

Configuration Requirements: The IUT is configured with a non-empty FDT.

Test Steps:

1. MAKE (the IUT generate a Delete-Foreign-Device-Table-Entry to configure the TD with a tester selected FDT, F)
2. RECEIVE Delete-Foreign-Device-Table-Entry,  
(F: a valid FDT entry in IUT's FDT)
3. TRANSMIT BVLC-Result,  
'Result Code' = Successful completion

[Add **Clause 12.3.X2**, p.625]



### 12.3.X2 BBMD Configuration Tests – B side

[Reviewer Note: NM-BBMD-B specifically mandates this capacity behavior is supported by the product.]

#### 12.3.X2.1 Broadcast-Distribution-Table Holds at Least 5 Entries

Purpose: Verify that IUT implements capacity mandated for the product by NM-BBMD-B.

Test Concept: Fill the IUT's broadcast distribution table with at least five distinct peer BBMDs entries (in addition to the entry containing the address of itself in the table).

Notes to Tester: In a device claiming Protocol\_Revision 16 or less, the means by which the product's Broadcast Distribution Table is configured is not restricted to BACnet network transmissions and can be through the product's end-user interface.

Test Steps:

1. MAKE (IUT enter mode functioning as a BBMD implementation)
2. MAKE (the IUT's broadcast distribution table contain its own entry and entries for at least 5 other BBMDs)
3. TRANSMIT Read-Broadcast-Distribution-Table
4. RECEIVE Read-Broadcast-Distribution-Table-Ack,  
'List of BDT Entries' = (the table as configured, in any order)

[Reviewer Note: NM-BBMD-B specifically mandates this capacity behavior is supported by BBMDs.]

#### 12.3.X2.2 Holds at Least 5 Foreign Device Registrations

Purpose: Verify that when configured to accept foreign device registrations, the IUT supports at least five simultaneous foreign device registrations.

Test Concept: The IUT is configured to support foreign device registrations. Five Register-Foreign-Device requests are sent from 5 different devices, to verify that it supports five registrations simultaneously in the FDT.

Configuration Requirements: Set BBMD\_Accept\_FD\_Registrations in the Network Port object representing the port operating as a BBMD to TRUE. The TD will be configured to emulate 5 devices.

Test Steps:

1. REPEAT X = 1 to 5 {  
    TRANSMIT Register-Foreign-Device  
    SOURCE = (device X)  
    'Time-to-Live' = (a value longer than the length of the test)  
    RECEIVE BVLC-Result,  
    'Result Code' = Successful completion  
}
2. TRANSMIT Read-Foreign-Device-Table
3. RECEIVE Read-Foreign-Device-Table-Ack  
    List of FDT entries = (the 5 registered devices)

[Reviewer Note: The standard specifically mandates that BBMD\_Accept\_FD\_Registrations property is writable if present in BBMDs.]

#### 12.3.X2.3 Negative Foreign Device Registration when BBMD\_Accept\_FD\_Registrations is FALSE

Purpose: Verify that when BBMD\_Accept\_FD\_Registrations is configured as FALSE, the BBMD will accept no more foreign device registrations.

Test Concept: The IUT is configured with BBMD\_Accept\_FD\_Registrations property as FALSE. Then it is verified that no more Register-Foreign-Device registrations succeed, though those already in the FDT operate as normal.

Configuration Requirements: BBMD\_Accept\_FD\_Registrations in the Network Port object representing the port is initially TRUE.

Test Steps:

1. WRITE BBMD\_Accept\_FD\_Registrations = FALSE
2. TRANSMIT ReinitializeDevice-Request  
    'Reinitialized State of Device' = ACTIVATE\_CHANGES
3. WAIT **Activate Changes Fail Time**
4. TRANSMIT Register-Foreign-Device
5. RECEIVE BVLC-Result,  
    'Result Code' = Register-Foreign-Device NAK

[Reviewer Note: With the advent of Network Port objects, BBMDs now need to accept hostname BDT entries.]

### 12.3.X2.4 Broadcast Distribution Table Configuration via Hostname Entries

Purpose: Verify that the IUT accepts and resolves hostname entries in the BBMD\_Broadcast\_Distribution\_Table.

Test Concept: Fill the BBMD\_Broadcast\_Distribution\_Table with 4 entries: the IUT, an entry with an IP address (IP1), an entry with a resolvable hostname (at IP address IP2), and an entry with a non-resolvable hostname. Send a broadcast that the IUT should distribute to its peer BBMDs and verify that it sends to the resolvable entries. Verify that the Broadcast Distribution Table contains the correct entries.

Configuration Requirements: The IUT is configured to operate as a BBMD and the TD is located on the same IP subnet.

Notes to Tester: The Forwarded-NPDU messages can be received in any order.

Test Steps:

1. WRITE BBMD\_Broadcast\_Distribution\_Table = (4 entries:  
    the IUT,  
    an entry with an IP address,  
    an entry with a resolvable hostname,  
    an entry with a non-resolvable hostname)
2. TRANSMIT ReinitializeDevice-Request  
    'Reinitialized State of Device' = ACTIVATE\_CHANGES
3. WAIT **Activate Changes Fail Time**
4. WAIT until the IUT completes DNS resolution
5. TRANSMIT  
    DA = Local IP Broadcast,  
    SA = D1,  
    Original-Broadcast-NPDU,  
    NPDU = Who-Is-Request
6. RECEIVE  
    DA = IP1,  
    SA = IUT,  
    Forwarded-NPDU,  
        Originating-Device = D1,  
        NPDU = Who-Is
7. RECEIVE  
    DA = IP2,  
    SA = IUT,  
    Forwarded-NPDU,  
        Originating-Device = D1,  
        NPDU = Who-Is

8. READ BDT = BBMD\_Broadcast\_Distribution\_Table -- re-read the table to determine the order the IUT  
-- placed the entries in
9. RECEIVE Read-Broadcast-Distribution-Table-Ack,  
    'List of BDT Entries' =                   (4 entries:  
        the IUT's IP address,  
        the IP address entry,  
        the IP address for the resolved hostname entry,  
        X'000000000000' for the non-resolvable entry  
    -- in the same order as read from BBMD\_Broadcast\_Distribution\_Table)

### 135.1-2019s-6 Renumber Clause 15

#### Rationale

Changes made in 135.1-2019s-5 require Clause 15 to be renumbered.

[Renumber existing **Clause 15**, p.704]

#### **15.14. Reporting Test Results**

A report of the test results shall be provided to the manufacturer that contains:

- (a) a summary of each test case executed;
- (b) the results (pass/fail) for each test case executed; and
- (c) any diagnostic information available for test cases that failed.

## 135.1-2019s-7 Add new and correct existing Object Support Tests

### Rationale

Errors have been identified in a number of object support tests in ANSI/ASHRAE Standard 135.1-2019. In addition, the test coverage is increased by the addition of new tests.

## 7. OBJECT SUPPORT TESTS

### 7.1 Read Support for Properties in the Test Database

[Change **Clause 7.1.1**, p. 19]

[Reviewer Note: Updated the error codes allowed if prior to Protocol\_Revision 13. Added Explanatory Notes To Tester for using with ReadPropertyMultiple. Moved line from Purpose to Test Concept. Added 'server' entry in Abort response.]

#### 7.1.1 Read Support Test Procedure

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

Purpose: To verify that all properties of all objects can be read using ReadProperty and ReadPropertyMultiple services. ~~The test is performed once using ReadProperty and once using ReadPropertyMultiple, if supported. When verifying array properties, the whole array shall be read without using an ARRAY INDEX, where possible.~~

*Test Concept: The test is performed once using ReadProperty and once using ReadPropertyMultiple, if supported. When verifying array properties, the whole array shall be read without using an array index, where possible.*

*Notes to Tester: For cases where the EPICS indicates that the value of a property is unspecified using the "?" symbol, any value that is of the correct datatype shall be considered to be a match. When using the ReadPropertyMultiple service, a received ReadPropertyMultiple-ACK containing the specified Error Class and Error Code shall also be considered a Passing result.*

Test Steps:

1. REPEAT X = (all objects in the IUT's database) DO {
  - REPEAT Y = (all properties in object X) DO {
    - IF (Y = property indicated as not accessible via the ReadProperty Service) THEN
      - TRANSMIT ReadProperty-Request,
      - 'Object Identifier' = X,
      - 'Property Identifier' = Y
    - IF (Protocol\_Revision >= 137) THEN
      - RECEIVE BACnet-Error PDU,
      - 'Error Class' = PROPERTY,
      - 'Error Code' = READ\_ACCESS\_DENIED
    - ELSE
      - RECEIVE BACnet-Error PDU,
      - 'Error Class' = OBJECT | PROPERTY,
      - 'Error Code' = (any of the error codes for an OBJECT or PROPERTY class)
      - ~~Error Class = PROPERTY,~~
      - ~~Error Code = READ\_ACCESS\_DENIED | OTHER~~
      - ~~| (BACnet-Error PDU,~~
      - ~~Error Class = OBJECT,~~
      - ~~Error Code = OTHER),~~
      - ~~| (BACnet-Error-PDU,~~
      - ~~Error Class = SERVICES,~~
      - ~~Error Code = SERVICE\_REQUEST\_DENIED | OTHER)~~

```

ELSE IF (Y is an array and is too long to return given the IUT's
APDU and segmentation limitations) THEN
  TRANSMIT ReadProperty-Request,
    'Object Identifier' = X,
    'Property Identifier' = Y
  RECEIVE BACnet-Abort-PDU,
    'Server' = TRUE,
    'Abort Reason' = SEGMENTATION_NOT_SUPPORTED
    | BUFFER_OVERFLOW
  TRANSMIT ReadProperty-Request,
    'Object Identifier' = X,
    'Property Identifier' = Y,
    'Property Array Index' = 0
  RECEIVE ReadProperty-ACK,
    'Object Identifier' = X,
    'PropertyIdentifier' = Y,
    'Property Array Index' = 0,
    'Property Value' = (N: the number of array elements in Y as indicated in the EPICS)
  REPEAT Z = (1 .. N) DO {
    VERIFY (X), Y = (the value for element Z as indicated in the EPICS), ARRAY INDEX = Z
  }
ELSE
  VERIFY (X), Y = (the value for this property specified in the EPICS)
}
}

```

~~Notes to Tester: For cases where the EPICS indicates that the value of a property is unspecified using the "?" symbol, any value that is of the correct datatype shall be considered to be a match.~~

[Change **Clause 7.1.2**, p. 21]

[Reviewer Note: Revised test to exclude special property identifiers. Add a test step to cover the range add from revision 20.]

### 7.1.2 Non-documented Property Test

Purpose: To verify that all properties contained in every object are documented in the EPICS.

Test Concept: For each object in the EPICS database, attempt to read each standard property that the EPICS does not document as being part of the object.

*Notes to Tester: The objects selected by the tester should include one instance of each supported object type. Where some instances of an object type differ in the set of supported properties, the allowable value ranges for a property, or the writability of a property, then one instance of each variant of that object type should be selected.*

Test Steps:

1. REPEAT X = (a tester selected set of objects) DO {
 REPEAT Y = (0 through 511 except 8 (all), 80 (optional) and 105 (required)) DO {
 IF (the property Y is not in the EPICS for object X) THEN
 TRANSMIT ReadProperty-Request,
 'Object Identifier' = X,
 'Property Identifier' = Y
 RECEIVE BACnet-Error-PDU,
 'Error Class' = PROPERTY,
 'Error Code' = UNKNOWN\_PROPERTY
 }
 }
 }
- 2. IF (Protocol\_Revision >= 20) THEN
 REPEAT Y = (random selection of N property IDs between 4194304 and  $(2^{32} - 1)$ ) DO {
 IF (the property Y is not in the EPICS for object X) THEN
 TRANSMIT ReadProperty-Request,
 Object Identifier' = X,

```

        'Property Identifier' = Y
    RECEIVE BACnet-Error-PDU,
        'Error Class' = PROPERTY,
        'Error Code' = UNKNOWN_PROPERTY
}

```

~~Notes to Tester: The objects selected by the tester should include one instance of each supported object type. Where some instances of an object type differ in the set of supported properties, the allowable value ranges for a property, or the writability of a property, then one instance of each variant of that object type should be selected.~~

[Change **Clause 7.1.3**, p. 21]

[Reviewer Note: BTL-CR-0413 directed to add a Note to Tester mentioning that element by element reading of the properties may be necessary.]

### 7.1.3 Verifying Property\_List against the EPICS

Purpose: To verify the correct content of the Property\_List using the properties in each object as claimed in the EPICS.

Test Concept: Match the properties in each object as claimed in the EPICS, against the content of each object's Property\_List.

Test Conditionality: If Protocol\_Revision is not present, or Protocol\_Revision < 14, then this test shall be skipped.

*Notes to Tester: Object\_Name (77), Object\_Type (79), Object\_Identifier (75), and Property\_List (371) will appear in the EPICS but shall not appear in the Property\_List value. Any proprietary properties that are supported for the object-type shall be in the Property\_List but are not required to appear in the EPICS. The order in which property identifiers appear in the EPICS is not required to match the order that they appear in the Property\_List value. If the whole BACnetARRAY cannot be read because it exceeds the Maximum Transmissible APDU, then the tester shall read it element-by-element in order to obtain the complete value.*

Test Steps:

```

1.  READ OL = Object_List
2.  REPEAT O1 = (each object in the content of OL) DO {
    READ PL = O1, Property_List
    CHECK (that the property identifiers in the EPICS for O1 and those in PL match, except as specified in Notes to
Tester)
}

```

~~Notes to Tester: Object\_Name (77), Object\_Type (79), Object\_Identifier (75), and Property\_List (371) will appear in the EPICS, but shall not appear in the Property\_List value. Any proprietary properties that are supported for the object type shall be in the Property\_List, but are not required to appear in the EPICS. The order in which property identifiers appear in the EPICS, is not required to match the order that they appear in the Property\_List value.~~

## 7.2 Write Support for Properties in Test Database

### 7.2.1 Functional Range Requirements for Property Values

[Change **Clause 7.2.1.3**, p. 22]

[Reviewer Note: Added Support for UTF-8.]

### 7.2.1.3 Octetstrings and Characterstrings

Properties with an octetstring or characterstring datatype shall be tested with a string of the minimum supported length, a string with the maximum supported length, and a string with some length between the two. The vendor shall provide the values of the minimum and maximum string lengths in the EPICS. For string properties that do not have a fixed maximum length, the vendor shall provide a maximum length that is acceptable under normal operating conditions for use in these tests. In such cases, no statement is made as to whether or not a longer string value would or would not be accepted by the property. See Clause 4.4.2.

*When testing character string properties in a device that supports UTF-8 (Protocol\_Revision >= 10), at least one of the data values shall contain multi-byte characters.*

[Change **Clause 7.2.2**, p. 22]

[Reviewer Note: 'Notes to Tester' is missing. Added in special handling for properties in the Network Port object. Moved content from the Purpose into Test Concept as appropriate.]

### 7.2.2 Write Support Test Procedure

Purpose: To verify that all writable properties of all objects can be written to using BACnet WriteProperty and WritePropertyMultiple services. ~~The test is performed once using WriteProperty and once using WritePropertyMultiple. When writing to array properties, the whole array shall be written without using an array index, where possible.~~

*Test Concept: Each writable property is written multiple times verifying the writable range. After each write, the value is verified to have been updated in the property. The test is performed once using WriteProperty and once using WritePropertyMultiple. When writing to array properties, the whole array shall be written without using an array index, where possible.*

*Notes to Tester: An internal process may set the Present\_Value of some properties back to the default value after a successful write, as in the case of a momentary pushbutton, or the Record\_Count property. For properties that exhibit this type of behavior, skip the VERIFY step.*

*Notes to Tester: When a property is currently not writable, the IUT shall return an Error-PDU with 'Error Class' = PROPERTY and 'Error Code' = WRITE\_ACCESS\_DENIED.*

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

Test Steps:

1. REPEAT X = (all objects in the IUT's database, *except Network Port objects*) DO {
  - REPEAT Y = (all writable properties in object X) DO {
    - REPEAT Z = (all values meeting the functional range requirements of 7.2.1, and any additional restrictions placed on the allowable property values by the vendor) DO {
      - WRITE (X), Y = Z,
      - VERIFY (X), Y = Z

[Change **Clause 7.2.3**, p. 23]



[Reviewer Note: This test corrects the use of the READ statement. Added 'Configuration Requirements'. Moved 'Notes to Tester'.]

### 7.2.3 Read-only Property Test

Purpose: To verify that properties marked as read-only in the EPICS are in fact read-only.

Test Concept: To each read-only (not writable and not conditionally writable) property in the EPICS, write the value of the property as read from the device and verify that an error is returned. Write another value that is within the acceptable range for the datatype and verify that an error is returned. If the property is a list and the IUT supports AddListElement, attempt to modify the property with AddListElement and verify that an error is returned. ~~If the IUT does not support the WriteProperty service, then this test shall be skipped.~~

*Configuration Requirements: If the IUT does not support the WriteProperty service, then this test shall be skipped.*

*Notes to Tester: The objects selected by the tester should include one instance of each supported object type. Where some instances of an object type differ in the set of supported properties, the allowable value ranges for a property, or the writability of a property, then one instance of each variant of that object type should be selected.*

*Notes to Tester: When modifying a property, it is expected that an Error Class of PROPERTY with an error code of WRITE\_ACCESS\_DENIED will be returned, but the IUT may instead return an error\_class of PROPERTY with an error\_code of VALUE\_OUT\_OF\_RANGE, or an error\_class of RESOURCES with an error\_code of NO\_SPACE\_TO\_WRITE\_PROPERTY.*

Test Steps:

1. REPEAT X = (a tester selected set of objects) DO {
  - REPEAT Y = (all read-only properties in object X) DO {
    - IF (the property is not an array) THEN
 

---

~~READ Z = X~~
  
 READ Z = (X), property Y
   
 TRANSMIT WriteProperty-Request,
 

'Object Identifier' =	X,
'Property Identifier' =	Y,
'Property Value' =	Z

  
 RECEIVE BACnet-Error-PDU,
 

'Error Class' =	PROPERTY,
'Error Code' =	WRITE_ACCESS_DENIED

  
  
 TRANSMIT WriteProperty-Request,
 

'Object Identifier' =	X,
'Property Identifier' =	Y,
'Property Value' =	(any value meeting the range requirements of 7.2.1 except Z)

  
 RECEIVE BACnet-Error-PDU,
 

'Error Class' =	PROPERTY,
'Error Code' =	WRITE_ACCESS_DENIED

  
  
 IF (the IUT supports AddListElement and the property is a list) THEN
      - TRANSMIT AddListElement-Request,
 

'Object Identifier' =	X,
'Property Identifier' =	Y,
'List of Elements' =	(any elements value meeting the range requirements of 7.2.1 excluding those in Z)

  
 RECEIVE BACnet-Error-PDU,
 

'Error Class' =	PROPERTY,
-----------------	-----------

```

        'Error Code' = WRITE_ACCESS_DENIED
    ELSE
        READ LEN = X, Array_Index = 0
        READ LEN = (X), Y, ARRAY INDEX = 0
        IF (LEN > 0) THEN
        READ Z = X, Array_Index = 1
        READ Z = (X), Y, ARRAY INDEX = 1
        TRANSMIT WriteProperty-Request,
            'Object Identifier' = X,
            'Property Identifier' = Y,
            'Property Value' = Z,
            'Property Array Index' = 1
        RECEIVE BACnet-Error-PDU,
            'Error Class' = PROPERTY,
            'Error Code' = WRITE_ACCESS_DENIED

        TRANSMIT WriteProperty-Request,
            'Object Identifier' = X,
            'Property Identifier' = Y,
            'Property Value' = (any value meeting the range requirements of 7.2.1 except Z)
            'Property Array Index' = 1
        RECEIVE BACnet-Error-PDU,
            'Error Class' = PROPERTY,
            'Error Code' = WRITE_ACCESS_DENIED

        IF (the IUT supports AddListElement and the property is an array of lists) THEN
            TRANSMIT AddListElement-Request,
                'Object Identifier' = X,
                'Property Identifier' = Y,
                'Property Array Index' = 1
                'List of Elements' = (any elements value meeting the range requirements of 7.2.1 excluding
                                    those in Z)
            RECEIVE BACnet-Error-PDU,
                'Error Class' = PROPERTY,
                'Error Code' = WRITE_ACCESS_DENIED
        ELSE
            TRANSMIT WriteProperty-Request,
                'Object Identifier' = X,
                'Property Identifier' = Y,
                'Property Value' = (any value meeting the range requirements of 7.2.1)
            RECEIVE BACnet-Error-PDU,
                'Error Class' = PROPERTY,
                'Error Code' = WRITE_ACCESS_DENIED
        }
    }

```

Notes to Tester: The objects selected by the tester should include one instance of each supported object type. Where some instances of an object type differ in the set of supported properties, the allowable value ranges for a property, or the writability of a property, then one instance of each variant of that object type should be selected.

Notes to Tester: When modifying a property, it is expected that an error class of PROPERTY with an error code of WRITE\_ACCESS\_DENIED will be returned, but the IUT may instead return an error\_class of PROPERTY with an error\_code of VALUE\_OUT\_OF\_RANGE, or an error\_class of RESOURCES with an error\_code of NO\_SPACE\_TO\_WRITE\_PROPERTY.

[Add new **Clause 7.2.X1**, p.24]

### 7.2.X1 Date Pattern Properties Test

Purpose: To verify that the property being tested accepts special date field values.

Test Concept: The property being tested, P1, is written with each of the special date field values to ensure that the property accepts them. A date, D1, is selected which is within the date range that the IUT will accept for the property. The value, written to the property, is the date D1 with one of its fields replaced with one of the date special values. If the property is a complex datatype, the other fields in the value shall be set within the range accepted by the IUT. The list of Specials comes from the Chapter 21 Application Types section on Date. The day-of-week field is redundant information and can be regenerated from the other fields. In order to not fail devices which always ensure this field is consistent with the other fields in the date value, the use of an unspecified day of week is always tested in conjunction with another pattern value.

Configuration Requirements: The IUT shall be configured with a Calendar object that contains a Date\_List with a single BACnetCalendarEntry in the form of a Date. If Date\_List property cannot be configured with a BACnetCalendarEntry in the form of a Date, then this test shall be skipped.

Notes to Tester: If P1 is an array, then an array index shall be provided in the WRITE and VERIFY operations.

Test Steps:

1. IF (Protocol\_Revision is not present or Protocol\_Revision < 4) THEN  
     Specials = (year unspecified, month unspecified, day of month unspecified)  
   ELSE IF (Protocol\_Revision >= 4 and Protocol\_Revision < 10) THEN  
     Specials = (year unspecified, month unspecified, day of month unspecified, odd months, even months, last day of month)  
   ELSE  
     Specials = (year unspecified, month unspecified, day of month unspecified, odd months, even months, last day of month, even days, odd days)
2. REPEAT SV = (each value in Specials) DO {  
     IF (SV <> day of week unspecified) THEN  
       V1 = D1 updated with the value SV  
     ELSE  
       V1 = D1 updated with the value SV and any other value from Specials  
     WRITE P1 = (V1)  
     VERIFY P1 = (V1)  
   }

[Add new **Clause 7.2.X2**, p.24]

### 7.2.X2 Time Pattern Properties Test

Purpose: To verify that the property being tested accepts special time field values.

Test Concept: The property being tested, P1, is written with each of the special time field values to ensure that the property accepts them. A time, T1, is selected which is within the time range that the IUT will accept for the property. The value, written to the property, is the time T1 with one of its fields replaced with one of the time special values. If the property is a complex datatype, the other fields in the value shall be set within the range accepted by the IUT.

Notes to Tester: If P1 is an array, then an array index shall be provided in the WRITE and VERIFY operations.

Test Steps:

1. REPEAT SV = (hour unspecified, minute unspecified, second unspecified, hundredths unspecified) DO {  
WRITE P1 = (T1 updated with the value SV)  
VERIFY P1 = (T1 updated with the value SV)  
}

[Add new **Clause 7.2.X3**, p.24]

### 7.2.X3 DateTime Pattern Properties Test

Purpose: To verify that the property being tested accepts special date and time field values.

Test Concept: The property being tested, P1, is written with each of the special date and time field values to ensure that the property accepts them. A date, D1, is selected which is within the date range that the IUT will accept for the property. A time, T1, is selected which is within the time range that the IUT will accept for the property. The value, written to the property, is the date D1 and time T1 with one of its fields replaced with one of the date or time special values. If the property is a complex datatype which contains the BACnetDateTime, the other fields in the value shall be set within the range accepted by the IUT. The list of DateSpecials comes from the Chapter 21 Application Types section on Date, and the list of TimeSpecials comes from the Chapter 21 Application Types section on Time.

Notes to Tester: If P1 is an array, then an array index shall be provided in the WRITE and VERIFY operations.

Test Steps:

1. IF (Protocol\_Revision is not present or Protocol\_Revision < 4) THEN  
DateSpecials = (year unspecified, month unspecified, day of month unspecified, day of week unspecified)  
ELSE IF (Protocol\_Revision >= 4 and Protocol\_Revision < 10) THEN  
DateSpecials = (year unspecified, month unspecified, day of month unspecified, day of week unspecified, odd months, even months, last day of month)  
ELSE  
DateSpecials = (year unspecified, month unspecified, day of month unspecified, day of week unspecified, odd months, even months, last day of month, even days, odd days)
2. TimeSpecials = (hour unspecified, minute unspecified, second unspecified, hundredths unspecified)
3. REPEAT SV = (each value in DateSpecials + TimeSpecials) DO {  
WRITE P1 = (D1+T1 updated with the value SV)  
VERIFY P1 = (D1+T1 updated with the value SV)  
}

[Add new **Clause 7.2.X4**, p.24]

### 7.2.X4 Date Non-Pattern Properties Test

Purpose: To verify that the property being tested does not accept special date field values.

Test Concept: The property being tested, P1, is written with each of the special date field values to ensure that the property does not accept them. A date is selected which is within the date range that the IUT will accept for the property. The value, V1, written to the property, is the date D1 with one of its fields replaced with one of the date special values. If the property is a complex datatype, the other fields in the value shall be set within the range accepted by the IUT. This test shall only be applied to devices claiming Protocol\_Revision 11 or higher.

Notes to Tester: If P1 is an array, then an array index shall be provided in the TRANSMIT portion of step 1.

Test Steps:

1. REPEAT SV = (year unspecified, month unspecified, day of month unspecified, day of week unspecified, odd months, even months, last day of month, even days, odd days) DO {  
TRANSMIT WriteProperty-Request  
'Object Identifier' = O1,  
'Property Identifier' = P1,  
'Property Value' = (V1 updated with the special value SV)

```
RECEIVE BACnet-Error-PDU
  'Error Class' =      PROPERTY,
  'Error Code' =      VALUE_OUT_OF_RANGE
```

[Add new **Clause 7.2.X5**, p.24]

### 7.2.X5 Time Non-Pattern Properties Test

Purpose: To verify that the property being tested does not accept special time field values.

Test Concept: The property being tested, P1, is written with each of the special time field values to ensure that the property does not accept them. A time is selected which is within the time range that the IUT will accept for the property. The value, V1, written to the property, is the time T1 with one of its fields replaced with one of the time special values. If the property is a complex datatype, the other fields in the value shall be set within the range accepted by the IUT. This test shall only be applied to devices claiming Protocol\_Revision 11 or higher.

Notes to Tester: If P1 is an array, then an array index shall be provided in the TRANSMIT portion of step 1.

Test Steps:

1. REPEAT SV = (hour unspecified, minute unspecified, second unspecified, hundredths unspecified)
 

```
TRANSMIT WriteProperty-Request
  'Object Identifier' = O1,
  'Property Identifier' = P1,
  'Property Value' = (V1 updated with the special value SV)
RECEIVE BACnet-Error-PDU
  'Error Class' =      PROPERTY,
  'Error Code' =      VALUE_OUT_OF_RANGE
```

[Add new **Clause 7.2.X6**, p.24]

### 7.2.X6 DateTime Non-Pattern Properties Test

Purpose: To verify that the property being tested does not accept special date field values.

Test Concept: The property being tested, P1, is written with each of the special datetime field values to ensure that the property does not accept them. A datetime DT1 is selected which is within the range that the IUT will accept for the property. The value, V1, written to the property, is the datetime DT1 with one of its fields replaced with one of the date or time special values. If the property is a complex datatype, the other fields in the value shall be set within the range accepted by the IUT. This test shall only be applied to devices claiming Protocol\_Revision 11 or higher.

Notes to Tester: If P1 is an array, then an array index shall be provided in the TRANSMIT portion of step 1.

Test Steps:

1. REPEAT SV = (year unspecified, month unspecified, day of month unspecified, odd months, even months, last day of month, even days, odd days, hour unspecified, minute unspecified, second unspecified, hundredths unspecified) DO {
 

```
TRANSMIT WriteProperty-Request
  'Object Identifier' = O1,
  'Property Identifier' = P1,
  'Property Value' = (DT1 updated with the special value SV)
RECEIVE BACnet-Error-PDU
  'Error Class' =      PROPERTY,
  'Error Code' =      VALUE_OUT_OF_RANGE
```

[Add new **Clause 7.2.X7**, p.24]

## 7.2.X7 BACnetDateRange Non-Pattern Properties Test

Purpose: To verify that the property being tested does not accept special date field values.

Test Concept: A BACnetDateRange property, or property that is a complex datatype containing a BACnetDateRange,  $P_1$  is written with each of the special date field values to ensure that the property does not accept them. Each half of the dateRange  $DR_1$  is selected so it is within the range that the IUT will accept for the property. The value,  $V_1$ , written to the property, is the dateRange  $DR_1$  with one of its fields replaced with one of the date special values. If the property is a complex datatype such as a BACnetCalendarEntry, the other fields in the value shall be set within the range accepted by the IUT.

Configuration Requirements: This test shall only be applied to devices claiming Protocol\_Revision 11 or higher. The IUT shall be configured with a Calendar object that contains a Date\_List with a single BACnetCalendarEntry in the form of a BACnetDateRange. If Date\_List property cannot be configured with a BACnetCalendarEntry in the form of a BACnetDateRange, then this test shall be skipped.

Notes to Tester: If  $P_1$  is an array, then an array index shall be provided in the TRANSMIT portion of step 1.

Test Steps:

1. REPEAT SV = (year unspecified, month unspecified, day of month unspecified, odd months, even months, last day of month, even days, odd days) DO {
  - TRANSMIT WriteProperty-Request
    - 'Object Identifier' =  $O_1$ ,
    - 'Property Identifier' =  $P_1$ ,
    - 'Property Value' = ( $DR_1$  with startDate updated with the special value SV)
  - RECEIVE BACnet-Error-PDU
    - 'Error Class' = PROPERTY,
    - 'Error Code' = VALUE\_OUT\_OF\_RANGE
  - TRANSMIT WriteProperty-Request
    - 'Object Identifier' =  $O_1$ ,
    - 'Property Identifier' =  $P_1$ ,
    - 'Property Value' = ( $DR_1$  with endDate updated with the special value SV)
  - Receive BACnet-Error-PDU
    - 'Error Class' = PROPERTY,
    - 'Error Code' = VALUE\_OUT\_OF\_RANGE

[Add new **Clause 7.2.X8**, p.24]

## 7.2.X8 BACnetDateRange Open-Ended Pattern Properties Test

Purpose: To verify that the property being tested accepts a fully unspecified date in either or both halves of the value.

Test Concept: A BACnetDateRange property, or property that has a complex datatype containing a BACnetDateRange,  $P_1$  is written with a fully unspecified date in either or both halves to ensure that the property accepts them.  $DR_1$  is selected which is within the date range that the IUT will accept for the property. The value, written to the property, is the dateRange  $DR_1$  replaced with a fully unspecified date in either or both startDate and endDate. If the property is a complex datatype, the other fields in the value shall be set within the range accepted by the IUT.

Configuration Requirements: This test shall only be applied to devices claiming Protocol\_Revision 11 or higher. The IUT shall be configured with a Calendar object that contains a Date\_List with a single BACnetCalendarEntry in the form of a BACnetDateRange. If Date\_List property cannot be configured with a BACnetCalendarEntry in the form of a BACnetDateRange, then this test shall be skipped.

Notes to Tester: If  $P_1$  is an array, then an array index shall be provided in the WRITES and VERIFYs.

Test Steps:

1. WRITE  $P_1$  = ( $DR_1$  updated with a fully unspecified date in startDate)
2. VERIFY  $P_1$  = (the value written)

3. WRITE P<sub>1</sub> = (DR<sub>1</sub> updated with a fully unspecified date in endDate)
4. VERIFY P<sub>1</sub> = (the value written)
5. WRITE P<sub>1</sub> = (DR<sub>1</sub> updated with a fully unspecified date in both startDate and endDate)
6. VERIFY P<sub>1</sub> = (the value written)

## 7.3 Object Functionality Tests

### 7.3.1 Property Tests

[Change and renumber **Clause 7.3.1.1**, p. 24]

#### 7.3.1.1.X1 ~~7.3.1.1~~ Out\_Of\_Service, Status\_Flags, and Reliability Test

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

~~BACnet Reference Clauses: 12.1.7, 12.1.9, 12.1.10, 12.2.7, 12.2.9, 12.2.10, 12.3.7, 12.3.9, 12.3.10, 12.4.6, 12.4.8, 12.4.9, 12.6.7, 12.6.9, 12.6.10, 12.7.7, 12.7.9, 12.7.10, 12.8.6, 12.8.8, 12.8.9, 12.15.8, 12.15.10, 12.15.11, 12.16.8, 12.16.10, 12.16.11, 12.17.6, 12.17.8, 12.17.9, 12.18.7, 12.18.9, 12.18.10, 12.19.7, 12.19.9, 12.19.10, 12.20.6, 12.20.8, 12.20.9, 12.23.7, 12.23.9, and 12.23.10.~~

~~Purpose: This test case verifies that To verify that Present\_Value is writable when Out\_Of\_Service is TRUE and. It also that the interrelationship between the Out\_Of\_Service, Status\_Flags, and Reliability properties. If the PICS indicates that the Out\_Of\_Service property of the object under test is not writable, and if the value of the property cannot be changed by other means, then this test shall be omitted. This test applies to Accumulator, Analog Input, Analog Output, Analog Value, Binary Input, Binary Output, Binary Value, Life Safety Point, Life Safety Zone, Multi state Input, Multi state Output, Multi state Value, Loop and Pulse Converter objects.~~

~~Test Concept: The IUT will select one instance of each appropriate object type and test it as described. If the Reliability property is not supported then step 4 shall be omitted. The value of the Out\_Of\_Service property is set to TRUE, and the Present\_Value property is tested to be writable. The value of the Status\_Flags property is validated and, if present, the value of the Reliability property is also validated. The value of the Status\_Flags property, SF1, and, if present, the Reliability property, R1, are checked to ensure they return to there initial values when the value of the Out\_Of\_Service property is set to FALSE.~~

~~Configuration Requirements: If the selected object is commandable, the values of the entries in the Priority\_Array above the selected priority, PTY1, shall be NULL.~~

Test Steps:

1. SF1 = Status\_Flags
2. R1 = Reliability
34. IF (Out\_Of\_Service is writable) THEN  
    WRITE Out\_Of\_Service = TRUE  
ELSE  
    MAKE (Out\_Of\_Service TRUE)
42. VERIFY Out\_Of\_Service = TRUE
53. VERIFY Status\_Flags = (?, FALSE?, ?, TRUE)
64. REPEAT X = (all values meeting the functional range requirements of 7.2.1) DO {  
    WRITE Present\_Value, PTY1 = X  
    VERIFY Present\_Value = X  
}
75. IF (Reliability is present and writable) THEN  
    REPEAT X = (all values of the Reliability enumeration appropriate to the object type except NO\_FAULT\_DETECTED) DO {  
        WRITE Reliability = X  
        VERIFY Reliability = X  
        VERIFY Status\_Flags = (?, TRUE, ?, TRUE)

```

        WRITE Reliability = NO_FAULT_DETECTED
        VERIFY Reliability = NO_FAULT_DETECTED
        VERIFY Status_Flags = (?, FALSE, ?, TRUE)
    }
86. IF (Out_Of_Service is writable) THEN
    WRITE Out_Of_Service = FALSE
ELSE
    MAKE (Out_Of_Service FALSE)
97. VERIFY Out_Of_Service = FALSE
108. VERIFY Status_Flags = (?, ?, ?, FALSE) SF1
11. VERIFY Reliability = R1

```

~~Notes to Tester: If the object being tested is commandable and there is an internal process writing to the Present\_Value property, then each WriteProperty request shall contain a priority sufficient to override the internal process. After step 4 the priority array slot shall be relinquished.~~

[Add new **Clause 7.3.1.1.X2**, p. 24]

#### **7.3.1.1.X2 Out\_Of\_Service for Commandable Value Objects Test**

Purpose: To verify that Present\_Value is no longer updated by software local to the IUT when Out\_Of\_Service is TRUE.

Test Concept: Select an object whose Present\_Value is being modified by software local to the IUT at Priority PTY1. The value of the Out\_Of\_Service property is set to TRUE, the Present\_Value property is written at PTY1, and the Present\_value is checked to ensure the Present-Value is no longer being modified by software local to the IUT.

Configuration Requirements: The values of the entries in the Priority\_Array above PTY1 shall be NULL.

Test Steps:

1. MAKE (Present\_Value = PV1, any valid value, using software local to the IUT)
2. IF (Out\_Of\_Service is writable) THEN
  - WRITE Out\_Of\_Service = TRUE
  - ELSE
    - MAKE (Out\_Of\_Service TRUE)
3. VERIFY Present\_Value = PV1
4. WRITE Present\_Value, PTY1 = PV2, any valid value other than PV1
5. MAKE (Present\_Value = PV3, any valid value other than PV2, using software local to the IUT)
6. VERIFY Present\_Value = PV2

[Add new **Clause 7.3.1.1.X3**, p. 24]

#### **7.3.1.1.X3 Out\_Of\_Service, Status\_Flags, and Reliability Test for Objects without Present\_Value**

Purpose: This test verifies the interrelationship between the Out\_Of\_Service, Status\_Flags, and Reliability properties. If the PICS indicates that the Out\_Of\_Service property of the object under test is not writable, and if the value of the property cannot be changed by other means, then this test shall be omitted. This test applies to objects that do not contain Present\_Value.

Test Concept: Write to and verify the interrelationship between the Out\_Of\_Service, Status\_Flags, and Reliability properties of an object which does not contain Present\_Value.

Configuration Requirements: The selected object is configured such that its Reliability is NO\_FAULT\_DETECTED before execution of this test.

Test Steps:

1. IF (Out\_Of\_Service is writable) THEN



```

WRITE Out_Of_Service = TRUE
ELSE
    MAKE (Out_Of_Service = TRUE)
2. VERIFY Out_Of_Service = TRUE
3. VERIFY Status_Flags = (?, FALSE, ?, TRUE)
4. IF (Reliability is present and writable) THEN
    REPEAT X = (all values of the Reliability enumeration appropriate to the object type except
        NO_FAULT_DETECTED) DO {
        WRITE Reliability = X
        VERIFY Reliability = X
        VERIFY Status_Flags = (?, TRUE, ?, TRUE)
        WRITE Reliability = NO_FAULT_DETECTED
        VERIFY Reliability = NO_FAULT_DETECTED
        VERIFY Status_Flags = (?, FALSE, ?, TRUE)
    }
5. CHECK (functionality that should stop or be disabled is. For example, with a Network Port object, all communication
    of the protocol modeled by the object, through that port is disabled)
6. IF (Out_Of_Service is writable) THEN
    WRITE Out_Of_Service = FALSE
ELSE
    MAKE (Out_Of_Service = FALSE)
7. VERIFY Out_Of_Service = FALSE
8. VERIFY Status_Flags = ( ?, ?, ?, FALSE)

```

[Add new **Clause 7.3.1.6**, p. 29]

### 7.3.1.6 Minimum On/Off Time Tests

[Change and renumber **Clause 7.3.1.6**, p. 29]

[Reviewer Note: The test was re-written to remove the dependence on the presence of the Minimum\_Off\_Time property. This test was renumbered from 7.3.1.6 to 7.3.1.6.1.]

#### 7.3.1.6.1 Override of Minimum Time

~~Dependencies: ReadProperty Service Execution Tests, 9.15; WriteProperty Service Execution Tests, 9.19.~~

~~BACnet Reference Clause: 19.~~

Purpose: To verify that higher priority commands override minimum on or off times. If neither minimum on time or minimum off time is supported this test shall be omitted. This test applies to Binary Output and *commandable* Binary Value objects.

Test Concept: The initial Present\_Value of the object tested is set to INACTIVE and it is controlled at a priority numerically greater (lower priority) than 6. The object has been in this state long enough for any minimum off and/or minimum on time to have expired. The Present\_Value is written to with a value of ACTIVE at priority 7. The value of slot 6 of the Priority\_Array is monitored to verify that it contains the value ACTIVE. Before the minimum on time expires the Present\_Value is written to with a value of INACTIVE and a priority numerically lower (higher priority) than 6. This overrides the minimum on time and immediately initiates the minimum off time algorithm.

Configuration Requirements: The object to be tested shall be configured such that all slots in the Priority\_Array numerically less than 7 have a value of NULL and no internal algorithms are issuing commands to this object at a priority numerically lesser (higher priority) than the priority that is currently controlling Present\_Value. *Minimum\_On\_Time must be configured with a large enough value to allow execution of all test steps before it expires.*

Notes to Tester: *If minimum on time is not supported but minimum off time is supported, this test should be conducted by using INACTIVE in steps 1, 2, 3, and 6 through 3 and ACTIVE in steps 4 through 76 and 5, and by using the Minimum\_Off\_Time in Step 4.*

#### Test Steps:

1. WRITE Present\_Value = ACTIVE, PRIORITY = 7
2. VERIFY Present\_Value = ACTIVE
3. VERIFY Priority\_Array = ACTIVE, ARRAY INDEX = 6
4. BEFORE Minimum\_On\_Time  
WRITE Present\_Value = INACTIVE, PRIORITY = (any value numerically lower than 6 (higher priority))
5. VERIFY Present\_Value = INACTIVE
- ~~6. VERIFY Priority\_Array = INACTIVE, PRIORITY = 6~~
6. VERIFY Priority\_Array <> ACTIVE, ARRAY INDEX = 6

Notes to Tester: If minimum on time is not supported but minimum off time is supported, this test should be conducted by using INACTIVE in steps 1 through 3 and ACTIVE in steps 4 through 6, and by using Minimum\_Off\_Time in step 4.

[Add new **Clause 7.3.1.6.2**, p. 29]

#### 7.3.1.6.2 Minimum Off Time - Writing at priorities numerically greater than 6

Purpose: To verify that commands written at a lower priority than 6 will not affect Present\_Value while Minimum\_Off\_Time is in effect.

Test Concept: The initial Present\_Value of the object tested is set to ACTIVE, and it is controlled by the Relinquish\_Default value or at a priority numerically greater (lower priority) than P9 (P9 > 7). The object has been in this state long enough for any minimum on time to have expired. The Present\_Value of the object is set to INACTIVE at a priority P9. Before Minimum\_Off\_Time expires, Present\_Value is written with values of ACTIVE and NULL at Priorities P9 and P7, where P7 is a priority between P9 and 6. The Priority\_Array is monitored to verify that it contains the appropriate values, and Present\_Value is monitored to verify that it does not change before Minimum\_Off\_Time expires.

Test Step(s) →	Start of Test	1-3	4-6	7-10	11-15	16
Present_Value	Active	Inactive	Inactive	Inactive	Inactive	Active
PA_Index = 6	Null	Inactive	Inactive	Inactive	Inactive	<>Inactive
PA_Index = P7	Null	Null	Null	<b>Active</b>	Active	Active
PA_Index = P9	Null	<b>Inactive</b>	<b>Null</b>	Null	<b>Active</b>	Active
Relinquish_Default	Active	Active	Active	Active	Active	Active

Note: Bold font indicates the change invoked by write operation

Minimum\_Off\_Time

End of Test

Configuration Requirements: The object to be tested shall be configured such that all slots in the Priority\_Array from P9 and higher (numerically lesser) have a value of NULL and no internal algorithms are issuing commands to this object. Minimum\_Off\_Time must be configured with a large enough value to allow execution of test steps 1-14 before it expires.

#### Test Steps:

1. WRITE Present\_Value = INACTIVE, PRIORITY = P9
2. VERIFY Present\_Value = INACTIVE
3. VERIFY Priority\_Array = INACTIVE, ARRAY INDEX = 6
4. WRITE Present\_Value = NULL, PRIORITY = P9
5. VERIFY Present\_Value = INACTIVE
6. VERIFY Priority\_Array = INACTIVE, ARRAY INDEX = 6

- ...(Steps 4-6:Check that a NULL value at P9 does not affect ARRAY INDEX = 6 or PV)
7. WRITE Present\_Value = ACTIVE, PRIORITY = P7 (6 < P7 < P9)
  8. VERIFY Present\_Value = INACTIVE
  9. VERIFY Priority\_Array = ACTIVE, ARRAY INDEX = P7
  10. VERIFY Priority\_Array = INACTIVE, ARRAY INDEX = 6
- ...(Steps 7-10:Check that an ACTIVE value at P7 does not affect ARRAY INDEX = 6 or PV)
11. WRITE Present\_Value = ACTIVE, PRIORITY = P9
  12. VERIFY Present\_Value = INACTIVE
  13. VERIFY Priority\_Array = ACTIVE, ARRAY INDEX = P9
  14. VERIFY Priority\_Array = INACTIVE, ARRAY INDEX = 6
- ...(Steps 11-14:Check that an ACTIVE value at P9 does not affect ARRAY INDEX = 6 or PV)
15. WAIT (Minimum\_Off\_Time + Internal Processing Fail Time)
  16. VERIFY Present\_Value = ACTIVE

[Add new **Clause 7.3.1.6.3**, p. 29]

### 7.3.1.6.3 Minimum On Time - Writing at priorities numerically greater than 6

Purpose: To verify that commands written at a lower priority than 6 will not affect Present\_Value while Minimum\_On\_Time is in effect.

Test Concept: The initial Present\_Value of the object tested is set to INACTIVE, and it is controlled by the Relinquish\_Default value or at a priority numerically greater (lower priority) than P9 (P9 > 7). The object has been in this state long enough for any minimum off time to have expired. The Present\_Value of the object tested is set to ACTIVE at a priority P9. Before Minimum\_On\_Time expires, Present\_Value is written with values of ACTIVE and NULL at Priorities P9 and P7, where P7 is a priority between P9 and 6. The Priority\_Array is monitored to verify that it contains the appropriate values, and Present\_Value is monitored to verify that it does not change before Minimum\_On\_Time expires.

Test Step(s) →	Start of Test	1-3	4-6	7-10	11-15	16
Present_Value	Inactive	Active	Active	Active	Active	Inactive
PA_Index = 6	Null	Active	Active	Active	Active	◇Active
PA_Index = P7	Null	Null	Null	<b>Inactive</b>	Inactive	Inactive
PA_Index = P9	Null	<b>Active</b>	<b>Null</b>	Null	<b>Inactive</b>	Inactive
Relinquish_Default	Inactive	Inactive	Inactive	Inactive	Inactive	Inactive

Note: Bold font indicates the change invoked by write operation

Minimum\_On\_Time

End of Test

Configuration Requirements: The object to be tested shall be configured such that all slots from P9 and higher (numerically lesser) in the Priority\_Array have a value of NULL and no internal algorithms are issuing commands to this object. Minimum\_On\_Time must be configured with a large enough value to allow execution of test steps 1-14 before it expires.

Test Steps:

1. WRITE Present\_Value = ACTIVE, PRIORITY = P9
2. VERIFY Present\_Value = ACTIVE
3. VERIFY Priority\_Array = ACTIVE, ARRAY INDEX = 6
4. WRITE Present\_Value = NULL, PRIORITY = P9
5. VERIFY Present\_Value = ACTIVE
6. VERIFY Priority\_Array = ACTIVE, ARRAY INDEX = 6

- ...(Steps 4-6:Check that a NULL value at P9 does not affect ARRAY INDEX = 6 or PV)
7. WRITE Present\_Value = INACTIVE, PRIORITY = P7 (6 < P7 < P9)
  8. VERIFY Present\_Value = ACTIVE
  9. VERIFY Priority\_Array = INACTIVE, ARRAY INDEX = P7
  10. VERIFY Priority\_Array = ACTIVE, ARRAY INDEX = 6
- ...(Steps 7-10:Check that an INACTIVE value at P7 does not affect ARRAY INDEX = 6 or PV)
11. WRITE Present\_Value = INACTIVE, PRIORITY = P9
  12. VERIFY Present\_Value = ACTIVE
  13. VERIFY Priority\_Array = INACTIVE, ARRAY INDEX = P9
  14. VERIFY Priority\_Array = ACTIVE, ARRAY INDEX = 6
- ...(Steps 11-14:Check that an INACTIVE value at P9 does not affect ARRAY INDEX = 6 or PV)
15. WAIT (Minimum\_On\_Time + Internal Processing Fail Time)
  16. VERIFY Present\_Value = INACTIVE

[Add new **Clause 7.3.1.6.4**, p. 29]

### 7.3.1.6.4 Minimum Off Time - Writing at priorities numerically lesser than 6

Purpose: To verify that the value at priority 6 contains the appropriate value when commands are written at a higher priority and minimum off time is in effect.

Test Concept: The initial Present\_Value of the object tested is set to ACTIVE, and it is controlled by the Relinquish\_Default value or at a priority numerically greater (lower priority) than 6. The object has been in this state long enough for any minimum on time to have expired. The Present\_Value of the object tested is set to INACTIVE at a priority P5 (P5 < 6). Before Minimum\_Off\_Time expires, Present\_Value is written with values of NULL and ACTIVE, and the Present\_Value and Priority\_Array properties are observed for correct behavior.

Test Steps →	Start of Test	1-3	4-7	8-11
Present_Value	Active	Inactive	Inactive	Active
PA_Index = P5	Null	<b>Inactive</b>	<b>Null</b>	<b>Active</b>
PA_Index = 6	Null	Inactive	Inactive	◇Inactive
Relinquish_Default	Active	Active	Active	Active

Note: Bold font indicates the change invoked by write operation

Minimum Off Time

Configuration Requirements: The object to be tested shall be configured such that all slots in the Priority\_Array higher (numerically lesser) than 6 have a value of NULL and no internal algorithms are issuing commands to this object. Minimum\_Off\_Time must be configured with a large enough value to allow execution of the test steps before it expires.

Test Steps:

1. WRITE Present\_Value = INACTIVE, PRIORITY = P5
  2. VERIFY Present\_Value = INACTIVE
  3. VERIFY Priority\_Array = INACTIVE, ARRAY INDEX = 6
  4. WRITE Present\_Value = NULL, PRIORITY = P5
  5. VERIFY Present\_Value = INACTIVE
  6. VERIFY Priority\_Array = INACTIVE, ARRAY INDEX = 6
  7. VERIFY Priority\_Array = NULL, ARRAY INDEX = P5
- ...(Steps 4-7:Check that a NULL value at P5 will NOT change ARRAY INDEX = 6 or PV)
8. WRITE Present\_Value = ACTIVE, PRIORITY = P5
  9. VERIFY Present\_Value = ACTIVE

10. VERIFY Priority\_Array = ACTIVE, ARRAY INDEX = P5
11. VERIFY Priority\_Array  $\neq$  INACTIVE, ARRAY INDEX = 6
- ...(Steps 8-11:Check that an ACTIVE value at P5 will change ARRAY INDEX = 6 and PV)

[Add new **Clause 7.3.1.6.5**, p. 29]

### 7.3.1.6.5 Minimum On Time - Writing at priorities numerically lesser than 6

Purpose: To verify that the value at priority 6 contains the appropriate value when commands are written at a higher priority and minimum on time is in effect.

Test Concept: The initial Present\_Value of the object tested is set to INACTIVE, and it is controlled by the Relinquish\_Default value or at a priority numerically greater (lower priority) than 6. The object has been in this state long enough for any minimum off time to have expired. The Present\_Value of the object tested is set to ACTIVE at a priority P5 ( $P5 < 6$ ). Before Minimum\_On\_Time expires, Present\_Value is written with values of NULL and INACTIVE, and the Present\_Value and Priority\_Array properties are observed for correct behavior.

Test Steps →	Start of Test	1-3	4-7	8-11
Present_Value	Inactive	Active	Active	Inactive
PA_Index = P5	Null	<b>Active</b>	<b>Null</b>	<b>Inactive</b>
PA_Index = 6	Null	Active	Active	$\neq$ Active
Relinquish_Default	Inactive	Inactive	Inactive	Inactive

Note: Bold font indicates the change invoked by write operation

Minimum\_On\_Time

Configuration Requirements: The object to be tested shall be configured such that all slots in the Priority\_Array higher (numerically lesser) than 6 have a value of NULL and no internal algorithms are issuing commands to this object. Minimum\_On\_Time must be configured with a large enough value to allow execution of the test steps before it expires.

Test Steps:

1. WRITE Present\_Value = ACTIVE, PRIORITY = P5
2. VERIFY Present\_Value = ACTIVE
3. VERIFY Priority\_Array = ACTIVE, ARRAY INDEX = 6
4. WRITE Present\_Value = NULL, PRIORITY = P5
5. VERIFY Present\_Value = ACTIVE
6. VERIFY Priority\_Array = ACTIVE, ARRAY INDEX = 6
7. VERIFY Priority\_Array = NULL, ARRAY INDEX = P5
- ...(Steps 4-7:Check that a NULL value at P5 will NOT change ARRAY INDEX = 6 or PV)
8. WRITE Present\_Value = INACTIVE, PRIORITY = P5
9. VERIFY Present\_Value = INACTIVE
10. VERIFY Priority\_Array = INACTIVE, ARRAY INDEX = P5
11. VERIFY Priority\_Array  $\neq$  ACTIVE, ARRAY INDEX = 6
- ...(Steps 8-11:Check that an INACTIVE value at P5 will change ARRAY INDEX = 6 and PV)

[Add new **Clause 7.3.1.6.6**, p. 29]

### 7.3.1.6.6 Minimum\_Off\_Time - Clock is not affected by additional write operations

Purpose: To verify that the Minimum\_Off\_Time timer is not affected by subsequent write operations that do not cause present-value to change.

Test Concept: The initial Present\_Value of the object being tested is set to ACTIVE, and the value at slot 6 in the priority-array has a value of NULL. Present\_Value of the object is written to INACTIVE at priority P8, such that present-value and slot 6 in the priority-array change to INACTIVE. At time T1, which occurs before minimum off time expires, another write request, at priority P9, with a value of INACTIVE, is executed by the device. After minimum off time expires but before T1 + Minimum\_Off\_Time, slot 6 in the priority-array is checked to verify that it returned to NULL and was not affected by the second request.

Test Step(s) →	1-2	3-4	5-8	9
Present_Value	Active	Inactive	Inactive	Inactive
PA_Index = P6	Null	Inactive	Inactive	Null
PA_Index = P <del>X</del> 8	Null	<b>Inactive</b>	Inactive	Inactive
PA_Index = P <del>Y</del> 9	Null	Null	<b>Inactive</b>	Inactive

Note: Bold font indicates the change invoked by write operation

Minimum Off Time

Configuration Requirements: The object to be tested shall be configured such that the present value is set to ACTIVE and slot 6 in the Priority\_Array has a value of NULL. P8 and P9 are selected such that they are higher (lesser numerically) than any other commanding priority.

Notes to Tester: P8 and P9 may assume any value in the Priority\_Array (except 6) and may be equal.

Test Steps:

1. VERIFY Present\_Value = ACTIVE
2. VERIFY Priority\_Array = NULL, ARRAY INDEX = 6
3. WRITE Present\_Value = INACTIVE, PRIORITY = P8
4. VERIFY Priority\_Array = INACTIVE, ARRAY INDEX = 6
- ...(Execute step 5 at time T1)
5. WRITE Present\_Value = INACTIVE, PRIORITY = P~~Y~~9
- ...(Execute steps 6 and 7 before Minimum\_Off\_Time expires)
6. VERIFY Present\_Value = INACTIVE
7. VERIFY Priority\_Array = INACTIVE, ARRAY INDEX = 6
8. WAIT for Minimum\_Off\_Time to expire
- ...(Execute step 9 before T1 + Minimum\_Off\_Time)
9. VERIFY Priority\_Array = NULL, ARRAY INDEX = 6

[Add new **Clause 7.3.1.6.7**, p. 29]

### 7.3.1.6.7 Minimum\_On\_Time - Clock is not affected by additional write operations

Purpose: To verify that the Minimum\_On\_Time timer is not affected by subsequent write operations that do not cause present-value to change.

Test Concept: The initial Present\_Value of the object being tested is set to INACTIVE, and the value at slot 6 in the priority-array has a value of NULL. Present\_Value of the object is written to ACTIVE, at priority P8, such that present-value and slot 6 in the priority-array change to ACTIVE. At time T1, which occurs before minimum on time expires, another write request, at priority P9, with a value of ACTIVE, is executed by the device. After minimum on time expires but before T1 + Minimum\_On\_Time, slot 6 in the priority-array is checked to verify that it returned to NULL and was not affected by the second request.

Test Step(s) →	1-2	3-4	5-8	9
Present_Value	Inactive	Active	Active	Active
PA_Index = P6	Null	Active	Active	Null
PA_Index = P8	Null	<b>Active</b>	Active	Active
PA_Index = P9	Null	Null	<b>Active</b>	Active

Note: Bold font indicates the change invoked by write operation

Minimum On Time

Configuration Requirements: The object to be tested shall be configured such that the present value is set to INACTIVE, and slot 6 in the Priority\_Array has a value of NULL. P8 and P9 are selected such that they are higher (lesser numerically) than any other commanding priority.

Notes to Tester: P8 and P9 may assume any value in the Priority\_Array (except 6) and may be equal.

Test Steps:

1. VERIFY Present\_Value = INACTIVE
2. VERIFY Priority\_Array = NULL, ARRAY INDEX = 6
3. WRITE Present\_Value = ACTIVE, PRIORITY = P8
4. VERIFY Priority\_Array = ACTIVE, ARRAY INDEX = 6
- ...(Execute step 5 at time T1)
5. WRITE Present\_Value = ACTIVE, PRIORITY = P9
- ...(Execute steps 6 and 7 before Minimum\_On\_Time expires)
6. VERIFY Present\_Value = ACTIVE
7. VERIFY Priority\_Array = ACTIVE, ARRAY INDEX = 6
8. WAIT for Minimum\_On\_Time to expire
- ...(Execute step 9 before T1 + Minimum\_On\_Time)
9. VERIFY Priority\_Array = NULL, ARRAY INDEX = 6

[Add new **Clause 7.3.1.6.8**, p. 29]

### 7.3.1.6.8 Ensuring Minimum\_Off\_Time starts at transition to INACTIVE

Purpose: To verify that Minimum\_Off\_Time does not start immediately after a write operation while Minimum\_On\_Time is in effect and present-value is ACTIVE.

Test Concept: The initial Present\_Value of the object being tested is set to INACTIVE, and the value at slot 6 in the priority-array has a value of NULL. Present\_Value of the object is written to ACTIVE at P9, where P9 is a priority between 7 and 16, such that present-value and slot 6 in the priority-array change to ACTIVE. Before Minimum\_On\_Time expires, Present\_Value is written to INACTIVE at P7, where P7 is a priority between 7 and P9, such that Present\_Value would change if Minimum\_On\_Time were not in effect. Slot 6 in the priority-array is monitored at specific time intervals to ensure that it contains the appropriate value. Time references T1 and T2 are defined for this test as follows:

T1 = the time when the INACTIVE request is executed by the device + Minimum\_Off\_Time

T2 = the time when the ACTIVE request is executed by the device + Minimum\_On\_Time + Minimum\_Off\_Time

Test Steps →	1-2	3-5	6-9	10-11	12-13	14-15
Present_Value	Inactive	Active	Active	Inactive	Inactive	Inactive
PA_Index = 6	Null	Active	Active	Inactive	Inactive	Null
PA_Index = P7	Null	Null	<b>Inactive</b>	Inactive	Inactive	Inactive
PA_Index = P9		<b>Active</b>	Active	Active	Active	Active

Note: Bold font indicates the change invoked by write operation

Minimum\_On\_Time

T1

T2

Configuration Requirements: The object to be tested shall be configured such that the present value is set to INACTIVE, and slot 6 in the Priority\_Array has a value of NULL. The object being tested must also be configured with Minimum\_On\_Time and Minimum\_Off\_Time values sufficiently large enough to allow execution of this test. If no object exists with both Minimum\_On\_Time and Minimum\_Off\_Time properties, this test shall be skipped.

Notes to Tester: P9 and P7 may be equal.

Test Steps:

1. VERIFY Present\_Value = INACTIVE
2. VERIFY Priority\_Array = NULL, ARRAY INDEX = 6
3. WRITE Present\_Value = ACTIVE, PRIORITY = P9
4. VERIFY Present\_Value = ACTIVE
5. VERIFY Priority\_Array = ACTIVE, ARRAY INDEX = 6
- ...(Execute steps 6 through 7 before Minimum\_On\_Time expires)
6. WRITE Present\_Value = INACTIVE, PRIORITY = P7
7. VERIFY Present\_Value = ACTIVE
8. VERIFY Priority\_Array = ACTIVE, ARRAY INDEX = 6
9. WAIT for Minimum\_On\_Time to expire
- ...(Execute steps 10 and 11 before T1)
10. VERIFY Present\_Value = INACTIVE
11. VERIFY Priority\_Array = INACTIVE, ARRAY INDEX = 6
- ...(Execute step 12 between T1 and T2)
12. VERIFY Present\_Value = INACTIVE
13. VERIFY Priority\_Array = INACTIVE, ARRAY INDEX = 6
- ...(Execute step 14 and 15 after T2)
14. VERIFY Present\_Value = INACTIVE
15. VERIFY Priority\_Array = NULL, ARRAY INDEX = 6

[Add new **Clause 7.3.1.6.9**, p. 29]

### 7.3.1.6.9 Ensuring Minimum\_On\_Time starts at transition to ACTIVE

Purpose: To verify that Minimum\_On\_Time does not start immediately after a write operation while Minimum\_Off\_Time is in effect and present-value is INACTIVE.

Test Concept: The initial Present\_Value of the object being tested is set to ACTIVE, and the value at slot 6 in the priority-array has a value of NULL. Present\_Value of the object is written to INACTIVE at P9, where P9 is a priority between 7 and 16, such that present-value and slot 6 in the priority-array change to INACTIVE. Before Minimum\_Off\_Time expires, Present\_Value is written to ACTIVE at P7, where P7 is a priority between 7 and P9, such that Present\_Value would change if



Minimum\_Off\_Time were not in effect. Slot 6 in the priority-array is monitored at specific time intervals to ensure that it contains the appropriate value. Time references T1 and T2 are defined for this test as follows:

T1 = the time when the ACTIVE request is executed by the device + Minimum\_On\_Time

T2 = the time when the INACTIVE request is executed by the device + Minimum\_Off\_Time + Minimum\_On\_Time

Test Steps →	1-2	3-5	6-9	10-11	12-13	14-15
Present_Value	Active	Inactive	Inactive	Active	Active	Active
PA_Index = 6	Null	Inactive	Inactive	Active	Active	Null
PA_Index = P7	Null	Null	<b>Active</b>	Active	Active	Active
PA_Index = P9		<b>Inactive</b>	Inactive	Inactive	Inactive	Inactive

Note: Bold font indicates the change invoked by write operation

Minimum\_On\_Time

T1

T2

Configuration Requirements: The object to be tested shall be configured such that the present value is set to ACTIVE, and slot 6 in the Priority\_Array has a value of NULL. The object being tested must also be configured with Minimum\_On\_Time and Minimum\_Off\_Time values sufficiently large enough to allow execution of this test. If no object exists with both Minimum\_On\_Time and Minimum\_Off\_Time properties, this test shall be skipped.

Notes to Tester: P9 and P7 may be equal.

Test Steps:

1. VERIFY Present\_Value = ACTIVE
2. VERIFY Priority\_Array = NULL, ARRAY INDEX = 6
3. WRITE Present\_Value = INACTIVE, PRIORITY = P9
4. VERIFY Present\_Value = INACTIVE
5. VERIFY Priority\_Array = INACTIVE, ARRAY INDEX = 6
- ...(Execute steps 6 through 7 before Minimum\_Off\_Time expires)
6. WRITE Present\_Value = ACTIVE, PRIORITY = P7
7. VERIFY Present\_Value = INACTIVE
8. VERIFY Priority\_Array = INACTIVE, ARRAY INDEX = 6
9. WAIT for Minimum\_Off\_Time to expire
- ...(Execute steps 10 and 11 before T1)
10. VERIFY Present\_Value = ACTIVE
11. VERIFY Priority\_Array = ACTIVE, ARRAY INDEX = 6
- ...(Execute step 12 between T1 and T2)
12. VERIFY Present\_Value = ACTIVE
13. VERIFY Priority\_Array = ACTIVE, ARRAY INDEX = 6
- ...(Execute step 14 and 15 after T2)
14. VERIFY Present\_Value = ACTIVE
15. VERIFY Priority\_Array = NULL, ARRAY INDEX = 6

[Add new **Clause 7.3.1.6.10**, p. 29]

### 7.3.1.6.10 Ensuring Minimum Times Are Not Affected By Time Changes

Purpose: To verify that minimum times are not affected by changing the time in a device via TimeSynchronization or UTCTimeSynchronization requests.

Test Concept: The initial Present\_Value of the object being tested is set to ACTIVE, and the value at slot 6 in the priority-array has a value of NULL. Present\_Value of the object is written to INACTIVE such that present-value and slot 6 in the priority-

array change to INACTIVE. Before Minimum\_Off\_Time expires, the time is changed to a value T1 which is more than Minimum\_Off\_Time in the future and Present\_Value, and Slot 6 in the priority-array are read to verify that they were not affected by the time change. After Minimum\_Off\_Time expires, slot 6 in the priority-array is read again to verify that it is no longer INACTIVE.

Configuration Requirements: The object to be tested shall be configured such that the present value is set to ACTIVE, and slot 6 in the Priority\_Array has a value of NULL. If the IUT does not support TimeSynchronization or UTC-TimeSynchronization, then this test shall be omitted.

Notes to Tester: The test above is written for Minimum\_Off\_Time. To execute this test for Minimum\_On\_Time, use INACTIVE where ACTIVE is specified, ACTIVE where INACTIVE is specified, and Minimum\_On\_Time where Minimum\_Off\_Time is specified.

Test Steps:

1. VERIFY Present\_Value = ACTIVE
2. VERIFY Priority\_Array = NULL, ARRAY INDEX = 6
3. WRITE Present\_Value = INACTIVE
4. VERIFY Present\_Value = INACTIVE
5. VERIFY Priority\_Array = INACTIVE, ARRAY INDEX = 6
6. TRANSMIT
  - DA = GLOBAL BROADCAST,
  - SA = TD
  - BACnet-Unconfirmed-Request-PDU,
  - ‘Service Choice’ = TimeSynchronization-Request,
  - Date = T1,
  - Time = T1
7. TRANSMIT
  - DA = GLOBAL BROADCAST,
  - SA = TD
  - BACnet-Unconfirmed-Request-PDU,
  - ‘Service Choice’ = UTC-TimeSynchronization-Request,
  - Date = T1,
  - Time = T1
8. VERIFY Priority\_Array = INACTIVE, ARRAY INDEX = 6
9. WAIT (the remainder of Minimum\_Off\_Time)
10. VERIFY Priority\_Array <> INACTIVE, ARRAY INDEX = 6

[Add new **Clause 7.3.1.6.11**, p. 29]

### 7.3.1.6.11 Minimum\_Off\_Time - Value Source Mechanism

Purpose: To verify that the value source used for priority 6 is the commanded object while Minimum\_Off\_Time is in effect.

Test Concept: A commandable object which supports the value source mechanism is selected for the test. When Minimum\_Off\_Time takes effect, the Present\_Value is written. The Value\_Source and Value\_Source\_Array properties are monitored to verify that the source for priority 6 is the commanded object.

Configuration Requirements: The object, O1, to be tested shall be configured such that slot 6 in the Priority\_Array and Value\_Source\_Array has a value of NULL. The object being tested must also be configured with Minimum\_Off\_Time values sufficiently large enough to allow execution of this test. If no object exists with Minimum\_Off\_Time property, this test shall be skipped.

Test Steps:

1. VERIFY Value\_Source = (any valid value)
2. VERIFY Priority\_Array = NULL, ARRAY INDEX = 6
3. VERIFY Value\_Source\_Array = NULL, ARRAY INDEX = 6
4. WRITE Present\_Value = INACTIVE, PRIORITY > 6
5. VERIFY Present\_Value = INACTIVE

6. VERIFY Priority\_Array = INACTIVE, ARRAY INDEX = 6
7. VERIFY Value\_Source = O1
8. VERIFY Value\_Source\_Array = O1, ARRAY INDEX = 6
9. WAIT (Minimum ON/OFF Fail Time + Minimum\_Off\_Time )
10. VERIFY Value\_Source = 'None'

[Add new **Clause 7.3.1.6.12**, p. 29]

### **7.3.1.6.12 Minimum\_On\_Time - Value Source Mechanism**

Purpose: To verify that the value source used for priority 6 is the commanded object while Minimum\_On\_Time is in effect.

Test Concept: A commandable object which supports the value source mechanism is selected for the test. When Minimum\_On\_Time takes effect, the Present\_Value is written. The Value\_Source and Value\_Source\_Array properties are monitored to verify that the source for priority 6 is the commanded object.

Configuration Requirements: The object, O1, to be tested shall be configured such that slot 6 in the Priority\_Array and Value\_Source\_Array has a value of NULL. The object being tested must also be configured with Minimum\_On\_Time values sufficiently large enough to allow execution of this test. If no object exists with Minimum\_On\_Time property, this test shall be skipped.

Test Steps:

1. VERIFY Value\_Source = (any valid value)
2. VERIFY Priority\_Array = NULL, ARRAY INDEX = 6
3. VERIFY Value\_Source\_Array = NULL, ARRAY INDEX = 6
4. WRITE Present\_Value = ACTIVE, PRIORITY > 6
5. VERIFY Present\_Value = ACTIVE
6. VERIFY Priority\_Array = ACTIVE, ARRAY INDEX = 6
7. VERIFY Value\_Source = O1
8. VERIFY Value\_Source\_Array = O1, ARRAY INDEX = 6
9. WAIT (Minimum ON/OFF Fail Time + Minimum\_On\_Time)
10. VERIFY Value\_Source = 'None'

[Change **Clause 7.3.1.7**, p. 29]

### **7.3.1.7 COV Tests**

~~Tests to demonstrate COV functionality are covered in 8.2 and 9.6.~~

[Add new **Clause 7.3.1.7.X1**, p. 29]

### **7.3.1.7.X1 COV\_Resubscription\_Interval Test**

Purpose: To verify that object O1 acquiring data via COV notification reissues its subscription at the interval set by COV\_Resubscription\_Interval.

Test Concept: O1 is configured to acquire data from the TD by COV notification. The TD verifies the resubscription interval.

Configuration Requirements: O1 is configured to acquire data from TD by COV notification. Non-zero values shall be chosen for COV\_Resubscription\_Interval in accordance with the range and resolution specified by the manufacturer for this property.

Test Steps:

1. IF (the IUT uses SubscribeCOV) THEN  
RECEIVE SubscribeCOV-Request,

```

'Subscriber Process Identifier' = (SPI1, any value),
'Monitored Object Identifier' = (MOI1, the object to be monitored),
'Issue Confirmed Notifications' = (ICN1 = TRUE | FALSE),
'Lifetime' = (L1, any value >= COV_Resubscription_Interval)
ELSE
    RECEIVE SubscribeCOVProperty-Request,
    'Subscriber Process Identifier' = (SPI1, any value),
    'Monitored Object Identifier' = (MOI1, the object to be monitored),
    'Issue Confirmed Notifications' = (ICN1 = TRUE | FALSE),
    'Lifetime' = (L1, any value >= COV_Resubscription_Interval),
    'Monitored Property Identifier' = (MPI1, the property to be monitored),
    'COV Increment' = (CI1, Client_COV_Increment -- optional)
2. TRANSMIT BACnet-SimpleACK-PDU
3. TRANSMIT ConfirmedCOVNotification-Request,
    'Subscriber Process Identifier' = SPI1,
    'Initiating Device Identifier' = TD,
    'Monitored Object Identifier' = MOI1,
    'Issue Confirmed Notifications' = ICN1,
    'Time Remaining' = (any value <= L1),
    'List of Values' = (appropriate BACnetPropertyValue(s))
4. RECEIVE BACnet-SimpleACK-PDU
5. BEFORE (the lesser of COV_Resubscription_Interval + Re-subscription Interval Tolerance and L1LifeTime from step
4)
    IF (the IUT uses SubscribeCOV) THEN
        RECEIVE SubscribeCOV-Request,
        'Subscriber Process Identifier' = SPI1,
        'Monitored Object Identifier' = MOI1,
        'Issue Confirmed Notifications' = ICN1,
        'Lifetime' = (L2, any value >= COV_Resubscription_Interval)
    ELSE
        RECEIVE SubscribeCOVProperty-Request,
        'Subscriber Process Identifier' = SPI1,
        'Monitored Object Identifier' = MOI1,
        'Issue Confirmed Notifications' = ICN1,
        'Lifetime' = (L2, any value >= COV_Resubscription_Interval)
        'Monitored Property Identifier' = MPI1,
        'COV Increment' = CI1
6. TRANSMIT BACnet-SimpleACK-PDU
7. TRANSMIT ConfirmedCOVNotification-Request,
    'Subscriber Process Identifier' = SPI1,
    'Initiating Device Identifier' = TD,
    'Monitored Object Identifier' = MOI1,
    'Issue Confirmed Notifications' = ICN1,
    'Time Remaining' = (any value <= L2),
    'List of Values' = (appropriate BACnetPropertyValue(s))
8. RECEIVE BACnet-SimpleACK-PDU
9. WAIT (COV_Resubscription_Interval - Re-subscription Interval Tolerance)
10. BEFORE (2 * Re-subscription Interval Tolerance)
    IF (the IUT uses SubscribeCOV) THEN
        RECEIVE SubscribeCOV-Request,
        'Subscriber Process Identifier' = SPI1,
        'Monitored Object Identifier' = MOI1,
        'Issue Confirmed Notifications' = ICN1,
        'Lifetime' = L1
    ELSE
        RECEIVE SubscribeCOVProperty-Request,
        'Subscriber Process Identifier' = SPI1,
        'Monitored Object Identifier' = MOI1,
        'Issue Confirmed Notifications' = ICN1,
        'Lifetime' = L1;

```

'Monitored Property Identifier' = MPI1,  
'COV Increment' = CI1

## 11. TRANSMIT BACnet-SimpleACK-PDU

Passing Result: Where the Lifetime parameter of a SubscribeCOV request is less than COV\_Resubscription\_Interval + Re-subscription Interval Tolerance, the IUT shall send the subsequent SubscribeCOV request within Lifetime seconds even though this is a smaller time window than defined by the test. If the IUT does not meet this stricter time window, then the IUT shall fail the test.

[Change **Clause 7.3.1.8**, p. 29]

[Reviewer Note: *Renamed the test and modified steps to express using READ, WRITE and VERIFY commands.*]

### 7.3.1.8 ~~Binary Object~~ Change of State Tests

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

~~BACnet Reference Clauses: 12.6.14, 12.6.15, 12.6.16, 12.7.14, 12.7.15, 12.7.16, 12.8.12, 12.8.13, and 12.8.14.~~

Purpose: To verify that the properties of ~~binary~~ objects that collectively track state changes (changes in Present\_Value) function as required. ~~If the Change\_Of\_State\_Count, Change\_Of\_State\_Time, and Time\_Of\_State\_Count\_Reset properties are not supported this test shall be omitted. This test applies to Binary Input, Binary Output, and Binary Value objects.~~

Test Concept: The Present\_Value of the ~~binary~~ object under test is changed. The Change\_Of\_State\_Count property is checked to verify that it has been incremented, and the Change\_Of\_State\_Time property is checked to verify that it has been updated. The Change\_Of\_State\_Count is reset and Time\_Of\_State\_Count\_Reset is checked to verify that it has been updated appropriately.

Configuration Requirements: The object being tested shall be configured such that the Present\_Value and Change\_Of\_State\_Count properties are writable, or another means of changing these properties shall be provided.

Test Steps:

1. READ PV = Present\_Value
2. READ N = Change\_of\_State\_Count
3. IF (PV = ACTIVE) THEN
  - IF (Present\_Value is writable) THEN
    - WRITE Present\_Value = INACTIVE
    - VERIFY Present\_Value = INACTIVE
  - ELSE
    - MAKE (Present\_Value = INACTIVE)
- ELSE
  - IF (Present\_Value is writable) THEN
    - WRITE Present\_Value = ACTIVE
    - VERIFY Present\_Value = ACTIVE
  - ELSE
    - MAKE (Present\_Value = ACTIVE)
4. VERIFY (Change\_of\_State\_Count = N+1)
5. VERIFY (Change\_Of\_State\_Time ~= the current local date and time)
6. IF (Change\_of\_State\_Count is writable) THEN
  - WRITE Change\_of\_State\_Count = 0
- ELSE
  - MAKE (Change\_of\_State\_Count = 0)
7. VERIFY Time\_Of\_State\_Count\_Reset ~= (the current local date and time)
- ~~1. TRANSMIT ReadProperty Request,~~
  - ~~—— 'Object Identifier' = (the object being tested),~~
  - ~~—— 'Property Identifier' = Present\_Value~~
- ~~2. RECEIVE ReadProperty ACK,~~
  - ~~—— 'Object Identifier' = (the object being tested),~~
  - ~~—— 'Property Identifier' = Present\_Value,~~

```

—— 'Property Value' = —— ACTIVE | INACTIVE
3. TRANSMIT ReadProperty Request,
—— 'Object Identifier' = —— (the object being tested),
—— 'Property Identifier' = —— Change_Of_State_Count
4. RECEIVE ReadProperty ACK,
—— 'Object Identifier' = —— (the object being tested),
—— 'Property Identifier' = Change_Of_State_Count,
—— 'Property Value' = —— (any valid value, N)
5. IF (Present_Value is writable) THEN
—— IF (the value returned in step 2 was ACTIVE) THEN
—— WRITE Present_Value = INACTIVE
—— VERIFY Present_Value = INACTIVE
—— ELSE
—— WRITE Present_Value = ACTIVE
—— VERIFY Present_Value = ACTIVE
—— ELSE
—— MAKE (Present_Value change to the opposite state)
6. TRANSMIT ReadProperty Request,
—— 'Object Identifier' = —— (the IUT's Device object),
—— 'Property Identifier' = —— Local_Date
7. RECEIVE ReadProperty ACK,
—— 'Object Identifier' = —— (the IUT's Device object),
—— 'Property Identifier' = Local_Date,
—— 'Property Value' = —— (the current local date, D)
8. TRANSMIT ReadProperty Request,
—— 'Object Identifier' = —— (the IUT's Device object),
—— 'Property Identifier' = —— Local_Time
9. RECEIVE ReadProperty ACK,
—— 'Object Identifier' = —— (the IUT's Device object),
—— 'Property Identifier' = Local_Time,
—— 'Property Value' = —— (the current local time, TLOC)
10. WAIT Internal Processing Fail Time
11. TRANSMIT ReadProperty Request,
—— 'Object Identifier' = —— (the object being tested),
—— 'Property Identifier' = —— Change_Of_State_Time
12. RECEIVE ReadProperty ACK,
—— 'Object Identifier' = —— (the object being tested),
—— 'Property Identifier' = Change_Of_State_Time,
—— 'Property Value' = —— (a date and time such that the date = D and the time is approximately TLOC)
13. TRANSMIT ReadProperty Request,
—— 'Object Identifier' = —— (the object being tested),
—— 'Property Identifier' = —— Change_Of_State_Count
14. RECEIVE ReadProperty ACK,
—— 'Object Identifier' = —— (the object being tested),
—— 'Property Identifier' = Change_Of_State_Count,
—— 'Property Value' = —— N + 1
15. IF (Change_Of_State_Count is writable) THEN
—— WRITE Change_Of_State_Count = 0
—— VERIFY Change_Of_State_Count = 0
—— ELSE
—— MAKE (Change_Of_State_Count = 0)
16. TRANSMIT ReadProperty Request,
—— 'Object Identifier' = —— (the IUT's Device object),
—— 'Property Identifier' = —— Local_Time
17. RECEIVE ReadProperty ACK,
—— 'Object Identifier' = —— (the IUT's Device object),
—— 'Property Identifier' = Local_Time,
—— 'Property Value' = —— (the current local time, TLOC)
18. TRANSMIT ReadProperty Request,
—— 'Object Identifier' = —— (the object being tested),

```

~~'Property Identifier' = Time\_Of\_State\_Count\_Reset~~  
19. ~~RECEIVE ReadProperty ACK,~~  
~~'Object Identifier' = (the object being tested),~~  
~~'Property Identifier' = Time\_Of\_State\_Count\_Reset,~~  
~~'Property Value' = (a date and time such that the date = D and the time is approximately T<sub>LOC</sub>)~~

[Change Clause 7.3.1.9, p. 31]

[Reviewer Note: Changes were made to address errors which were pointed out via BTL-CR-0253, and in order to express using READ, WRITE and VERIFY commands.]

### 7.3.1.9 Binary Object-Elapsed Active Time Tests

Purpose: To verify that the properties of binary objects that collectively track active time function properly. ~~If the Elapsed\_Active\_Time and Time\_Of\_Active\_Time\_Reset properties are not supported then this test shall be omitted. This test applies to Binary Input, Binary Output, and Binary Value and Binary Lighting Output objects.~~

Test Concept: The Present\_Value or Feedback\_Value of the binary object being tested is set to INACTIVE. The Elapsed\_Active\_Time property is checked to verify that it does not accumulate time while the object is in an INACTIVE state. The Present\_Value or Feedback\_Value is then set to ACTIVE. The Elapsed\_Active\_Time property is checked to verify that it is accumulating time while the object is in an ACTIVE state. ~~The Present\_Value or Feedback\_Value is then set to INACTIVE and the Elapsed\_Active\_Time is reset.~~ The Time\_Of\_Active\_Time\_Reset property is checked to verify that it has been updated.

Configuration Requirements: The object being tested shall be configured such that the Present\_Value or Feedback\_Value if that is used for the calculation, and Elapsed\_Active\_Time properties are writable or another means of changing these properties shall be provided. *Whether Present\_Value or Feedback\_Value is used as the indicator for the calculation of the Elapsed\_Active\_Time is a local matter.*

Notes To Tester: *It was intentional to specify that the alternative use of Feedback\_Value tracking specified in 135-2010ad-3 is allowed regardless of the Protocol\_Revision claimed by the implementation.*

Test Steps:

1. IF (Present\_Value is writable) THEN  
    WRITE Present\_Value = INACTIVE  
    VERIFY Present\_Value = INACTIVE  
ELSE  
    MAKE (Present\_Value = INACTIVE)
  2. IF (Feedback\_Value is used for Elapsed\_Active\_Time tracking) THEN  
    WAIT(long enough for Feedback\_Value to reflect the Present\_Value)  
    VERIFY Feedback\_Value = INACTIVE  
~~TRANSMIT ReadProperty Request,~~  
~~'Object Identifier' = (the object being tested),~~  
~~'Property Identifier' = Elapsed\_Active\_Time~~
  3. ~~RECEIVE ReadProperty ACK,~~  
~~'Object Identifier' = (the object being tested),~~  
~~'Property Identifier' = Elapsed\_Active\_Time,~~  
~~'Property Value' = (the elapsed active time, T<sub>ELAPSED</sub> in seconds)~~
  3. READ Elapsed\_Active\_Time = initialElapsedTime
- verify that Elapsed\_Active\_Time does not change when the object is INACTIVE
4. ~~WAIT (1 minute)~~ WAIT (more than Internal Processing Fail Time + at least 1 second)
  5. VERIFY Elapsed\_Active\_Time = initialElapsedTime
- verify that Elapsed\_Active\_Time correctly reflects the time the object is ACTIVE
5. ~~TRANSMIT ReadProperty Request,~~  
~~'Object Identifier' = (the object being tested),~~  
~~'Property Identifier' = Elapsed\_Active\_Time~~
  6. ~~RECEIVE ReadProperty ACK,~~

```

'Object Identifier' = (the object being tested),
'Property Identifier' = Elapsed_Active_Time,
'Property Value' = (the same TELAPSED as step 3)
6. IF (Present_Value is writable) THEN
    WRITE Present_Value = ACTIVE
    VERIFY Present_Value = ACTIVE
ELSE
    MAKE (Present_Value = ACTIVE)
7. IF (Feedback_Value is used for Elapsed_Active_Time tracking) THEN
    WAIT (long enough for Feedback_Value to reflect the Present_Value)
    VERIFY Feedback_Value = ACTIVE
8. READ initialTime = (the IUT's Device object) Local_Time
9. WAIT (more than Internal Processing Fail Time + 30 seconds)
10. IF (Present_Value is writable) THEN
    WRITE Present_Value = INACTIVE
    VERIFY Present_Value = INACTIVE
ELSE
    MAKE (Present_Value = INACTIVE)
11. IF (Feedback_Value is used for Elapsed_Active_Time tracking) THEN
    WAIT (long enough for Feedback_Value to reflect the Present_Value)
    VERIFY Feedback_Value = INACTIVE
12. READ currentTime = (the IUT's Device object) Local_Time
13. READ totalElapsedTime = Elapsed_Active_Time
14. CHECK (totalElapsedTime ~ (currentTime - initialTime) - initialElapsedTime)

-- verify ability to reset Elapsed_Active_Time, if it is writable
15. IF (Elapsed_Active_Time is writable) THEN
    WRITE Elapsed_Active_Time = 0
    READ currentDate = (the IUT's Device object) Local_Date
    READ currentTime = (the IUT's Device object) Local_Time
    VERIFY Time_Of_Active_Time_Reset ~ { currentDate, currentTime }

10. TRANSMIT ReadProperty Request,
'Object Identifier' = (the object being tested),
'Property Identifier' = Elapsed_Active_Time
11. RECEIVE ReadProperty ACK,
'Object Identifier' = (the object being tested),
'Property Identifier' = Elapsed_Active_Time,
'Property Value' = (T: (TELAPSED + 30) ≤ T ≤ (TELAPSED + TimeX, where TimeX is the time between the beginning
of step 7 and this step 30 + Internal Processing Fail Time))
11. IF (Present_Value is writable) THEN
WRITE Present_Value = INACTIVE
VERIFY Present_Value = INACTIVE
ELSE
MAKE (Present_Value = INACTIVE)
12. IF (Elapsed_Active_Time is writable) THEN
WRITE Elapsed_Active_Time = 0
VERIFY Elapsed_Active_Time = 0
ELSE
MAKE (Elapsed_Active_Time = 0)
13. TRANSMIT ReadProperty Request,
'Object Identifier' = (the IUT's Device object),
'Property Identifier' = Local_Date
14. RECEIVE ReadProperty ACK,
'Object Identifier' = (the IUT's Device object),
'Property Identifier' = Local Date,
'Property Value' = (the current local date, D)
15. TRANSMIT ReadProperty Request,
'Object Identifier' = (the IUT's Device object),
'Property Identifier' = Local_Time

```



16. ~~RECEIVE ReadProperty ACK,~~  
~~'Object Identifier' = (the IUT's Device object),~~  
~~'Property Identifier' = Local\_Time,~~  
~~'Property Value' = (the current local time, T<sub>LOC</sub>)~~  
17. ~~TRANSMIT ReadProperty Request,~~  
~~'Object Identifier' = (the object being tested),~~  
~~'Property Identifier' = Time\_Of\_Active\_Time\_Reset~~  
18. ~~RECEIVE ReadProperty ACK,~~  
~~'Object Identifier' = (the object being tested),~~  
~~'Property Identifier' = Present\_ValueTime\_Of\_Active\_Time\_Reset,~~  
~~'Property Value' = (a date and time such that the date = D and the time is approximately T<sub>LOC</sub>)~~

### 7.3.1.10 Event\_Enable Tests

[Change **Clause 7.3.1.10.1**, p. 32]

[Reviewer Note: This test was modified to add a clarifying sentence to Purpose and clarify that the received response in step 11 is from O1.]

#### 7.3.1.10.1 Event\_Enable Test for TO\_OFFNORMAL and TO\_NORMAL, and TO\_FAULT

Purpose: To verify that notification messages are transmitted only if the bit in Event\_Enable corresponding to the event transition has a value of TRUE. *This test applies to Event Enrollment objects and objects that support intrinsic reporting.*

Test Concept: The IUT is configured with an event-generating object, O1, such that the Event\_Enable property is tested in all supported states. Each event transition is triggered and the IUT is monitored to verify that notification messages are transmitted only for those transitions for which the Event\_Enable property has a value of TRUE.

Configuration Requirements: If the Event\_Enable property is configurable, repeat the test with Event\_Enable = (T,F,F),(F,T,F),(F,F,T). If the Event\_Enable property is not configurable, then follow the test steps as written and verify correct behavior for the value of the Event\_Enable property. All other properties in O1, and any supporting objects, shall be configured to allow these events to be generated. The event-generating object shall be in a NORMAL state at the start of the test. D1 is either the pTimeDelay parameter or, in the case of the EXTENDED and proprietary algorithms, a vendor specific delay (may be zero).

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

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

*Test Steps:*

1. VERIFY pCurrentState = NORMAL
2. WAIT (pTimeDelay + **Notification Fail Time**)
3. IF (O1 contains pFeedbackValue) THEN  
MAKE (pFeedbackValue differ from pMonitoredValue)  
ELSE IF (pMonitoredValue is writable) THEN  
WRITE pMonitoredValue = (a value that is OFFNORMAL)  
ELSE  
MAKE (pMonitoredValue have a value that is OFFNORMAL)
4. WAIT (D1)
5. BEFORE **Notification Fail Time**  
IF (the Transitions bit corresponding to the TO-OFFNORMAL transition is TRUE) THEN {  
RECEIVE ConfirmedEventNotification-Request,  
'Process Identifier' = (any valid process ID),  
'Initiating Device Identifier' = IUT,  
'Event Object Identifier' = O1,

```

        'Time Stamp' = (any valid time stamp),
        'Notification Class' = (the class corresponding to the object being tested),
        'Priority' = (the value configured to correspond to a TO-OFFNORMAL transition),
        'Event Type' = (any valid event type),
        'Message Text' = (optional, any valid message text),
        'Notify Type' = EVENT | ALARM,
        'AckRequired' = TRUE | FALSE,
        'From State' = NORMAL,
        'To State' = OFFNORMAL,
        'Event Values' = (values appropriate to the event type)
    TRANSMIT BACnet-SimpleACK-PDU
}
ELSE
    CHECK (verify that the IUT did not transmit an event notification message)
6. VERIFY pCurrentState = OFFNORMAL
7. IF (O1 contains pFeedbackValue) THEN
    MAKE (pFeedbackValue equal to pMonitoredValue)
ELSE IF (pMonitoredValue is writable) THEN
    WRITE pMonitoredValue = (a value that is NORMAL)
ELSE
    MAKE (pMonitoredValue have a value that is NORMAL)
8. WAIT (D2)
9. BEFORE Notification Fail Time
    IF (the Transitions bit corresponding to the TO-NORMAL transition is TRUE) THEN {
        RECEIVE ConfirmedEventNotification-Request,
        'Process Identifier' = (any valid process ID),
        'Initiating Device Identifier' = IUT,
        'Event Object Identifier' = O1,
        'Time Stamp' = (any valid time stamp),
        'Notification Class' = (the class corresponding to the object being tested),
        'Priority' = (the value configured to correspond to a TO-NORMAL transition),
        'Event Type' = (any valid event type),
        'Message Text' = (optional, any valid message text),
        'Notify Type' = EVENT | ALARM,
        'AckRequired' = TRUE | FALSE,
        'From State' = OFFNORMAL,
        'To State' = NORMAL,
        'Event Values' = (values appropriate to the event type)
    TRANSMIT BACnet-SimpleACK-PDU
    }
ELSE
    CHECK (verify that the IUT did not transmit an event notification message)
10. VERIFY pCurrentState = NORMAL
11. IF (O1 can be placed into a fault condition) THEN {
    MAKE (a condition exist that will cause O1 to generated a TO-FAULT transition)
    BEFORE Notification Fail Time
    IF (the Transitions bit corresponding to the TO-FAULT transition is TRUE) THEN {
        RECEIVE ConfirmedEventNotification-Request,
        'Process Identifier' = (any valid process ID),
        'Initiating Device Identifier' = IUT,
        'Event Object Identifier' = O1 (the event generating object configured for this test),
        'Time Stamp' = (any valid time stamp),
        'Notification Class' = (the class corresponding to the object being tested),
        'Priority' = (the value configured to correspond to a TO-FAULT transition),
        'Event Type' = (any valid event type),
        'Message Text' = (optional, any valid message text),
        'Notify Type' = EVENT | ALARM,
        'AckRequired' = TRUE | FALSE,
        'From State' = NORMAL,
        'To State' = FAULT,

```

```

        'Event Values' = (values appropriate to the event type)
        TRANSMIST BACnet-SimpleACK-PDU
    }
    ELSE
        CHECK (verify that the IUT did not transmit an event notification message)
        VERIFY Event_State = FAULT
    }

```

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

[Change **Clause 7.3.1.11**, p.36]

[Reviewer Note: Corrected errata issues which exist in 135.1-2019. Improved the text for Notes To Tester. Renumbered and renamed the test to make it a subclause of 7.3.1.11.]

### 7.3.1.11 Acked\_Transitions Tests

#### 7.3.1.11.1 Acked\_Transitions Tests Test

Purpose: To verify that the Acked\_Transitions property tracks whether or not an acknowledgment has been received for a previously issued event notification. It also verifies the interrelationship between Status\_Flags and Event\_State.

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

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

*Notes to Tester: The UnconfirmedEventNotification service may be substituted for the ConfirmedEventNotification service, in which case the TD shall skip sending the BACnet-SimpleACK-PDU messages after receiving the notifications.*

Test Steps:

1. VERIFY pCurrentState = NORMAL
2. VERIFY Acked\_Transitions = (TRUE, TRUE, TRUE)
3. IF (Protocol\_Revision is present AND Protocol\_Revision >= 13) THEN  
    VERIFY pStatusFlags = (FALSE, FALSE, ?, ?)
4. IF (pMonitoredValue is writable) THEN  
    WRITE pMonitoredValue = (a value that is OFFNORMAL)  
ELSE  
    MAKE (pMonitoredValue have a value that is OFFNORMAL)
5. WAIT (pTimeDelay)
6. BEFORE **Notification Fail Time**  
    RECEIVE ConfirmedEventNotification-Request,  
        'Process Identifier' = (PI1: any valid process ID),  
        'Initiating Device Identifier' = IUT,  
        'Event Object Identifier' = (the event-generating object configured for this test),  
        'Time Stamp' = (Toffnormal: any valid time stamp),  
        'Notification Class' = (the class corresponding to the object being tested),  
        'Priority' = (Poffnormal: the value configured to correspond to a TO-OFFNORMAL transition),  
        'Event Type' = (any valid event type),

```

'Message Text' =          (optional, any valid message text),
'Notify Type' =          (the notify type configured for this event),
'AckRequired' =          TRUE,
'From State' =          NORMAL,
'To State' =          OFFNORMAL,
'Event Values' =          (values appropriate to the event type)
7. TRANSMIT BACnet-SimpleACK-PDU
8. VERIFY pCurrentState = OFFNORMAL
9. VERIFY Acked_Transitions = (FALSE, TRUE, TRUE)
10. IF (Protocol_Revision is present AND Protocol_Revision >= 13) THEN
    VERIFY pStatusFlags = (TRUE, FALSE, ?, ?)
11. IF (pMonitoredValue is writable) THEN
    WRITE pMonitoredValue = (a value that is NORMAL)
ELSE
    MAKE (pMonitoredValue have a value that is NORMAL)
12. WAIT (pTimeDelayNormal)
13. BEFORE Notification Fail Time
    RECEIVE ConfirmedEventNotification-Request,
    'Process Identifier' =      (PI2: any valid process ID),
    'Initiating Device Identifier' = IUT,
    'Event Object Identifier' = (the event-generating object configured for this test),
    'Time Stamp' =            (Tnormal: any valid time stamp),
    'Notification Class' =     (the class corresponding to the object being tested),
    'Priority' =              (Pnormal: the value configured to correspond to a TO-NORMAL transition),
    'Event Type' =           (any valid event type),
    'Message Text' =          (optional, any valid message text),
    'Notify Type' =          (the notify type configured for this event),
    'AckRequired' =          TRUE,
    'From State' =          OFNORMAL,
    'To State' =            NORMAL,
    'Event Values' =          (values appropriate to the event type)
14. TRANSMIT BACnet-SimpleACK-PDU
15. VERIFY pCurrentState = NORMAL
16. VERIFY Acked_Transitions = (FALSE, TRUE, FALSE)
17. IF (Protocol_Revision is present AND Protocol_Revision >= 13) THEN
    VERIFY pStatusFlags = (FALSE, FALSE, ?, ?)
18. IF (the event-triggering object can be placed into a fault condition) THEN {
    MAKE (a condition exist that will cause the object to generate a fault condition)
    BEFORE Notification Fail Time
        RECEIVE ConfirmedEventNotification-Request,
        'Process Identifier' =      (PI3: any valid process ID),
        'Initiating Device Identifier' = IUT,
        'Event Object Identifier' = (the event-generating object configured for this test),
        'Time Stamp' =            (Tfault: any valid time stamp),
        'Notification Class' =     (the class corresponding to the object being tested),
        'Priority' =              (Pfault: the value configured to correspond to a TO-FAULT transition),
        'Event Type' =           IF (Protocol_Revision < 13) THEN
                                (any valid event type),
                                ELSE
                                CHANGE_OF_RELIABILITY,
        'Message Text' =          (optional, any valid message text),
        'Notify Type' =          (the notify type configured for this event),
        'AckRequired' =          TRUE,
        'From State' =          NORMAL,
        'To State' =            FAULT,
        'Event Values' =          (values appropriate to the event type)
    TRANSMIT BACnet-SimpleACK-PDU
    VERIFY pCurrentState = FAULT
    VERIFY Acked_Transitions = (FALSE, FALSE, FALSE)
    TRANSMIT AcknowledgeAlarm-Request,

```

```

'Acknowledging Process Identifier' = (PI3),
'Event Object Identifier' = (the event-generating object configured for this test),
'Event State Acknowledged' = FAULT,
'Acknowledgement Source' = (a character string),
'Time Stamp' = (Tfault),
'Time of Acknowledgment' = (the TD's current time)
RECEIVE BACnet-SimpleACK-PDU
IF (Protocol_Revision is present AND Protocol_Revision ≥ 1) THEN
    BEFORE Notification Fail Time
        RECEIVE ConfirmedEventNotification-Request,
            'Process Identifier' = (PI3),
            'Initiating Device Identifier' = IUT,
            'Event Object Identifier' = (the event-generating object configured for this test),
            'Time Stamp' = (Faultthe IUT's current time or sequence number),
            'Notification Class' = (the class corresponding to the object being tested),
            'Priority' = (Pfault),
            'Event Type' = IF (Protocol_Revision < 13)
                            (any valid event type),
                            ELSE
                                CHANGE_OF_RELIABILITY,
            'Message Text' = (optional, any valid message text),
            'Notify Type' = ACK_NOTIFICATION,
            'To State' = FAULT
    ELSE
        BEFORE Notification Fail Time
            RECEIVE ConfirmedEventNotification-Request,
                'Process Identifier' = (PI3),
                'Initiating Device Identifier' = IUT,
                'Event Object Identifier' = (the event-generating object configured for this test),
                'Time Stamp' = (Faultthe IUT's current time or sequence number),
                'Notification Class' = (the class corresponding to the object being tested),
                'Priority' = (Pfault),
                'Event Type' = (any valid event type),
                'Notify Type' = ACK_NOTIFICATION
            TRANSMIT BACnet-SimpleACK-PDU
            VERIFY Acked_Transitions = (FALSE, TRUE, FALSE)
        }
19. TRANSMIT AcknowledgeAlarm-Request,
    'Acknowledging Process Identifier' = (PI2),
    'Event Object Identifier' = (the event-generating object configured for this test),
    'Event State Acknowledged' = NORMAL,
    'Time Stamp' = (Tnormal),
    'Acknowledgement Source' = (a character string),
    'Time of Acknowledgment' = (the TD's current time)
20. RECEIVE BACnet-SimpleACK-PDU
21. IF (Protocol_Revision is present and Protocol_Revision ≥ 1) THEN
    BEFORE Notification Fail Time
        RECEIVE ConfirmedEventNotification-Request,
            'Process Identifier' = (PI2),
            'Initiating Device Identifier' = IUT,
            'Event Object Identifier' = (the event-generating object configured for this test),
            'Time Stamp' = (Tnormalthe IUT's current time or sequence number),
            'Notification Class' = (the class corresponding to the object being tested),
            'Priority' = (Pnormal),
            'Event Type' = (any valid event type),
            'Notify Type' = ACK_NOTIFICATION,
            'To State' = NORMAL
    ELSE
        BEFORE Notification Fail Time
            RECEIVE ConfirmedEventNotification-Request,

```

```

        'Process Identifier' = (PI2),
        'Initiating Device Identifier' = IUT,
        'Event Object Identifier' = (the event-generating object configured for this test),
        'Time Stamp' = (Tnormal the IUT's current time or sequence number),
        'Notification Class' = (the class corresponding to the object bind tested),
        'Priority' = (Pnormal),
        'Event Type' = (any valid event type),
        'Message Text' = (optional, any valid message text),
        'Notify Type' = ACK_NOTIFICATION
22. TRANSMIT BACnet-SimpleACK-PDU
23. VERIFY Acked_Transitions = (FALSE, TRUE, TRUE)
24. TRANSMIT AcknowledgeAlarm-Request,
    'Acknowledging Process Identifier' = (PI1),
    'Event Object Identifier' = (the event-generating object configured for this test),
    'Event State Acknowledged' = OFFNORMAL,
    'Time Stamp' = (Toffnormal),
    'Acknowledgement Source' = (a character string),
    'Time of Acknowledgment' = (the TD's current time)
25. RECEIVE BACnet-SimpleACK-PDU
26. IF (Protocol_Revision is present and Protocol_Revision ≥ 1) THEN
    BEFORE Notification Fail Time
    RECEIVE ConfirmedEventNotification-Request,
        'Process Identifier' = (PI1),
        'Initiating Device Identifier' = IUT,
        'Event Object Identifier' = (the event-generating object configured for this test),
        'Time Stamp' = (Toffnormal the IUT's current time or sequence number),
        'Notification Class' = (the class corresponding to the object being tested),
        'Priority' = (Poffnormal),
        'Event Type' = (any valid event type),
        'Message Text' = (optional, any valid message text),
        'Notify Type' = ACK_NOTIFICATION,
        'To State' = OFFNORMAL
    ELSE
    BEFORE Notification Fail Time
    RECEIVE ConfirmedEventNotification-Request,
        'Process Identifier' = (PI1),
        'Initiating Device Identifier' = IUT,
        'Event Object Identifier' = (the event-generating object configured for this test),
        'Time Stamp' = (Toffnormal the IUT's current time or sequence number),
        'Notification Class' = (the class corresponding to the object being tested),
        'Priority' = (Poffnormal),
        'Event Type' = (any valid event type),
        'Message Text' = (optional, any valid message text),
        'Notify Type' = ACK_NOTIFICATION
27. TRANSMIT BACnet-SimpleACK-PDU
28. VERIFY Acked_Transitions = (TRUE, TRUE, TRUE)

```

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

[Add new **Clause 7.3.1.11.2**, p.40]

### 7.3.1.11.2 Acked\_Transitions Test for Latching Objects

Purpose: To verify that the Acked\_Transitions property tracks the acknowledgment state for a transition type.

Test Concept: This test is a single transition test for latching life safety objects which are not able to perform the regular Acked\_Transitions test for all transitions. An object, O1, in the IUT is made to generate a transition which requires an

acknowledgement. The Acked\_Transitions property is verified that the corresponding flag is cleared (set to FALSE). The transition is acknowledged, and the flag is verified to have been set back to TRUE.

Configuration Requirements: O1 is configured to generate events and to require acknowledgements for the transition being tested. O1 should have no event transitions which have outstanding acknowledgements.

Notes to Tester: The UnconfirmedEventNotification service may be substituted for the ConfirmedEventNotification service, in which case the TD shall skip sending the BACnet-SimpleACK-PDU messages after receiving the notifications.

Test Steps:

1. VERIFY Acked\_Transitions = (TRUE, TRUE, TRUE)
2. MAKE (O1 transition)
3. WAIT (pTimeDelay)
4. **BEFORE Notification Fail Time**  
RECEIVE ConfirmedEventNotification-Request,  
'Process Identifier' = (PI1: any valid process ID),  
'Initiating Device Identifier' = IUT,  
'Event Object Identifier' = O1,  
'Time Stamp' = (T1: any valid time stamp),  
'Notification Class' = (NC1: the class corresponding to the object being tested),  
'Priority' = (PRIO1: the value configured to correspond to the transition type),  
'Event Type' = (E1: any valid event type),  
'Message Text' = (optional, any valid message text),  
'Notify Type' = (the notify type configured for this event),  
'AckRequired' = TRUE,  
'From State' = S1,  
'To State' = S2,  
'Event Values' = (values appropriate to the event type)
5. TRANSMIT BACnet-SimpleACK-PDU
6. VERIFY pCurrentState = S2
7. IF S2 is NORMAL THEN  
VERIFY Acked\_Transitions = (TRUE, TRUE, FALSE)  
VERIFY pStatusFlags = (FALSE, FALSE, ?, ?)  
ELSE IF S2 is FAULT THEN  
VERIFY Acked\_Transitions = (TRUE, FALSE, TRUE)  
VERIFY pStatusFlags = (TRUE, TRUE, ?, ?)  
ELSE  
VERIFY Acked\_Transitions = (FALSE, TRUE, TRUE)  
VERIFY pStatusFlags = (TRUE, FALSE, ?, ?)
8. TRANSMIT AcknowledgeAlarm-Request,  
'Acknowledging Process Identifier' = PI1,  
'Event Object Identifier' = O1,  
'Event State Acknowledged' = S2,  
'Time Stamp' = T1,  
'Acknowledgement Source' = (a character string),  
'Time of Acknowledgment' = (the TD's current time)
9. RECEIVE BACnet-SimpleACK-PDU
10. IF (Protocol\_Revision is present and Protocol\_Revision ≥ 1) THEN  
**BEFORE Notification Fail Time**  
RECEIVE ConfirmedEventNotification-Request,  
'Process Identifier' = PI1,  
'Initiating Device Identifier' = IUT,  
'Event Object Identifier' = O1,  
'Time Stamp' = the IUT's current time or sequence number,  
'Notification Class' = NC1,  
'Priority' = PRIO1,  
'Event Type' = E1,  
'Message Text' = (optional, any valid message text),  
'Notify Type' = ACK\_NOTIFICATION,

```

        'To State' =                S2
ELSE
    BEFORE Notification Fail Time
        RECEIVE ConfirmedEventNotification-Request,
            'Process Identifier' =    PI1,
            'Initiating Device Identifier' = IUT,
            'Event Object Identifier' = O1,
            'Time Stamp' =           the IUT's current time or sequence number,
            'Notification Class' =    NC1,
            'Priority' =              PRIO1,
            'Event Type' =           E1,
            'Message Text' =         (optional, any valid message text),
            'Notify Type' =          ACK_NOTIFICATION

```

11. TRANSMIT BACnet-SimpleACK-PDU

12. VERIFY Acked\_Transitions = (TRUE, TRUE, TRUE)

### 7.3.1.13 Limit\_Enable Tests

[Change **Clause 7.3.1.13.1**, p.41]

[Reviewer Note: The 'Event Type' is checked to be an out-of-range algorithm appropriate to the object type.]

#### 7.3.1.13.1 Limit\_Enable Test, LowLimitEnable

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

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

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

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

Test Steps:

1. MAKE pLimitEnable = (TRUE, ?)
2. VERIFY pCurrentState = NORMAL
3. MAKE (pMonitoredValue a value less than pLowLimit)
4. WAIT (pTimeDelay)
5. BEFORE Notification Fail Time
 

```

                RECEIVE ConfirmedEventNotification-Request,
                    'Process Identifier' =    (any valid process ID),
                    'Initiating Device Identifier' = IUT,
                    'Event Object Identifier' = (the object configured for this test),
                    'Time Stamp' =           (the current local time),
                    'Notification Class' =    (the class corresponding to the object being tested),
                    'Priority' =              (the value configured to correspond to a TO-OFFNORMAL transition),

```



'Event Type' = *(the algorithm appropriate to the object type i.e., OUT\_OF\_RANGE, SIGNED\_OUT\_OF\_RANGE, UNSIGNED\_OUT\_OF\_RANGE, or DOUBLE\_OUT\_OF\_RANGE),*  
 'Message Text' = (optional, any valid message text),  
 'Notify Type' = ALARM | EVENT,  
 'AckRequired' = TRUE | FALSE,  
 'From State' = NORMAL,  
 'To State' = LOW\_LIMIT,  
 'Event Values' = (values appropriate to the event type)

6. TRANSMIT BACnet-SimpleACK-PDU
7. VERIFY pCurrentState = LOW\_LIMIT
8. MAKE (pMonitoredValue a value that is between pLowLimit + pDeadband and pHighLimit)
9. WAIT (pTimeDelayNormal)
10. BEFORE Notification Fail Time

RECEIVE ConfirmedEventNotification-Request,

'Process Identifier' = (any valid process ID),  
 'Initiating Device Identifier' = IUT,  
 'Event Object Identifier' = (the object configured for this test),  
 'Time Stamp' = (the current local time),  
 'Notification Class' = (the class corresponding to the object being tested),  
 'Priority' = (the value configured to correspond to a TO-NORMAL transition),  
 'Event Type' = *(the algorithm appropriate to the object type i.e. OUT\_OF\_RANGE, SIGNED\_OUT\_OF\_RANGE, UNSIGNED\_OUT\_OF\_RANGE, or DOUBLE\_OUT\_OF\_RANGE,*  
 'Message Text' = (optional, any valid message text),  
 'Notify Type' = ALARM | EVENT,  
 'AckRequired' = TRUE | FALSE,  
 'From State' = LOW\_LIMIT,  
 'To State' = NORMAL,  
 'Event Values' = (values appropriate to the event type)

11. TRANSMIT BACnet-SimpleACK-PDU
12. MAKE pLimitEnable = (FALSE, ?)
13. VERIFY pCurrentState = NORMAL
14. MAKE (pMonitoredValue a value less than pLowLimit)
15. WAIT (pTimeDelay + Notification Fail Time)
16. CHECK (verify that no notification message was transmitted)
17. VERIFY pCurrentState = NORMAL

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

[Change **Clause 7.3.1.13.2**, p.42]

[Reviewer Note: The 'Event Type' is checked to be an out-of-range algorithm appropriate to the object type.]

### 7.3.1.13.2 Limit\_Enable Test, HighLimitEnable

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

Test Concept: The HighLimitEnable flag is set to true in the Limit\_Enable property and the event-triggering property is manipulated to cause the high limit to be exceeded. This should generate an event notification and make Event\_State = High\_Limit. After the event-triggering property is returned to a normal value, the HighLimitEnable flag is set to false and the event-triggering property is again manipulated to exceed the high limit. No event notification should be observed and the Event\_State must have a value of normal.

Configuration Requirements: Configure the object with pHighLimit, pLowLimit and pDeadband values such that pHighLimit - pDeadband > pLowLimit and both the pLowLimit and pHighLimit values are within the valid range of values for the event triggering property. If the device cannot be configured with limit values that meet these conditions, then this test shall be

ANSI/ASHRAE Addendum s to ANSI/ASHRAE Standard 135.1-2019

skipped. The Event\_Enable property shall be set to (TRUE, ?, TRUE) for this test. If the Event\_Enable property cannot be configured such that the TO-NORMAL and the TO-OFFNORMAL transitions are TRUE, this test shall be skipped.

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

#### Test Steps:

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

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

[Change **Clause 7.3.1.17**, p.46]

[Reviewer Note: Removed the allowance for the test to be skipped.]

### 7.3.1.17 Event\_Message\_Texts Tests

Purpose: To verify that the value of the Event\_Message\_Texts property is updated when an object generates an event notification.

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

Configuration Requirements: The IUT shall be configured with an event-generation object, O1 which shall be in a NORMAL Event\_State at the beginning of the test, *and if Event\_Enable is configurable, it shall have all bits set to TRUE for which the object supports transitions. If the algorithm of the object does not support NORMAL to NORMAL transitions, then the TO\_OFFNORMAL bit of the Event\_Enable shall be TRUE. If the IUT does not contain any objects which can transition to any offnormal state, then this test shall be skipped.*

Test Steps:

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

```

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

```

### 7.3.1.20 Event\_Algorithm\_Inhibit\_Ref Tests

[Change Clause 7.3.1.20.1, p.52]

#### 7.3.1.20.1 Event\_Algorithm\_Inhibit\_Ref Test

Purpose: To verify that the object referenced by Event\_Algorithm\_Inhibit\_Ref controls Event\_Algorithm\_Inhibit and thus whether or not the event state detection algorithm is executed.

Test Concept: Execute test ~~7.3.1.X2.1~~ 7.3.1.19.1 against an object O2 which supports both Event\_Algorithm\_Inhibit\_Ref and Event\_Algorithm\_Inhibit and instead of writing Event\_Algorithm\_Inhibit, write the property referenced by Event\_Algorithm\_Inhibit\_Ref to change the value in the Event\_Algorithm\_Inhibit property.

Configuration Requirements: If the IUT has no object *which has an in which the Event\_Algorithm\_Inhibit\_Ref property is absent or can be made uninitialized, or has no object in which Event\_Detection\_Enable can be made TRUE*, this test shall be skipped.

[Change Clause 7.3.1.20.2, p.52]

#### 7.3.1.20.2 Event\_Algorithm\_Inhibit Writable Test

Purpose: To verify that *whenever if* the Event\_Algorithm\_Inhibit\_Ref property is absent or is uninitialized then the Event\_Algorithm\_Inhibit property shall be writable.

Configuration Requirements: Select an event-initiating object, O1 *which has an Event\_Algorithm\_Inhibit property, but in which Event\_Algorithm\_Inhibit\_Ref property is absent or is uninitialized*. If the IUT has no such object, this test shall be skipped.

Test Steps:

1. WRITE Event\_Algorithm\_Inhibit = TRUE
2. WRITE Event\_Algorithm\_Inhibit = FALSE

### 7.3.1.21 Reliability\_Evaluation\_Inhibit Tests

[Change Clause 7.3.1.21.1, p.52]

#### 7.3.1.21.1 Reliability\_Evaluation\_Inhibit Test

Purpose: To verify that Reliability\_Evaluation\_Inhibit controls whether or not fault conditions are detected.

Test Concept: Select an event generating object, O1, which supports the Reliability\_Evaluation\_Inhibit property. With Reliability\_Evaluation\_Inhibit FALSE, make a fault condition exist. Verify that Reliability changes and, *if event reporting is supported*, that a notification is generated. Set Reliability\_Evaluation\_Inhibit to TRUE. Verify that the Reliability changes to NO\_FAULT\_DETECTED and, *if event reporting is supported*, that a TO\_NORMAL notification is generated. Remove the fault condition and ensure that no notification is generated. Make a fault condition exist and verify that Reliability remains NO\_FAULT\_DETECTED, and that no notification is generated.

Test Configuration: O1 is configured to detect and, *if event reporting is supported*, report unconfirmed events, is in the NORMAL state, and Reliability\_Evaluation\_Inhibit equals FALSE, so that reliability evaluation for that object is configured to detect fault conditions. If no object exists in the IUT for which fault conditions can be generated then this test shall be skipped.

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

Test Steps:

1. VERIFY pCurrentState = NORMAL
2. VERIFY Reliability = NO\_FAULT\_DETECTED
3. MAKE(~~a condition exist that would cause O1 to generate a TO\_FAULT transition~~ a fault condition exist for O1)
4. IF the IUT supports event reporting THEN
  - BEFORE Notification Fail Time
  - RECEIVE UnconfirmedEventNotification-Request
 

'Process Identifier' =	(the value configured for the transition),
'Initiating Device Identifier' =	IUT,
'Event Object Identifier' =	O1,
'Time Stamp' =	(any valid timestamp),
'Priority' =	(any valid priority),
'Event Type' =	CHANGE_OF_RELIABILITY,
'Message Text' =	(optional, any valid message text),
'Notify Type' =	ALARM   EVENT,
'AckRequired' =	TRUE   FALSE,
'From State' =	NORMAL,
'To State' =	FAULT,
'Event Values' =	(any values appropriate to CHANGE_OF_RELIABILITY)
  5. VERIFY Reliability <> NO\_FAULT\_DETECTED
  6. IF Reliability\_Evaluation\_Inhibit is writable THEN
    - WRITE Reliability\_Evaluation\_Inhibit = TRUE
    - ELSE

*MAKE(Reliability\_Evaluation\_Inhibit TRUE)*

7. *IF the IUT supports event reporting THEN*

**BEFORE Internal Processing Fail Time + Notification Fail Time**

RECEIVE UnconfirmedEventNotification-Request

'Process Identifier' = (the value configured for the transition),

'Initiating Device Identifier' = IUT,

'Event Object Identifier' = O1,

'Time Stamp' = (any valid timestamp),

'Priority' = (any valid priority),

'Event Type' = CHANGE\_OF\_RELIABILITY,

'Message Text' = (optional, any valid message text),

'Notify Type' = ALARM | EVENT,

'AckRequired' = TRUE | FALSE,

'From State' = FAULT,

'To State' = NORMAL,

'Event Values' = (any values appropriate to CHANGE\_OF\_RELIABILITY)

8. VERIFY Reliability = NO\_FAULT\_DETECTED

9. VERIFY pCurrentState = NORMAL

10. MAKE(remove the fault condition)

11. WAIT(pTimeDelayNormal)

12. WAIT **Notification Fail Time**

13. CHECK (that the IUT did not send any event notifications for O1)

14. *VERIFY Reliability = NO\_FAULT\_DETECTED*

15. ~~MAKE(a condition exist that would cause O1 to generate a TO\_NORMAL transition)~~ *MAKE(a fault condition exist for O1)*

16. WAIT **Notification Fail Time**

17. VERIFY Reliability = NO\_FAULT\_DETECTED

18. VERIFY pCurrentState = NORMAL

19. CHECK (that the IUT did not send any event notifications for O1)

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

[Add new **Clause 7.3.1.X16**, p. 55]

[Reviewer Note: 135.1-2019 has a test case using WriteProperty service. Added a new test case using WritePropertyMultiple service.]

### **7.3.1.X16 Array Resizing Test using WritePropertyMultiple Service**

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

Test Concept: The resizable array property P1 of object O1 is written with WritePropertyMultiple 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 WritePropertyMultiple operations.

Test Steps:

1. TRANSMIT WritePropertyMultiple-Request,  
'Object Identifier' = O1,  
'Property Identifier' = P1,  
'Property Value' = (array A1 of non-zero size N1)
2. RECEIVE BACnet-SimpleACK-PDU
3. VERIFY P1 = (array A1), ARRAY INDEX = 0, (array size i.e. N1)

--Resize the array to make it smaller in size

4. TRANSMIT WritePropertyMultiple-Request,  
'Object Identifier' = O1,

'Property Identifier' = P1

'Property Value' = (array A2 of non-zero size N2, where  $N2 \leq N1$ )

5. RECEIVE BACnet-SimpleACK-PDU

6. VERIFY P1 = (array A2), ARRAY INDEX = 0, (array size N2)

--Resize the array to make it larger in size

7. TRANSMIT WritePropertyMultiple-Request,

'Object Identifier' = O1,

'Property Identifier' = P1

'Property Value' = (array A3 of non-zero size N3, where  $N3 \geq N1$ ),

8. RECEIVE BACnet-SimpleACK-PDU

9. VERIFY P1 = (array A3), ARRAY INDEX = 0, (array size N3)

--Modify the existing content of element

10. TRANSMIT WritePropertyMultiple-Request,

'Object Identifier' = O1,

'Property Identifier' = P1,

'Property Value' = (array A4 of non-zero unsigned value N4, where  $N4 \leq N1$ ),

11. RECEIVE BACnet-SimpleACK-PDU

12. VERIFY P = (array A4), ARRAY INDEX = 0, (array size N4)

--Resize the array by writing the size of the array

13. TRANSMIT WritePropertyMultiple-Request

'Object Identifier' = O1,

'Property Identifier' = P1

'Property Value' = (N5, where  $N5 \geq N4$ ),

'Property Array Index' = 0,

14. RECEIVE BACnet-SimpleACK-PDU

15. VERIFY (array contains unchanged first N4 elements of the array written in step 10, plus N5-N4 additional elements, initialized to particular values for the array property being tested)

16. VERIFY P1, ARRAY INDEX = 0, (array size N5)

--Try to add the array element at Array Index which is greater than the size of the array

17. TRANSMIT WritePropertyMultiple-Request,

'Object Identifier' = O1,

'Property Identifier' = P1,

'Property Value' = (one array element),

'Property Array Index' = (N6, where  $N6 \geq N5$ ),

18. RECEIVE WritePropertyMultiple-Error

'Error Class' = PROPERTY,

'Error Code' = INVALID\_ARRAY\_INDEX

'Object Identifier' = O1,

'Property Identifier' = P1,

'Property Array Index' = N6

19. VERIFY (array is unchanged from step 15)

--Resize the array to size zero

20. IF (the array can be resized to have zero elements) THEN

TRANSMIT WritePropertyMultiple-Request,

'Object Identifier' = O1,

'Property Identifier' = P1,

'Property Value' = (empty array),

BACnet-SimpleACK-PDU

21. VERIFY P1 = (array is empty), ARRAY INDEX = 0, (array size is zero)

[Add new **Clause 7.3.1.X18**, p. 55]

[Reviewer Note: Additional behavior was specified in Protocol Revision 16 (135-2012az-1 and 135-2012az-2) when the Change\_of\_State\_Count property accepts writes of non-zero values.]

### **7.3.1.X18 Non-zero Writable State Count Test**

Purpose: To verify that the properties of objects that count the number of transitions and the time when that number of the transitions tracking started function properly.

Test Concept: The Change\_of\_State\_Count property is set with a non-zero value. The Time\_Of\_State\_Count\_Reset property is checked to verify that it has not been updated. The Time\_Of\_State\_Count\_Reset property is then checked to be writable.

Configuration Requirements: The object being tested shall contain Change\_of\_State\_Count and Time\_Of\_State\_Count\_Reset properties, and its Change\_of\_State\_Count property must accept writes of non-zero values.

Test Steps:

1. READ TSCR = Time\_Of\_State\_Count\_Reset
2. WRITE Change\_of\_State\_Count = (a value > 0)
3. VERIFY Time\_Of\_State\_Count\_Reset = TSCR
4. WRITE Time\_Of\_State\_Count\_Reset = (T1: any valid value)
5. VERIFY Time\_Of\_State\_Count\_Reset = T1

[Add new **Clause 7.3.1.X19**, p. 55]

[Reviewer Note: Additional behavior was specified in Protocol Revision 16 (135-2012az-1 and 135-2012az-2) when the Elapsed\_Active\_Time property accepts writes of non-zero values.]

### **7.3.1.X19 Non-zero Writable Elapsed Active Time Test**

Purpose: To verify that Time\_Of\_Active\_Time\_Reset is writable when Elapsed\_Active\_Time accepts writes of non-zero values and does not automatically change when Elapsed\_Active\_Time is written to a non-zero value.

Test Concept: The Present\_Value or Feedback\_Value is made INACTIVE and the Elapsed\_Active\_Time is set with a non-zero value. The Time\_Of\_Active\_Time\_Reset property is checked to verify that it has not been updated. The Time\_Of\_Active\_Time\_Reset property is then checked to be writable.

Configuration Requirements: The object being tested shall be chosen in which Elapsed\_Active\_Time and Time\_Of\_Active\_Time\_Reset properties are present, and in which the Elapsed\_Active\_Time property accepts writes of non-zero values.

Test Steps:

1. IF (Present\_Value is writable) THEN  
    WRITE Present\_Value = INACTIVE  
    VERIFY Present\_Value = INACTIVE  
ELSE  
    MAKE (Present\_Value = INACTIVE)
2. READ TATR = Time\_Of\_Active\_Time\_Reset
3. WRITE Elapsed\_Active\_Time = a supported non-zero value
4. VERIFY Time\_Of\_Active\_Time\_Reset = TATR
5. WRITE Time\_Of\_Active\_Time\_Reset = (T1: any valid value)
6. VERIFY Time\_Of\_Active\_Time\_Reset = T1

[Add new **Clause 7.3.1.X20**, p. 55]

[Reviewer Note: Additional behavior was specified in Protocol Revision 16 (135-2012az-1) when the Strike\_Count property accepts writes of non-zero values.]



### 7.3.1.X20 Strike Count Tests

#### 7.3.1.X20.1 Non-zero Writable Strike Count Test

Purpose: To verify that Time\_Of\_Count\_Reset is writable when Strike\_Count accepts writes of non-zero values and does not automatically change when Strike\_Count is written to a non-zero value.

Test Concept: The Strike\_Count property is set with a non-zero value. The Time\_Of\_Strike\_Count\_Reset property is checked to verify that it has not been updated. The Time\_Of\_Strike\_Count\_Reset property is then checked to be writable.

Configuration Requirements: The object being tested shall be chosen in which Strike\_Count and Time\_Of\_Strike\_Count\_Reset properties are present, and in which the Strike\_Count property accepts writes of non-zero values.

Test Steps:

1. READ TSCR = Time\_Of\_Strike\_Count\_Reset
2. WRITE Strike\_Count = (a value > 0)
3. VERIFY Time\_Of\_Strike\_Count\_Reset = TSCR
4. WRITE Time\_Of\_Strike\_Count\_Reset = (T1: any valid value)
5. VERIFY Time\_Of\_Strike\_Count\_Reset = T1

#### 7.3.1.X20.2 Strike Count Test

Purpose: To verify that the properties of an object (O1) that tracks strike counts.

Test Concept: The Present\_Value or Feedback\_Value of O1 can be used as the source S1 to increment Strike\_Count. S1 is transitioned from OFF to ON. The Strike\_Count property is checked to verify that it has been incremented. The Strike\_Count is reset and Time\_Of\_Strike\_Count\_Reset is checked to verify that it has been updated appropriately. Strike\_Count is set to a non-zero value and the Time\_Of\_Strike\_Count\_Reset is unchanged.

Configuration Requirements: O1 shall be configured such that the Present\_Value property is writable, or another means of changing these properties shall be provided.

Test Steps:

1. C1 = Strike\_Count
2. MAKE (S1 transition OFF to ON)
3. VERIFY (Strike\_Count = C1 + 1)
4. IF (Strike\_Count is writable) THEN  
    MAKE (Strike\_Count = 0)  
    VERIFY (Time\_Of\_Strike\_Count\_Reset = current local time)
5. IF (Strike\_Count is writable to a non-zero value) THEN  
    MAKE (Strike\_Count > 0)  
    VERIFY (Time\_Of\_Strike\_Count\_Reset is unchanged)

[Add new **Clause 7.3.1.X41**, p. 55]

### 7.3.1.X41 Blink Warn Tests

#### 7.3.1.X41.Y1 Blink-Warn WARN Command Test

Purpose: To verify the correct operation of the blink-warn WARN command.

Test Concept: Select an object O1 that supports blink-warn WARN command. Ensure O1 is not in egress mode and the specific properties have been configured to support blink-warn. Execute blink-warn WARN command by writing C1 to PROP\_REF at a priority PTY1 of O1 and validate the specified blink-warn command functions correctly. Validate the Priority\_Array value at priority PTY1 remains.

Configuration Requirements: O1 shall be configured such that all slots in the Priority\_Array numerically less than PTY1 have a value of NULL and no internal algorithms are issuing commands to O1 at a priority numerically less than or equal to PTY1. The Priority\_Array at PTY1 has a value V1, Blink\_Warn\_Enable is TRUE, Egress\_Active is FALSE.

	<b>Binary Lighting Output object</b>	<b>Lighting Output object</b>
<b>PROP_REF</b>	<b>Present_Value</b>	<b>Present_Value or Lighting_Command</b>
<b>C1</b>	<b>WARN</b>	<b>-1.0 if PROP_REF = Present_Value, otherwise WARN</b>
<b>V1</b>	<b>ON</b>	<b>&gt;1.0</b>

Test Steps:

1. VERIFY Priority\_Array = V1, ARRAY INDEX = PTY1
2. VERIFY Blink\_Warn\_Enable = TRUE
3. VERIFY Egress\_Active = FALSE
4. WRITE PROP\_REF = C1, PRIORITY = PTY1
5. BEFORE **Internal Processing Fail Time**  
CHECK (blink-warn occurred)
6. VERIFY Egress\_Active = FALSE
7. VERIFY Priority\_Array = V1, ARRAY INDEX = PTY1

### 7.3.1.X41.Y2 Blink-Warn WARN\_OFF Command Test

Purpose: To verify the correct operation of the blink-warn WARN\_OFF command.

Test Concept: Select an object O1 that supports blink-warn commands. Ensure O1 is not in egress mode and the specific properties have been configured to support blink-warn. Execute blink-warn WARN\_OFF command by writing C1 to PROP\_REF at a priority PTY1 of O1 and validate the specified blink-warn command functions correctly. Validate the Priority\_Array value at priority PTY1 after Egress\_Time seconds has elapsed.

Configuration Requirements: O1 shall be configured such that all slots in the Priority\_Array numerically less than PTY1 have a value of NULL and no internal algorithms are issuing commands to O1 at a priority numerically less than or equal to PTY1. The Priority\_Array at PTY1 has a value V1, Blink\_Warn\_Enable is TRUE, Egress\_Active is FALSE, and Egress\_Time is a non-zero value.

	<b>Binary Lighting Output object</b>	<b>Lighting Output object</b>
PROP_REF	Present Value	Present Value or Lighting Command
C1	WARN OFF	-3.0 if PROP_REF = Present Value, otherwise WARN OFF
V1	ON	>1.0
V2	OFF	0.0

Test Steps:

1. VERIFY Priority\_Array = V1, ARRAY INDEX = PTY1
2. VERIFY Blink\_Warn\_Enable = TRUE
3. VERIFY Egress\_Time > 0
4. VERIFY Egress\_Active = FALSE
5. WRITE PROP\_REF = C1, PRIORITY = PTY1
6. T1 = current local time
7. BEFORE **Internal Processing Fail Time**  
CHECK (blink-warn occurred)
8. WHILE (Egress\_Active = TRUE)  
VERIFY Priority\_Array = V1, ARRAY INDEX = PTY1
9. T2 = current local time
10. VERIFY Egress\_Time  $\approx$  (T1 - T2) +/- **Internal Processing Fail Time**
11. VERIFY Priority\_Array = V2, ARRAY INDEX = PTY1

### 7.3.1.X41.Y3 Blink-Warn WARN\_RELINQUISH Command Test

Purpose: To verify the correct operation of the blink-warn WARN\_RELINQUISH commands.

Test Concept: Select an object O1 that supports blink-warn commands. Ensure O1 is not in egress mode and the specific properties have been configured to support blink-warn. Execute blink-warn WARN\_RELINQUISH command by writing C1 to PROP\_REF at a priority PTY1 of O1 and validate the specified blink-warn command functions correctly. Validate the Priority\_Array value at priority PTY1 after Egress\_Time seconds has elapsed.

Configuration Requirements: O1 shall be configured such that all slots in the Priority\_Array numerically less than PTY1 shall have a value of NULL, at least one slot numerically greater than PTY1 or Relinquish\_Default shall have a value of V2, and all other slots numerically greater than PTY1 shall have a value of V0. No internal algorithms are issuing commands to O1 at any priority. The Priority\_Array at PTY1 has a value V1, Blink\_Warn\_Enable is TRUE, Egress\_Time is a non-zero value, and Egress\_Active is FALSE.

	Binary Lighting Output object	Lighting Output object
PROP_REF	Present Value	Present Value or Lighting Command
C1	WARN_RELINQUISH	-2.0 if PROP_REF = Present Value, otherwise WARN_OFF
V0	NULL or OFF	NULL or 0.0
V1	ON	>1.0
V2	OFF	0.0

Test Steps:

1. VERIFY Priority\_Array = V1, ARRAY INDEX = PTY1
2. VERIFY Blink\_Warn\_Enable = TRUE
3. VERIFY Egress\_Time > 0
4. VERIFY Egress\_Active = FALSE
5. WRITE PROP\_REF = C1, PRIORITY = PTY1
6. T1 = current local time
7. BEFORE **Internal Processing Fail Time**  
CHECK (blink-warn occurred)
8. WHILE (Egress\_Active = TRUE)  
VERIFY Priority\_Array = V1, ARRAY INDEX = PTY1
9. T2 = current local time
10. VERIFY Egress\_Time  $\approx$  (T1 - T2) +/- **Internal Processing Fail Time**
11. VERIFY Priority\_Array = NULL, ARRAY INDEX = PTY1

### 7.3.1.X41.Y4 Blink-Warn STOP Command Test

Purpose: To verify the correct operation of the blink-warn STOP command.

Test Concept: Select an object O1 that supports blink-warn commands. Ensure O1 is not in egress mode and the specific properties have been configured to support blink-warn. Execute blink-warn command by writing C1 to PROP\_REF at a priority PTY1 of O1 and validate that blink-warn occurs. Before the Egress\_Time times out, STOP the egress process and validate the Priority\_Array value at PTY1 remains equal to V1 after Egress\_Time.

Configuration Requirements: O1 shall be configured such that all slots in the Priority\_Array numerically less than PTY1 shall have a value of NULL, at least one slot numerically greater than PTY1 or Relinquish\_Default shall have a value of V2, and all other slots numerically greater than PTY1 shall have a value of V0. No internal algorithms are issuing commands to O1 at a priority numerically less than or equal to PTY1. The Priority\_Array at PTY1 has a value V1, Blink\_Warn\_Enable is TRUE, Egress\_Time is a non-zero value, and Egress\_Active is FALSE.

	Binary Lighting Output object	Lighting Output object
PROP_REF	Present Value	Lighting Command
C1	WARN_RELINQUISH or WARN_OFF	WARN_RELINQUISH or WARN_OFF

V0	NULL or OFF	NULL or 0.0
V1	ON	>1.0
V2	OFF	0.0

Test Steps:

1. VERIFY Priority\_Array = V1, ARRAY INDEX = PTY1
2. VERIFY Blink\_Warn\_Enable = TRUE
3. VERIFY Egress\_Time > 0
4. VERIFY Egress\_Active = FALSE
5. WRITE PROP\_REF = C1, PRIORITY = PTY1
6. T1 = current local time
7. BEFORE **Internal Processing Fail Time**  
CHECK (blink-warn occurred)
8. VERIFY Egress\_Active = TRUE
9. WAIT less than Egress\_Time  
WRITE PROP\_REF = STOP, PRIORITY = PTY1
10. T2 = current local time
11. WAIT **Internal Processing Fail Time**
12. VERIFY Egress\_Active = FALSE
13. WAIT Egress\_Time – (T2 – T1) + **Internal Processing Fail Time**
14. VERIFY Priority\_Array = V1, ARRAY INDEX = PTY1

### 7.3.1.X41.Y5 Blink-Warn WARN Command Failure Test

Purpose: To verify blink-warn WARN command does not occur when, the specified priority is not the highest active priority, the value at the specified priority is off or Blink\_Warn\_Enable is FALSE.

Test Concept: Select an object O1 that supports blink-warn commands. Configure O1 such that a blink-warn command would generate a blink-warn except set the specified failure conditions. Verify blink-warn does not occur and the Priority\_Array is not affected.

Configuration Requirements: O1 shall be configured such that all slots in the Priority\_Array numerically less than PTY1 have a value of NULL and no internal algorithms are issuing commands to O1 at a priority numerically less than or equal to PTY1. Select a priority, PTY2, which is numerically less than PTY1 and not equal to 6. Blink\_Warn\_Enable is TRUE, Egress\_Active is FALSE.

	Binary Lighting Output object	Lighting Output object
PROP_REF	Present Value	Present Value or Lighting Command
C1	WARN	-1.0 if PROP_REF = Present Value, otherwise WARN
V1, V2	ON	>1.0
V3	OFF	0.0

Test Steps:

-- Test for the specified priority is not the highest active priority

1. VERIFY Blink\_Warn\_Enable = TRUE
2. WRITE Present\_Value = V1, PRIORITY = PTY1
3. VERIFY Egress\_Active = FALSE
4. WRITE Present\_Value = V2, PRIORITY = PTY2
5. WRITE PROP\_REF = C1, PRIORITY = PTY1
6. WAIT **Internal Processing Fail Time**  
CHECK (blink-warn did not occur)
7. VERIFY Egress\_Active = FALSE
8. VERIFY Priority\_Array = V1, ARRAY INDEX = PTY1
9. WRITE Present\_Value = NULL, PRIORITY = PTY2

-- Test for the value at the specified priority is either OFF or 0.0

10. WRITE Present\_Value = V3, PRIORITY = PTY1
11. WRITE PROP\_REF = C1, PRIORITY = PTY1

12. **WAIT Internal Processing Fail Time**  
CHECK (blink-warn did not occur)
13. VERIFY Egress\_Active = FALSE
14. VERIFY Priority\_Array = V3, ARRAY INDEX = PTY1
15. WRITE Present\_Value = V1, PRIORITY = PTY1

-- Test for Blink\_Warn\_Enable is FALSE

16. IF (Blink\_Warn\_Enable is writable) THEN  
WRITE Blink\_Warn\_Enable = FALSE  
WRITE PROP\_REF = C1, PRIORITY = PTY1  
**WAIT Internal Processing Fail Time**  
CHECK (blink-warn did not occur)  
VERIFY Egress\_Active = FALSE  
VERIFY Priority\_Array = V1, ARRAY INDEX = PTY1

### 7.3.1.X41.Y6 Blink-Warn WARN\_OFF Command Failure Test

Purpose: To verify blink-warn WARN\_OFF command does not occur when the specified priority is not the highest active priority, the Present\_Value is either 0.0 or OFF, or Blink\_Warn\_Enable is FALSE.

Test Concept: Select an object O1 that supports blink-warn commands. Configure O1 such that a blink-warn command would generate a blink-warn except set the specified failure conditions. Verify blink-warn does not occur and the Priority\_Array is correctly changed.

Configuration Requirements: O1 shall be configured such that all slots in the Priority\_Array numerically less than PTY1 have a value of NULL and no internal algorithms are issuing commands to O1 at a priority numerically less than or equal to PTY1. Blink\_Warn\_Enable is TRUE, Egress\_Time is a non-zero value and Egress\_Active is FALSE.

	Binary Lighting Output object	Lighting Output object
PROP_REF	Present_Value	Present_Value or Lighting_Command
C1	WARN_OFF	-3.0 if PROP_REF = Present_Value, otherwise WARN_OFF
V1, V2	ON	>1.0
V3	OFF	0.0

Test Steps:

-- Test for the specified priority is not the highest active priority

1. VERIFY Blink\_Warn\_Enable = TRUE
2. VERIFY Egress\_Time > 0
3. WRITE Present\_Value = V1, PRIORITY = PTY1
4. VERIFY Egress\_Active = FALSE
5. WRITE Present\_Value = V2, PRIORITY = PTY2, a value not equal to 6 and less than PTY1
6. WRITE PROP\_REF = C1, PRIORITY = PTY1
7. **WAIT Internal Processing Fail Time**  
CHECK (blink-warn did not occur)
8. VERIFY Egress\_Active = FALSE
9. VERIFY Priority\_Array = V3, ARRAY INDEX = PTY1
10. WRITE Present\_Value = V1, PRIORITY = PTY1

-- Test for the Present\_Value is OFF or 0.0

11. WRITE Present\_Value = V3, PRIORITY = PTY2, a value not equal to 6 and less than PTY1
12. WRITE PROP\_REF = C1, PRIORITY = PTY1
13. **WAIT Internal Processing Fail Time**  
CHECK (blink-warn did not occur)
14. VERIFY Egress\_Active = FALSE
15. VERIFY Priority\_Array = V3, ARRAY INDEX = PTY1
16. WRITE Present\_Value = NULL, PRIORITY = PTY2
17. WRITE Present\_Value = V1, PRIORITY = PTY1

-- Test for Blink\_Warn\_Enable is FALSE

```

18. IF (Blink_Warn_Enable is writable) THEN
    WRITE Blink_Warn_Enable = FALSE
    WRITE PROP_REF = C1, PRIORITY = PTY1
    WAIT Internal Processing Fail Time
    CHECK (blink-warn did not occur)
    VERIFY Egress_Active = FALSE
    VERIFY Priority_Array = V3, ARRAY INDEX = PTY1

```

### 7.3.1.X41.Y7 Blink-Warn WARN\_RELINQUISH Command Failure Test

Purpose: To verify blink-warn WARN\_RELINQUISH command does not occur when the specified priority is not the highest active priority, the value at the specified priority is V0, the value of the next highest non-NULL priority, including Relinquish\_Default, is V1, or Blink\_Warn\_Enable is FALSE.

Test Concept: Select an object O1 that supports blink-warn commands. Configure O1 such that a blink-warn command would generate a blink-warn except set the specified failure conditions. Verify blink-warn does not occur and the Priority\_Array is correctly changed.

Configuration Requirements: O1 shall be configured such that all slots in the Priority\_Array numerically less than PTY1 shall have a value of NULL, at least one slot numerically greater than PTY1 or Relinquish\_Default shall have a value of V2, and all other slots numerically greater than PTY1 shall have a value of V0. No internal algorithms are issuing commands to O1 at any priority. Blink\_Warn\_Enable is TRUE, Egress\_Time is a non-zero value, Egress\_Active is FALSE.

	Binary Lighting Output object	Lighting Output object
PROP_REF	Present Value	Present Value or Lighting Command
C1	WARN_RELINQUISH	-2.0 if PROP_REF = Present_Value, otherwise WARN_RELINQUISH
V0	OFF or NULL	0.0 or NULL
V1	ON	>1.0
V2	OFF	0.0

#### Test Steps:

-- Test for the specified priority is not the highest active priority

1. VERIFY Blink\_Warn\_Enable = TRUE
2. VERIFY Egress\_Time > 0
3. WRITE Present\_Value = V1, PRIORITY = PTY1
4. VERIFY Egress\_Active = FALSE
5. WRITE Present\_Value = V1, PRIORITY = PTY2, a value not equal to 6 and less than PTY1
6. WRITE PROP\_REF = C1, PRIORITY = PTY1
7. WAIT **Internal Processing Fail Time**  
CHECK (blink-warn did not occur)
8. VERIFY Egress\_Active = FALSE
9. VERIFY Priority\_Array = NULL, ARRAY INDEX = PTY1
10. WRITE Present\_Value = NULL, PRIORITY = PTY2

-- Test for the value at the specified priority is OFF or 0.0

11. WRITE Present\_Value = V2 PRIORITY = PTY1
12. WRITE PROP\_REF = C1, PRIORITY = PTY1
13. WAIT **Internal Processing Fail Time**  
CHECK (blink-warn did not occur)
14. VERIFY Egress\_Active = FALSE
15. VERIFY Priority\_Array = NULL, ARRAY INDEX = PTY1

-- Test for the value at the specified priority is NULL

16. WRITE Present\_Value = NULL, PRIORITY = PTY1
17. WRITE PROP\_REF = C1, PRIORITY = PTY1
18. WAIT **Internal Processing Fail Time**  
CHECK (blink-warn did not occur)

```

19. VERIFY Egress_Active = FALSE
20. VERIFY Priority_Array = NULL, ARRAY INDEX = PTY1

-- Test for the value of the next highest non-NULL priority is ON or > 1.0
21. WRITE Present_Value = V1 PRIORITY = PTY1
22. WRITE Present_Value = V1, PRIORITY = PTY3, a value numerically greater than PTY1
23. WRITE PROP_REF = C1, PRIORITY = PTY1
24. WAIT Internal Processing Fail Time
    CHECK (blink-warn did not occur)
25. VERIFY Egress_Active = FALSE
26. VERIFY Priority_Array = NULL, ARRAY INDEX = PTY1
27. WRITE Present_Value = NULL, PRIORITY = PTY3

-- Test for the value of Relinquish_Default is ON or > 1.0
28. IF (Relinquish_Default is writable) THEN
    WRITE Present_Value = V1, PRIORITY = PTY1
    WRITE Relinquish_Default = V1
    WRITE PROP_REF = C1, PRIORITY = PTY1
    WAIT Internal Processing Fail Time
    CHECK (blink-warn did not occur)
    VERIFY Egress_Active = FALSE
    VERIFY Priority_Array = NULL, ARRAY INDEX = PTY1
    WRITE Relinquish_Default = V2

-- Test for Blink_Warn_Enable is FALSE
29. IF (Blink_Warn_Enable is writable) THEN
    WRITE Present_Value = V1, PRIORITY = PTY1
    WRITE Blink_Warn_Enable = FALSE
    WRITE PROP_REF = C1, PRIORITY = PTY1
    WAIT Internal Processing Fail Time
    CHECK (blink-warn did not occur)
    VERIFY Egress_Active = FALSE
    VERIFY Priority_Array = NULL, ARRAY INDEX = PTY1

```

### 7.3.1.X41.Y8 Blink-Warn WARN\_OFF Command Halted Test

Purpose: To verify blink-warn WARN\_OFF execution is halted when a higher priority entry is written or the Present\_Value at the specified priority is changed.

Test Concept: Select an object O1 that supports blink-warn commands. Configure O1 such that a blink-warn command will generate a blink-warn. Before the Egress timer expires, verify the specified actions clear the blink-warn properties and the Priority\_Array is correctly changed.

Configuration Requirements: O1 shall be configured such that all slots in the Priority\_Array numerically less than PTY1 have a value of NULL and no internal algorithms are issuing commands to O1 at a priority numerically less than or equal to PTY1. Blink\_Warn\_Enable is TRUE, Egress\_Time is a non-zero value and Egress\_Active is FALSE.

	Binary Lighting Output object	Lighting Output object
<b>PROP_REF</b>	<b>Present_Value</b>	<b>Present_Value or Lighting_Command</b>
<b>C1</b>	<b>WARN_OFF</b>	<b>-3.0 if PROP_REF = Present_Value, otherwise WARN_OFF</b>
<b>V1 to V3</b>	<b>ON</b>	<b>&gt;1.0</b>
<b>V4</b>	<b>OFF</b>	<b>0.0</b>

#### Test Steps:

-- Test for a higher priority entry is written to a non NULL value

1. WRITE Present\_Value = V1, PRIORITY = PTY1
2. VERIFY Blink\_Warn\_Enable = TRUE
3. VERIFY Egress\_Time > 0
4. VERIFY Egress\_Active = FALSE
5. WRITE PROP\_REF = C1, PRIORITY = PTY1
6. BEFORE **Internal Processing Fail Time**  
CHECK (blink-warn occurred)
7. BEFORE Egress\_Active = FALSE  
WRITE Present\_Value = V2, PRIORITY = PTY2, a value not equal to 6 and less than PTY1
8. VERIFY Egress\_Active = FALSE
9. VERIFY Priority\_Array = V4, ARRAY INDEX = PTY1
10. WRITE Present\_Value = NULL, PRIORITY = PTY2

-- Test for the Present\_Value at the specified property is changed

11. WRITE Present\_Value = V1, PRIORITY = PTY1
12. VERIFY Blink\_Warn\_Enable = TRUE
13. VERIFY Egress\_Time > 0
14. VERIFY Egress\_Active = FALSE
15. WRITE PROP\_REF = C1, PRIORITY = PTY1
16. BEFORE **Internal Processing Fail Time**  
CHECK (blink-warn occurred)
17. BEFORE Egress\_Active = FALSE  
WRITE Present\_Value = V3, PRIORITY = PTY1
18. VERIFY Egress\_Active = FALSE
19. VERIFY Priority\_Array = V3, ARRAY INDEX = PTY1

### 7.3.1.X41.Y9 Blink-Warn WARN\_RELINQUISH Command Halted Test

Purpose: To verify blink-warn WARN\_RELINQUISH execution is halted when a higher priority entry is written or the Present\_Value at the specified priority is changed.

Test Concept: Select an object O1 that supports blink-warn commands. Configure O1 such that a blink-warn command will generate a blink-warn. Before the Egress timer expires, verify the specified actions clear the blink-warn properties and the Priority\_Array is correctly changed.

Configuration Requirements: O1 shall be configured such that all slots in the Priority\_Array numerically less than PTY1 have a value of NULL and at least one slot numerically greater than PTY1 or Relinquish\_Default shall have a value of V1 and all other slots numerically greater than PTY1 shall have a value of V0. No internal algorithms are issuing commands to O1 at any priority. Blink\_Warn\_Enable is TRUE, Egress\_Time is a non-zero value, Egress\_Active is FALSE.

	Binary Lighting Output object	Lighting Output object
PROP_REF	Present_Value	Present_Value or Lighting_Command
C1	WARN_RELINQUISH	-2.0 if PROP_REF = Present_Value, otherwise WARN_RELINQUISH
V0	OFF or NULL	0.0 or NULL
V1	OFF	0.0
V2	ON	>1.0
V3	OFF	any value >1.0 and not equal to V2

#### Test Steps:

-- Test for a higher priority entry is written to a non NULL value

1. WRITE Present\_Value = V2, PRIORITY = PTY1
2. VERIFY Blink\_Warn\_Enable = TRUE
3. VERIFY Egress\_Time > 0
4. VERIFY Egress\_Active = FALSE
5. WRITE PROP\_REF = C1, PRIORITY = PTY1



6. BEFORE **Internal Processing Fail Time**  
CHECK (blink-warn occurred)
  7. BEFORE Egress\_Active = FALSE  
WRITE Present\_Value = V2, PRIORITY = PTY2, a value not equal to 6 and less than PTY1
  8. VERIFY Egress\_Active = FALSE
  9. VERIFY Priority\_Array = NULL, ARRAY INDEX = PTY1
  10. WRITE Present\_Value = NULL, PRIORITY = PTY2
- Test for the Present\_Value at the specified property is changed
11. WRITE Present\_Value = V2, PRIORITY = PTY1
  12. VERIFY Blink\_Warn\_Enable = TRUE
  13. VERIFY Egress\_Time > 0
  14. VERIFY Egress\_Active = FALSE
  15. WRITE PROP\_REF = C1, PRIORITY = PTY1
  16. BEFORE **Internal Processing Fail Time**  
CHECK (blink-warn occurred)
  17. BEFORE Egress\_Active = FALSE  
WRITE Present\_Value = V3, PRIORITY = PTY1
  18. VERIFY Egress\_Active = FALSE
  19. VERIFY Priority\_Array = V3, ARRAY INDEX = PTY1

[Add new **Clause 7.3.1.X42** p. 55]

### **7.3.1.X42 Value Source Mechanism Tests**

#### **7.3.1.X42.Y1 Writing to the Value\_Source Property by a Device Other than the Device that Commanded the Object.**

Purpose: To verify the IUT correctly refuses an attempt to write a Value\_Source property by a device other than the device that most recently commanded the object.

Test Concept: Command an object, O1, that supports the value source mechanism, from device D1, and verify the updated Value\_Source. Attempt to write to the Value\_Source property from device D2. Verify that an error is returned and Value\_Source does not change.

Test Steps:

1. TRANSMIT WriteProperty-Request,  
SOURCE = D1,  
'Object Identifier' = O1,  
'Property Identifier' = (P1: the property monitored by the Value\_Source mechanism for this object type),  
'Priority' = (PRIO: absent or any value other than 6)  
'Property Value' = (X2: any valid value)
2. RECEIVE BACnet-SimpleACK-PDU
3. IF (O1 is commandable) THEN  
VERIFY Priority\_Array = X2, ARRAY\_INDEX = PRIO  
ELSE  
VERIFY (P1) = X2
4. VERIFY Value\_Source = (D1's device identifier or network address)
5. TRANSMIT WriteProperty-Request,  
SOURCE = D2,  
'Object Identifier' = O1,  
'Property Identifier' = Value\_Source,  
'Priority' = PRIO,  
'Property Value' = (any valid value)
6. RECEIVE BACnet-Error PDU,  
'Error Class' = PROPERTY,  
'Error Code' = WRITE\_ACCESS\_DENIED
7. VERIFY Value\_Source = (D1's device identifier or network address)

### 7.3.1.X42.Y2 Non-commandable Value\_Source Property Test

Purpose: To verify that the Value\_Source property indicates the source of the current Present\_Value in a non-commandable object.

Test Concept: Select a non-commandable object with a writable Present\_Value which supports the Value Source mechanism. Present\_Value is written, and it is verified that Value\_Source is updated appropriately. Value\_Source is then written to verify that the last writer can update it.

Test Steps:

1. WRITE Present\_Value = V1
2. VERIFY Present\_Value = V1
3. VERIFY Value\_Source = (TD's device identifier or network address)
4. WRITE Value\_Source = (any valid value, V2)
5. VERIFY Value\_Source = V2

### 7.3.1.X42.Y3 Value\_Source Property None Test

Purpose: To verify that the Value\_Source property shall have the value 'None' when there is no active value source.

Test Concept: If there is no active value source, i.e., the Present\_Value has taken on the value of Relinquish\_Default, then the Value\_Source property shall have the value 'None'.

Configuration Requirements: The object (O1) to be tested shall have 1 non-NULL entry in its Priority\_Array and the Current\_Command\_Priority has a value other than NULL or 6.

Test Steps:

1. READ PRIO = Current\_Command\_Priority
2. CHECK(PRIO <> 6 and PRIO <> NULL)
3. VERIFY Value\_Source = (is not 'None')
4. WRITE Present\_Value = NULL, PRIORITY = PRIO
5. VERIFY Last\_Command\_Time ~= (the current local time)
6. IF (O1 has Minimum\_On\_Time or Minimum\_Off\_Time properties) THEN  
    WAIT the larger of Minimum\_Off\_Time and Minimum\_On\_Time
7. VERIFY Current\_Command\_Priority = NULL
8. VERIFY Value\_Source = 'None'           -- the value is the choice 'none'

### 7.3.1.X42.Y4 Commandable Value Source Test

Purpose: To verify that the Value\_Source, Value\_Source\_Array, and Last\_Command\_Time update correctly when Present\_Value is written in a commandable object.

Test Concept: A commandable object which supports the value source mechanism is selected for the test. The Present\_Value is written. Last\_Command\_Time, Value\_Source, and Value\_Source\_Array properties are checked to verify that they have been updated appropriately. Value\_Source is then written, and it is verified that Last\_Command\_Time property has not changed.

Configuration Requirements: The object being tested shall be commandable and support the Value Source mechanism. No other internal processes shall be controlling the object.

Test Steps:

-- Verify that the value source properties are updated when Present\_Value is commanded.

1. WRITE Present\_Value = (V1: any valid value), PRIORITY = (PRIO: any value other than 6)
2. VERIFY Value\_Source\_Array = (SRC1: TD's device identifier or network address),  
    ARRAY\_INDEX = PRIO
3. VERIFY Value\_Source = SRC1

4. VERIFY Last\_Command\_Time ~= (the current local time)

-- Verify that Value\_Source can be written and that Last\_Command\_Time does not update.

5. READ T1 = (O1), Last\_Command\_Time
6. WRITE Value\_Source = (SRC2: any valid value), PRIORITY = PRIO
7. VERIFY Value\_Source\_Array = SRC2, ARRAY\_INDEX = PRIO
8. IF (Current\_Command\_Priority == PRIO) THEN  
    VERIFY Value\_Source = SRC2
9. VERIFY Last\_Command\_Time = T1

### 7.3.1.X42.Y5 Life Safety Value\_Source Property Test

Purpose: To verify that the Value\_Source property indicates the source of the current Mode property in a life safety object.

Test Concept: Select a life safety object which supports the Value Source mechanism. Mode is written, and it is verified that Value\_Source is updated appropriately. Value\_Source is then written to verify that the last writer can update it.

Test Steps:

1. WRITE Mode = V1
2. VERIFY Mode = V1
3. VERIFY Value\_Source = (TD's device identifier or network address)
4. WRITE Value\_Source = (any valid value, V2)
5. VERIFY Value\_Source = V2

[Add new **Clause 7.3.1.X498**, p. 55]

### 7.3.1.X498 Audit\_Level Property Tests

#### 7.3.1.X498.1 Object Specific Configurable Audit\_Level NONE Test

Purpose: Verify that the Audit\_Level property, in an auditable object, controls the audit level for the object.

Test Concept: An object, O1, is selected which supports a configurable Audit\_Level property. With O1 configured to report notifications, Audit\_Level is changed to NONE. An auditable action is performed on O1 and it is verified that no notification is generated.

Audit\_Level is then changed to AUDIT\_CONFIG, and an auditable config action is performed on the object. It is verified that a notification is sent. An auditable non-config action is performed on the object, and it is verified that no notification is sent.

Audit\_Level is then changed to AUDIT\_ALL, and an auditable action is performed on the object. It is verified that a notification is sent.

Configuration Requirements: The IUT is configured to generate audit notifications. The selected object, O1, shall have auditable configuration operations that can be applied to it. AR is the Audit Reporter object configured to report for O1.

Notes to Tester: In this test, the audit reporting configuration properties are assumed to be writable. If one is only configurable, then the WRITE operation should be replaced with MAKE.

Test Steps:

1. WRITE O1, Audit\_Level = NONE
2. WRITE AR, Audit\_Level = (AUDIT\_CONFIG or AUDIT\_ALL)
3. MAKE(perform an operation on O1 that would be reported by the AR if O1.Audit\_Level were AUDIT\_ALL)

```

4. WAIT(AR.Maximum_Send_Delay + Notification Fail Time)
5. CHECK(that the IUT did not report an audit notification for the operation)

6. IF (O1 has auditable configuration operations, such as writable configuration properties) THEN {
    WRITE O1, Audit_Level = AUDIT_CONFIG
    WRITE AR, Audit_Level = NONE
    MAKE(perform a config operation on O1)
    IF the IUT is configured to generate unconfirmed audit notifications THEN {
        BEFORE AR.Maximum_Send_Delay + Notification Fail Time
        RECEIVE UnconfirmedAuditNotification-Request,
        'Notifications' = (a notification of the operation performed)
    } ELSE {
        BEFORE AR.Maximum_Send_Delay + Notification Fail Time
        RECEIVE ConfirmedAuditNotification-Request,
        'Notifications' = (a notification of the operation performed)
        TRANSMIT BACnet-SimpleACK-PDU
    }
    MAKE(perform an auditable non-config operation on O1)
    WAIT AR.Maximum_Send_Delay + Notification Fail Time
    CHECK(that the IUT did not report an audit notification for the operation)
}

7. WRITE O1, Audit_Level = AUDIT_ALL
8. WRITE AR, Audit_Level = NONE
9. MAKE(perform an auditable operation on O1)
10. IF the IUT is configured to generate unconfirmed audit notifications THEN {
    BEFORE AR.Maximum_Send_Delay + Notification Fail Time
    RECEIVE UnconfirmedAuditNotification-Request,
    'Notifications' = (a notification of the operation performed)
} ELSE {
    BEFORE AR.Maximum_Send_Delay + Notification Fail Time
    RECEIVE UnconfirmedAuditNotification-Request,
    'Notifications' = (a notification of the operation performed)
    TRANSMIT BACnet-SimpleACK-PDU
}

```

### 7.3.1.X498.2 Audit Reporter Audit\_Level Test

Purpose: Verify that the Audit\_Reporter's Audit\_Level property is used in objects without an Audit\_Level, or when an object's Audit\_Level is DEFAULT.

Test Concept: An object, O1, is selected which supports a configurable Audit\_Level property, or which does not have an Audit\_Level property. The Audit\_Reporter for O1 is referred to as AR1. If O1 has an Audit\_Level property, it is set to DEFAULT.

With O1 configured to report notifications, AR1's Audit\_Level is changed to NONE. An auditable action is performed on O1 and it is verified that no notification is generated.

AR1's Audit\_Level is then changed to AUDIT\_CONFIG, and an auditable config action is performed on O1. It is verified that a notification is sent. An auditable non-config action is performed on O1, and it is verified that no notification is sent.

Audit\_Level is then changed to AUDIT\_ALL, and an auditable action is performed on O1. It is verified that a notification is sent.

Configuration Requirements: The IUT is configured to generate audit notifications. The selected object, O1, should have auditable configuration operations and auditable non-configuration operations that can be applied to it. AR is the Audit Reporter object configured to report for O1.

Notes to Tester: In this test, the audit reporting configuration properties are assumed to be writable. If one is only configurable, then the WRITE operation should be replaced with MAKE.

#### Test Steps:

1. IF O1 contains an Audit\_Level property THEN  
    WRITE O1, Audit\_Level = DEFAULT
2. WRITE AR.Audit\_Level = NONE
3. MAKE(perform an operation on O1 that would be reported by the AR if O1.Audit\_Level were AUDIT\_ALL)
4. WAIT AR.Maximum\_Send\_Delay + Notification Fail Time
5. CHECK(that the IUT did not report an audit notification for the operation)
  
6. WRITE AR.Audit\_Level = AUDIT\_CONFIG
7. IF O1 supports auditable config operations THEN {  
    MAKE(perform a config operation on O1)  
    IF the IUT is configured to generate unconfirmed audit notifications THEN {  
        BEFORE AR.Maximum\_Send\_Delay + Notification Fail Time  
        RECEIVE UnconfirmedAuditNotification-Request,  
        'Notifications' = (a notification of the operation performed)  
    } ELSE {  
        BEFORE AR.Maximum\_Send\_Delay + Notification Fail Time  
        RECEIVE ConfirmedAuditNotification-Request,  
        'Notifications' = (a notification of the operation performed)  
        TRANSMIT BACnet-SimpleACK-PDU  
    }  
}
8. IF O1 supports auditable config operations THEN {  
    MAKE(perform a non-config operation on O1)  
    WAIT AR.Maximum\_Send\_Delay + Notification Fail Time  
    CHECK(that the IUT did not report an audit notification for the operation)  
}
  
9. WRITE AR.Audit\_Level = AUDIT\_ALL
10. IF O1 supports auditable config operations THEN {  
    MAKE(perform a config operation on O1)  
    IF the IUT is configured to generate unconfirmed audit notifications THEN {  
        BEFORE AR.Maximum\_Send\_Delay + Notification Fail Time  
        RECEIVE UnconfirmedAuditNotification-Request,  
        'Notifications' = (a notification of the operation performed)  
    } ELSE {  
        BEFORE AR.Maximum\_Send\_Delay + Notification Fail Time  
        RECEIVE ConfirmedAuditNotification-Request,  
        'Notifications' = (a notification of the operation performed)  
        TRANSMIT BACnet-SimpleACK-PDU  
    }  
}
11. IF O1 supports auditable config operations THEN {  
    MAKE(perform a non-config operation on O1)  
    IF the IUT is configured to generate unconfirmed audit notifications THEN {  
        BEFORE AR.Maximum\_Send\_Delay + Notification Fail Time  
        RECEIVE UnconfirmedAuditNotification-Request,  
        'Notifications' = (a notification of the operation performed)  
    } ELSE {  
        BEFORE AR.Maximum\_Send\_Delay + Notification Fail Time  
        RECEIVE ConfirmedAuditNotification-Request,  
        'Notifications' = (a notification of the operation performed)  
        TRANSMIT BACnet-SimpleACK-PDU  
    }  
}

### 7.3.1.X498.3 Audit\_Level Change Notification Test

Purpose: Verify that changes to Audit\_Level property results in audit notifications.

Test Concept: An object, O1, is selected which supports a writable and/or configurable Audit\_Level property. The Audit\_Level property is written to each valid audit level, and a notification of the change is checked for. The Audit\_Level property is then configured (not via BACnet writes) to each valid audit level, and a notification of the change is checked for.

Configuration Requirements: The IUT is configured to generate audit notifications, and O1's Audit\_Level property is set to NONE.

Test Steps:

```

1. IF Audit_Level is writable THEN {
  REPEAT AL = ( AUDIT_CONFIG, AUDIT_ALL, DEFAULT, NONE ) DO {
    IF O1 is an Audit Reporter and AL is DEFAULT THEN {
      -- don't test DEFAULT on Audit Report object
    } ELSE {
      WRITE O1, Audit = AL
      IF the IUT is configured to send unconfirmed audit notifications THEN {
        BEFORE AR.Maximum_Send_Delay + Notification Fail Time
        RECEIVE UnconfirmedAuditNotification-Request,
          'Notifications' = (a notification indicating change of
            Audit_Level)
      } ELSE {
        BEFORE AR.Maximum_Send_Delay + Notification Fail Time
        RECEIVE ConfirmedAuditNotification-Request,
          'Notifications' = (a notification indicating change of
            Audit_Level)
        TRANSMIT BACnet-SimpleACK-PDU
      }
    }
  }
}

2. IF Audit_Level is changeable without writing it via BACnet THEN {
  REPEAT AL = ( AUDIT_CONFIG, AUDIT_ALL, DEFAULT, NONE ) DO {
    IF O1 is an Audit Reporter and AL is DEFAULT THEN {
      -- don't test DEFAULT on Audit Report object
    } ELSE {
      MAKE(O1, Audit = AL without using BACnet services to write the property)
      IF the IUT is configured to send unconfirmed audit notifications THEN {
        BEFORE AR.Maximum_Send_Delay + Notification Fail Time
        RECEIVE UnconfirmedAuditNotification-Request,
          'Notifications' = (a notification indicating change of
            Audit_Level)
      } ELSE {
        BEFORE AR.Maximum_Send_Delay + Notification Fail Time
        RECEIVE ConfirmedAuditNotification-Request,
          'Notifications' = (a notification indicating change of
            Audit_Level)
        TRANSMIT BACnet-SimpleACK-PDU
      }
    }
  }
}

```

[Add new **Clause 7.3.1.X499**, p. 55]

### 7.3.1.X499 Audit\_Notification\_Recipient Property Tests

#### 7.3.1.X499.1 Audit\_Notification\_Recipient Test

Purpose: Verify that an Audit\_Notification\_Recipient accepts and correctly uses all forms of recipient addresses.

Test Concept: With an object configured to report notifications, the Device object's Audit\_Notification\_Recipient property is set to a local broadcast. An action is performed on the IUT which should result in an audit notification, and it is verified that the notification is locally broadcast. This is repeated for global broadcasts, and unicast recipients. Only the unicast form is tested for devices which do not generate UnconfirmedAuditNotifications.

Configuration Requirements: The IUT is configured to report audit notifications. If the IUT supports sending unconfirmed audit notifications, it shall be configured to do so. Audit\_Notification\_Recipient is configured to be the TD, and audit reporting is enabled in the IUT.

Notes to Tester: Where the IUT is expected to send multiple audit notifications for an operation, the notifications can be generated in any order.

Test Steps:

-- Local Broadcast Recipient

```
1. IF the IUT supports sending unconfirmed audit notifications THEN {
    WRITE Audit_Notification_Recipient = (local broadcast recipient)
    BEFORE Maximum_Send_Delay + Notification Fail Time
        RECEIVE UnconfirmedAuditNotification-Request,
            'Notifications' =          (a notification for the change to Audit_Notification_Recipient)
        |
        (UnconfirmedAuditNotification-Request
            DESTINATION =   GLOBAL BROADCAST,
            'Notifications' =          (a notification for the change to
                Audit_Notification_Recipient)
        )
    IF the first notification was not globally broadcast) THEN {
        RECEIVE UnconfirmedAuditNotification-Request,
            DESTINATION =   LOCAL BROADCAST,
            'Notifications' =          (a notification for the change to Audit_Notification_Recipient)
    }
    MAKE(perform an operation on the IUT which will result in an audit notification, other than changing the
        Audit_Notification_Recipient property)
    BEFORE Maximum_Send_Delay + Notification Fail Time
        RECEIVE UnconfirmedAuditNotification-Request,
            DESTINATION =   LOCAL BROADCAST,
            'Notifications' =          (a notification correctly indicating the operation performed)
    }
```

-- Global Broadcast Recipient

```

2. IF the IUT supports sending unconfirmed audit notifications THEN {
    WRITE Audit_Notification_Recipient = (global broadcast recipient)
    BEFORE Maximum_Send_Delay + Notification Fail Time
        RECEIVE UnconfirmedAuditNotification-Request,
            DESTINATION = LOCAL BROADCAST,
            'Notifications' = (a notification for the change to Audit_Notification_Recipient)
    |
    (UnconfirmedAuditNotification-Request
        DESTINATION = GLOBAL BROADCAST,
        'Notifications' = (a notification for the change to
            Audit_Notification_Recipient)
    )
    IF (the first notification was not globally broadcast) THEN {
        RECEIVE UnconfirmedAuditNotification-Request,
            DESTINATION = GLOBAL BROADCAST,
            'Notifications' = (a notification for the change to Audit_Notification_Recipient)
    }
    MAKE(perform an operation on the IUT which will result in an audit notification, other than changing the
        Audit_Notification_Recipient property)
    BEFORE Maximum_Send_Delay + Notification Fail Time
        RECEIVE UnconfirmedAuditNotification-Request,
            DESTINATION = GLOBAL BROADCAST,
            'Notifications' = (a notification correctly indicating the operation performed)
}

```

-- Unicast Recipient

```

3. WRITE Audit_Notification_Recipient = (D1: a device other than the TD)
4. IF the IUT is configured to send unconfirmed notifications THEN {
    BEFORE (Maximum_Send_Delay plus maximum time to resolve TD)
        RECEIVE UnconfirmedAuditNotification-Request,
            DESTINATION = TD,
            'Notifications' = (a notification for the change to Audit_Notification_Recipient)
    | (UnconfirmedAuditNotification-Request
        DESTINATION = GLOBAL BROADCAST,
        'Notifications' = (a notification for the change to Audit_Notification_Recipient)
    )
} ELSE {
    BEFORE (Maximum_Send_Delay plus maximum time to resolve TD)
        RECEIVE ConfirmedAuditNotification-Request,

```



```

        DESTINATION = D1,
        'Notifications' = (a notification for the change to Audit_Notification_Recipient)
    TRANSMIT BACnet-SimpleACK-PDU,
        SOURCE = D1
}

```

5. IF (the first notification was not globally broadcast) THEN

```

    RECEIVE UnconfirmedAuditNotification-Request,
        DESTINATION = TD,
        'Notifications' = (a notification for the change to Audit_Notification_Recipient)

```

6. MAKE(perform an operation on the IUT which will result in an audit notification, other than changing the Audit\_Notification\_Recipient property)

7. BEFORE Maximum\_Send\_Delay

```

    RECEIVE UnconfirmedAuditNotification-Request,
        DESTINATION = TD,
        'Notifications' = (a notification correctly indicating the operation performed)

```

[Add new **Clause 7.3.1.X500**, p. 55]

### 7.3.1.X500 Audit\_Priority\_Filter Property Tests

#### 7.3.1.X500.1 Audit\_Priority\_Filter Target Audit Reporting Test

Purpose: Verify that Audit\_Priority\_Filter correctly filters the generation of audit notifications based on priority.

Test Concept: An auditable commandable object for which Audit\_Priority\_Filter filtering can be applied is configured to report all write operations. The Audit\_Priority\_Filter is configured to restrict audit notifications to all but a single priority X. The object is commanded at a priority other than X and it is verified that an audit notification is sent. The object is then commanded at priority X and it is verified that no audit notification is sent.

Configuration Requirements: The IUT is configured to report audit notifications for writes on a commandable object, O1. If the IUT does not support the Priority\_Array property in any object for which audit reporting can be configured, or if the IUT does not support a configurable Audit\_Priority\_Filter property, this test shall be skipped. AR shall be the Audit Reporter object for O1.

Notes to Tester: In this test, the audit reporting configuration properties are assumed to be writable. If one is only configurable, then the WRITE operation should be replaced with MAKE.

Test Steps:

-- Test the object's Audit\_Priority\_Filter

```

1. IF O1 supports a configurable Audit_Priority_Filter property THEN {
    WRITE O1, Audit_Priority_Filter = { all ones except priority X (bit X-1) }
    TRANSMIT WriteProperty-Request,
        'Invoke Id' = I,
        'Object Identifier' = O1,
        'Property Identifier' = Present_Value,
        'Property Value' = (V: any valid value),
        'Priority' = (PRIO: any valid priority, but not X)
    BEFORE Internal Processing Fail Time
    RECEIVE BACnet-SimpleACK-PDU
}

```

```

|
(BACnet-Error-PDU,
  'Error Type' = (E: any error)
)
IF the IUT is configured to send unconfirmed audit notifications THEN {
  BEFORE AR.Maximum_Send_Delay + Notification Fail Time
  RECEIVE UnconfirmedAuditNotification-Request,
    'Notifications' = ({
      -- there may be a second notification included for the write to
      -- Audit_Priority_Filter. If there is, the order of the
      -- notifications in the list is not relevant

      -- source-timestamp absent
      target-timestamp = (IUT's local time),
      source-device = TD,
      -- source-object absent
      operation = WRITE,
      -- source-comment absent
      target-comment = (any valid value, or absent),
      invoke-id = (the invoke ID of the write to Present_Value),
      source-user-id = (the value from the operation if provided, otherwise absent),
      source-user-role = (the value from the operation if provided, otherwise absent),
      target-device = IUT,
      target-object = O1,
      target-property = Present_Value,
      target-priority = PRIO,
      target-value = V,
      current-value = (the value before the write. may be absent if the value size is
        larger than 32 encoded octets),
      result = (E, if the op failed, otherwise absent)
    })
} ELSE {
  BEFORE AR.Maximum_Send_Delay + Notification Fail Time
  RECEIVE ConfirmedAuditNotification-Request,
    'Notifications' = ({
      -- there may be a second notification included for the write to
      -- Audit_Priority_Filter. If there is, the order of the
      -- notifications in the list is not relevant

      -- source-timestamp absent
      target-timestamp = (IUT's local time),
      source-device = TD,
      -- source-object absent
      operation = WRITE,
      -- source-comment absent
      target-comment = (any valid value, or absent),
      invoke-id = (the invoke ID of the write to Present_Value),
      source-user-id = (the value from the operation if provided, otherwise absent),
      source-user-role = (the value from the operation if provided, otherwise absent),
      target-device = IUT,
      target-object = O1,
      target-property = Present_Value,
      target-priority = PRIO,
      target-value = V,
      current-value = (the value before the write. may be absent if the value size is
        larger than 32 encoded octets),
      result = (E, if the op failed, otherwise absent)
    })
  TRANSMIT BACnet-SimpleACK-PDU
}
TRANSMIT WriteProperty-Request,
  'Invoke Id' = I,
  'Object Identifier' = O1,
  'Property Identifier' = Present_Value,

```

```

    'Property Value' = (V: any valid value),
    'Priority' = (PRIO: X)
BEFORE Internal Processing Fail Time
    RECEIVE BACnet-SimpleACK-PDU
    |
    (BACnet-Error-PDU,
    'Error Type' = (E: any error)
    )
    WAIT AR.Maximum_Send_Delay + Notification Fail Time
    CHECK(no audit notification sent)
}

-- Test the Audit Reporter object's Audit_Priority_Filter
2. IF AR supports a configurable Audit_Priority_Filter property and O1 has no Audit_Priority_Filter or it can be configured to None THEN {
    IF O1 has an Audit_Priority_Filter THEN
        WRITE O1, Audit_Priority_Filter = NONE
        WRITE AR, Audit_Priority_Filter = { all ones except priority X (bit X-1) }
        TRANSMIT WriteProperty-Request,
        'Invoke Id' = I,
        'Object Identifier' = O1,
        'Property Identifier' = Present_Value,
        'Property Value' = (V: any valid value),
        'Priority' = (PRIO: any valid priority, but not X)
    BEFORE Internal Processing Fail Time
        RECEIVE BACnet-SimpleACK-PDU
        |
        (BACnet-Error-PDU,
        'Error Type' = (E: any error)
        )
    IF the IUT is configured to send unconfirmed audit notifications THEN {
        BEFORE AR.Maximum_Send_Delay + Notification Fail Time
            RECEIVE UnconfirmedAuditNotification-Request,
            'Notifications' = ( {
                -- write to Audit_Priority_Filter. If there is, the order
                -- of the notifications in the list is not relevant
                -- source-timestamp absent
                target-timestamp = (IUT's local time),
                source-device = TD,
                -- source-object absent
                operation = WRITE,
                -- source-comment absent
                target-comment = (any valid value, or absent),
                invoke-id = (the invoke ID of the write to Present_Value),
                source-user-id = (the value from the operation if provided, otherwise
                                absent),
                source-user-role = (the value from the operation if provided, otherwise
                                absent),
                target-device = IUT,
                target-object = O1,
                target-property = Present_Value,
                target-priority = PRIO,
                target-value = V,
                current-value = (the value before the write. may be absent if the value
                                size is larger than 32 encoded octets),
                result = (E, if the op failed, otherwise absent)
            } )
    } ELSE {
        BEFORE AR.Maximum_Send_Delay + Notification Fail Time
            RECEIVE ConfirmedAuditNotification-Request,

```

```

'Notifications' = ( {
    -- there may be a second notification included for the
    -- write to Audit_Priority_Filter. If there is, the order
    -- of the notifications in the list is not relevant
    -- source-timestamp absent
    target-timestamp = (IUT's local time),
    source-device = TD,
    -- source-object absent
    operation = WRITE,
    -- source-comment absent
    target-comment = (any valid value, or absent),
    invoke-id = (the invoke ID of the write to Present_Value),
    source-user-id = (the value from the operation if provided, otherwise
        absent),
    source-user-role = (the value from the operation if provided, otherwise
        absent),
    target-device = IUT,
    target-object = O1,
    target-property = Present_Value,
    target-priority = PRIO,
    target-value = V,
    current-value = (the value before the write. may be absent if the value
        size is larger than 32 encoded octets),
    result = (E, if the op failed, otherwise absent)
} )
TRANSMIT BACnet-SimpleACK-PDU
}
TRANSMIT WriteProperty-Request,
    'Invoke Id' = I,
    'Object Identifier' = O1,
    'Property Identifier' = Present_Value,
    'Property Value' = (V: any valid value),
    'Priority' = (PRIO: X)
BEFORE Internal Processing Fail Time
RECEIVE BACnet-SimpleACK-PDU
    |
    (BACnet-Error-PDU,
    'Error Type' = (E: any error)
    )
WAIT AR.Maximum_Send_Delay + Notification Fail Time
CHECK(no audit notification sent)
}

```

[Add new **Clause 7.3.1.X501**, p. 55]

### 7.3.1.X501 Auditable\_Operations Property Tests

#### 7.3.1.X501.1 Non-configurable Auditable\_Operations Property Test

Purpose: Verify that non-configurable Auditable\_Operations properties are set to the required value.

Test Concept: Select an object, O1, which has a non-configurable Auditable\_Operations property. Verify that the property is set to the required value.

Test Steps:

1. READ AO = Audit\_Operations
2. CHECK(Audit\_Operations = (FALSE,TRUE,TRUE,TRUE,?,TRUE,?,?,?,FALSE,FALSE,?,?,...))

```
-- flags READ, NOTIFICATION and SUBSCRIPTION are FALSE,
-- flags WRITE, CREATE, DELETE, ACKNOWLEDGE-ALARM are TRUE, and
-- all other flags can be any value
```

### 7.3.1.X501.2 Auditable\_Operations Target Audit Reporting Test

Purpose: Verify that Auditable\_Operations controls which operations are auditable.

Test Concept: The IUT is configured to report some operations and not others through the Auditable\_Operations property. Each of the standard auditable operations are performed against the IUT, and the filtering provided by Auditable\_Operations is verified to be correct.

Configuration Requirements: The IUT is configured to report all audit notifications with the exception that at least some auditable operations are disabled via Auditable\_Operations.

Test Steps:

1. REPEAT OP = (each of the standard auditable operations for which the IUT is configured to report except any operation which is unreasonably difficult to perform, such as

```
AUDIITING_FAILURE) DO {
  MAKE(perform OP on the IUT)
  IF the IUT is configured to send unconfirmed audit notifications THEN {
    BEFORE (Audit Reporter for OP).Maximum_Send_Delay + Notification Fail Time
    RECEIVE UnconfirmedAuditNotification-Request,
    'Notifications' = ({
      -- source-timestamp absent
      target-timestamp = (IUT's local time),
      source-device = TD,
      -- source-object absent
      operation = OP,
      -- source-comment absent
      target-comment = (any valid value, or absent unless OP is GENERAL),
      invoke-id = (the invoke id from the operation, or absent if it was
        unconfirmed),
      source-user-id = (the value from the operation if provided, otherwise
        absent),
      source-user-role = (the value from the operation if provided, otherwise
        absent),
      target-device = IUT,
      target-object = (the target object or absent if the target is not an object),
      target-property = (the target property or absent if the target is not a
        property),
      target-priority = (the priority supplied, or absent if the target is not a
        property. shall be 16 or absent if no priority supplied and the
        target is a property),
      target-value = (the target value or absent if no target value for the
        operation. may be absent if the value size is larger than 32
        encoded octets),
      current-value = (the value before the op or absent if no target value.
        may be absent if the value size is larger than 32 encoded
        octets),
      result = (the reason for failure if OP failed, otherwise absent)
    })
  } ELSE {
    BEFORE (Audit Reporter for OP).Maximum_Send_Delay + Notification Fail Time
    RECEIVE ConfirmedAuditNotification-Request,
    'Notifications' = ({
      -- source-timestamp absent
      target-timestamp = (IUT's local time),
      source-device = TD,
```

```

-- source-object absent
operation = OP,
-- source-comment absent
target-comment = (any valid value, or absent unless OP is GENERAL),
invoke-id = (the invoke id from the operation, or absent if it was
    unconfirmed),
source-user-id = (the value from the operation if provided, otherwise
    absent),
source-user-role = (the value from the operation if provided, otherwise
    absent),
target-device = IUT,
target-object = (the target object or absent if the target is not an object),
target-property = (the target property or absent if the target is not a
    property),
target-priority = (the priority supplied, or absent if the target is not a
    property. shall be 16 or absent if no priority supplied and the
    target is a property),
target-value = (the target value or absent if no target value for the
    operation. may be absent if the value size is larger than 32
    encoded octets),
current-value = (the value before the op or absent if no target value.
    may be absent if the value size is larger than 32 encoded
    octets),
result = (the reason for failure if OP failed, otherwise absent)
    } )
TRANSMIT BACnet-SimpleACK-PDU
}
}
2. REPEAT OP = (each of the standard auditable operations for which the IUT is configured to NOT report
    except any operation which is unreasonably difficult to perform, such as
    AUDITING_FAILURE) DO {
    MAKE(perform OP on the the IUT)
    WAIT (Audit Reporter for OP).Maximum_Send_Delay + Notification Fail Time
    CHECK(no audit notification was received)
}

```

### 7.3.1.X501.3 Auditable\_Operations Source Audit Reporting Test

Purpose: Verify that Auditable\_Operations controls which operations performed by the IUT are auditable.

Test Concept: The IUT is configured to report some source operations and not others through the Auditable\_Operations property. Each of the standard auditable operations which the IUT can perform are performed by the IUT, and the filtering provided by Auditable\_Operations is verified to be correct.

Configuration Requirements: The IUT is configured to report all source audit notifications with the exception that at least some auditable operations are disabled via Auditable\_Operations.

Test Steps:

1. REPEAT OP = (each of the standard auditable operations the IUT is able to perform on another device and is configured to report except any operation which is unreasonably difficult to perform) DO {
 MAKE(the IUT perform OP on the TD)
 IF the IUT is configured to send unconfirmed audit notifications THEN {
 BEFORE (Audit Reporter for OP).Maximum\_Send\_Delay + Notification Fail Time
 RECEIVE UnconfirmedAuditNotification-Request,
 'Notifications' = ( {
 source-timestamp = (IUT's local time),
 -- target-timestamp absent
 source-device = IUT,
 source-object = (the object which initiated the op or absent if not

```

        initiated by an object),
        operation = OP,
        source-comment = (any valid value, or absent unless OP is
            GENERAL),
        -- target-comment absent
        invoke-id = (the invoke id from the operation, or absent if it was
            unconfirmed),
        source-user-id = (the value from the operation if provided, otherwise
            absent),
        source-user-role = (the value from the operation if provided, otherwise
            absent),
        target-device = TD,
        target-object = (the target object or absent if the target is not an object),
        target-property = (the target property or absent if the target is not a
            property),
        target-priority = (the priority supplied, or absent if the target is not a
            property. shall be 16 or absent if no priority supplied and the
            target is a property),
        target-value = (the target value or absent if no target value for the
            operation. may be absent if the value size is larger than 32
            encoded octets),
        current-value = (the value before the op if the op targeted a property, or
            absent. May be absent even if targeting a property),
        result = (the reason for failure if the op failed, otherwise absent)
    } )
} ELSE {
    BEFORE (Audit Reporter for OP).Maximum_Send_Delay + Notification Fail Time
    RECEIVE ConfirmedAuditNotification-Request,
    'Notifications' = ( {
        source-timestamp = (IUT's local time),
        -- target-timestamp absent
        source-device = IUT,
        source-object = (the object which initiated the op or absent if not
            initiated by an object),
        operation = OP,
        source-comment = (any valid value, or absent unless OP is
            GENERAL),
        -- target-comment absent
        invoke-id = (the invoke id from the operation, or absent if it was
            unconfirmed),
        source-user-id = (the value from the operation if provided, otherwise
            absent),
        source-user-role = (the value from the operation if provided, otherwise
            absent),
        target-device = TD,
        target-object = (the target object or absent if the target is not an object),
        target-property = (the target property or absent if the target is not a
            property),
        target-priority = (the priority supplied, or absent if the target is not a
            property. shall be 16 or absent if no priority supplied and the
            target is a property),
        target-value = (the target value or absent if no target value for the
            operation. may be absent if the value size is larger than 32
            encoded octets),
        current-value = (the value before the op if the op targeted a property, or
            absent. May be absent even if targeting a property),
        result = (the reason for failure if the op failed, otherwise absent)
    } )
    TRANSMIT BACnet-SimpleACK-PDU
}

```

}

2. REPEAT OP = (each of the standard auditable operations the IUT is able to perform on another device and is configured to NOT report except any operation which is unreasonably difficult to perform) DO {  
MAKE(the IUT perform OP on the TD)  
WAIT AR.Maximum\_Send\_Delay + Notification Fail Time  
CHECK(no audit notifications were sent)  
}

[Add new **Clause 7.3.1.X503**, p. 55]

### **7.3.1.X503 Maximum\_Send\_Delay Property Tests**

#### **7.3.1.X503.1 Maximum\_Send\_Delay Test**

Purpose: Verify that Audit Reporter objects abide the Maximum\_Send\_Delay value.

Test Concept: An Audit Reporter object is selected which contains a Maximum\_Send\_Delay property. A sequence of N auditable operations is performed on the IUT, and if the IUT is a client, the IUT is made to perform a sequence of M auditable operations. The IUT is then monitored to ensure that the audit notifications are sent within the selected Maximum\_Send\_Delay.

N, the number of operations performed on the IUT shall be 2 or greater. M, the number of operations that the IUT will perform shall be 0 if the IUT does not perform auditable operations, and 2 or greater otherwise.

The value that Maximum\_Send\_Delay is configured for shall be large enough to allow for all of the operations to be performed.

Test Steps:

1. WRITE AR1, Maximum\_Send\_Delay = (MSD: a value large enough to accomplish and report N+M auditable operations)
2. WRITE AR2, Maximum\_Send\_Delay = MSD
3. MAKE(perform N auditable operations on the IUT which would be reported through AR1)
4. MAKE(the IUT perform M auditable operations which would be reported through AR2)
5. WAIT Maximum\_Send\_Delay + (Notification Fail Time \* M+N)
6. CHECK(that all expected operations are reported)

[Add new **Clause 7.3.1.X504**, p. 55]

### **7.3.1.X504 Monitored\_Objects property Tests**

#### **7.3.1.X504.1 Monitored\_Objects Test**

Purpose: Verify that the correct Audit Reporter is used for an auditable object.

Test Concept: Each Audit Reporter, which contains a Monitored\_Objects property, in the device is tested individually. The selected Audit Reporter is enabled, and all others are disabled. An object that the enabled Audit Reporter reports for is selected. An auditable operation is performed on the object, and it is verified that an audit notification is generated. An object that the enabled Audit Reporter does not report for is selected. An auditable operation is performed on the object, and it is verified that no audit notification is generated.

Configuration Requirements: The IUT is configured to send unconfirmed audit notifications. If the Monitored\_Objects property is not supported by any Audit Reporter objects in the IUT, this test shall be skipped. If the IUT only supports a single Audit Reporter object for target object reporting, and its Monitored\_Objects property is always set to all objects, this test shall be skipped. Configure all Audit Reporters to report all operations and set Audit\_Level to NONE for all Audit Reporter objects.



Notes to Tester: In this test, the audit reporting configuration properties are assumed to be writable. If one is only configurable, then the WRITE operation should be replaced with MAKE.

Test Steps:

```

1. IF the Monitored_Objects property is writable in one or more of the AR objects THEN
    MAKE(reconfigure which AR is used by which objects)
2. REPEAT AR = (each Audit Reporter object) DO {
    O1 = (an object O1 which reports thru AR as determined by Monitored_Objects and AR precedence)
    O2 = (an object O2 which reports thru a different Audit Reporter, AR2, as determined by
        Monitored_Objects and AR precedence)
    WRITE AR.Audit_Level = AUDIT_ALL
    MAKE(perform an auditable operation on O1)
    IF the IUT is configured to send unconfirmed auto notifications THEN {
        BEFORE AR.Maximum_Send_Delay + Notification Fail Time
        RECEIVE UnconfirmedAuditNotification-Request,
        'Notifications' = (a notification indicating the operation)
    } ELSE {
        BEFORE AR.Maximum_Send_Delay + Notification Fail Time
        RECEIVE ConfirmedAuditNotification-Request,
        'Notifications' = (a notification indicating the operation)
        TRANSMIT BACnet-SimpleACK-PDU
    }
    MAKE(perform an auditable operation on O2)
    WAIT AR2.Maximum_Send_Delay + Notification Fail Time
    CHECK(that the IUT did not report an audit notification for the operation)
    WRITE AR.Audit_Level = NONE
}

```

[Add new **Clause 7.3.1.X505**, p. 55]

### 7.3.1.X505 Send\_Now Property Tests

#### 7.3.1.X505.1 Send\_Now Test

Purpose: Verify that writing True to Send\_Now results in the sending of delayed audit notifications.

Test Concept: The IUT is configured to delay sending audit notifications. An Audit Reporter object, AR, is selected which contains a Send\_Now property. An auditable operation is performed on the IUT. Send\_Now is then written and it is verified that the audit notifications are sent.

Configuration Requirements: If the IUT cannot be made to delay sending notifications without heroic efforts, this test shall be skipped.

Test Steps:

```

1. WRITE AR, Maximum_Send_Delay = (a value large enough to accomplish the test)
2. MAKE(perform whatever actions are required to make the IUT delay sending notifications)
3. WRITE AR, Send_Now = TRUE
4. IF the IUT is configured to send unconfirmed notifications THEN {
    BEFORE Notification Fail Time
    RECEIVE UnconfirmedAuditNotification-Request,
    'Notifications' = (one or more notifications for operation applied to the IUT)
} ELSE {
    BEFORE Notification Fail Time
    RECEIVE ConfirmedAuditNotification-Request,
    'Notifications' = (one or more notifications for operation applied to the IUT)
}

```

```

    TRANSMIT BACnet-SimpleACK-PDU
}

```

## 7.3.2 Object Specific Tests

### 7.3.2.4 Averaging Object Tests

[Change **Clause 7.3.2.4.1**, p. 57]

[Reviewer Note: Modified test as per clarification BTL-CR-0309.]

#### 7.3.2.4.1 Reinitializing the Samples

Purpose: To verify that an Averaging object correctly resets the Attempted\_Samples, Valid\_Samples, Minimum\_Value, Average\_Value, and Maximum\_Value when Attempted\_Samples, Object\_Property\_Reference, Window\_Interval, or Window\_Samples are changed.

Test Concept: The IUT is configured with an Averaging object that is actively monitoring some property value. The sampling is reinitialized by writing to the Attempted\_Samples, ~~Object\_Property\_Reference~~, Window\_Interval, ~~Window\_Samples~~, and ~~Window\_Samples~~ ~~Object\_Property\_Reference~~ in turn. After each reinitialization, ~~the TD pauses and verifies~~ that new sampling has begun.

Configuration Requirements: The IUT shall be configured with an Averaging object that is actively monitoring some property value. The sampling interval shall be long enough to permit the TD to verify that the sample is properly reinitialized.

Test Steps:

- ~~1. VERIFY Minimum\_Value = (a value x: -INF < x < INF),~~
- ~~2. VERIFY Average\_Value = (a value ≠ NaN),~~
- ~~3. VERIFY Maximum\_Value = (a value x: Minimum\_Value ≤ x < INF),~~
- ~~4. VERIFY Attempted\_Samples = (a value x > 0),~~
- ~~5. VERIFY Valid\_Samples = (a value x > 0),~~
- ~~6. WRITE Attempted\_Samples = 0,~~
- ~~7. VERIFY Attempted\_Samples = 0,~~
- ~~8. VERIFY Minimum\_Value = INF,~~
- ~~9. VERIFY Maximum\_Value = INF,~~
- ~~10. VERIFY Average\_Value = NaN,~~
- ~~11. VERIFY Valid\_Samples = 0,~~
- ~~12. WAIT (at least two sample times),~~
- ~~13. VERIFY Minimum\_Value = (a value x: -INF < x < INF),~~
- ~~14. VERIFY Average\_Value = (a value ≠ NaN),~~
- ~~15. VERIFY Maximum\_Value = (a value x: Minimum\_Value ≤ x < INF),~~
- ~~16. VERIFY Attempted\_Samples = (a value x ≥ 2),~~
- ~~17. VERIFY Valid\_Samples = (a value x ≥ 2),~~
- ~~18. WRITE Window\_Interval = (any new value that will result in an appropriate sample time),~~
- ~~19. VERIFY Attempted\_Samples = 0,~~
- ~~20. VERIFY Minimum\_Value = INF,~~
- ~~21. VERIFY Maximum\_Value = INF,~~
- ~~22. VERIFY Average\_Value = NaN,~~
- ~~23. VERIFY Valid\_Samples = 0,~~
- ~~24. WAIT (at least two sample times),~~
- ~~25. VERIFY Minimum\_Value = (a value x: -INF < x < INF),~~
- ~~26. VERIFY Average\_Value = (a value ≠ NaN),~~
- ~~27. VERIFY Maximum\_Value = (a value x: Minimum\_Value ≤ x < INF),~~
- ~~28. VERIFY Attempted\_Samples = (a value x ≥ 2),~~
- ~~29. VERIFY Valid\_Samples = (a value x ≥ 2),~~
- ~~30. WRITE Window\_Samples = (any new value that will result in an appropriate sample time),~~

```

31. VERIFY Attempted_Samples = 0;
32. VERIFY Minimum_Value = INF;
33. VERIFY Maximum_Value = -INF;
34. VERIFY Average_Value = NaN;
35. VERIFY Valid_Samples = 0;
36/5. IF (Object_Property_Reference is writable) THEN {
    WAIT (at least two sample times),
    VERIFY Minimum_Value = (a value x: -INF < x < INF),
    VERIFY Average_Value = (a value ≠ NaN),
    VERIFY Maximum_Value = (a value x: Minimum_Value ≤ x < INF),
    VERIFY Attempted_Samples = (a value x ≥ 2),
    VERIFY Valid_Samples = (a value x ≥ 2),
    WRITE Object_Property_Reference = (any new value),
    IF (Samples_are_taken_immediately) THEN {
        VERIFY Attempted_Samples = 1,
        VERIFY Minimum_Value = Average_Value,
        VERIFY Maximum_Value = Average_Value,
        VERIFY Valid_Samples = 1
    }
    ELSE
        VERIFY Attempted_Samples = 0,
        VERIFY Minimum_Value = INF,
        VERIFY Maximum_Value = -INF,
        VERIFY Average_Value = NaN,
        VERIFY Valid_Samples = 0

```

[Change **Clause 7.3.2.4.2**, p. 58]

[Reviewer Note: Modified test as per clarification BTL-CR-0309.]

### 7.3.2.4.2 Managing the Sample Window

Purpose: To verify that an Averaging object correctly tracks the average, minimum, and maximum values attained in a sample. This includes monitoring before and after the sampling window is full.

Test Concept: An Averaging object is configured to monitor a property that can be controlled manually by the testing agent or by the TD. The TD initializes the sample and then monitors the Minimum\_Value, Average\_Value, Maximum\_Value, Attempted\_Samples, and Valid\_Samples properties after each sampling interval to verify that their values are properly tracking the monitored value. This requires the ability to manipulate the values of the monitored property value and a slow enough sampling interval to permit the analysis. This continues until after the sample window is full. ~~If the IUT does not support Averaging object configuration for this Test Concept, then this test shall be omitted.~~

Configuration Requirements: The IUT shall be configured with an Averaging object used to monitor a property that can be controlled by the testing agent or by the TD. The sampling interval shall be configured to allow time to change the monitored property value and to determine if each of the properties Minimum\_Value, Average\_Value, Maximum\_Value, Attempted\_Samples, and Valid\_Samples correctly changes after each sample interval.

Test Steps:

```

1. WRITE Attempted_Samples = 0;
2. VERIFY Attempted_Samples = 0;
3. VERIFY Minimum_Value = INF;
4. VERIFY Maximum_Value = -INF;
5. VERIFY Average_Value = NaN;
6. VERIFY Valid_Samples = 0;
2. READ StartingSample = Valid_Samples + 1
73. REPEAT X = (1 StartingSample to Window_Samples + 5) DO {
    WAIT (Window_Interval / Window_Samples)
    IF (X ≤ Window_Samples) THEN

```

```

    VERIFY Attempted_Samples = X
ELSE
    VERIFY Attempted_Samples = Window_Samples,
    VERIFY Minimum_Value = (the minimum of the monitored values so far),
    VERIFY Maximum_Value = (the maximum of the monitored values so far),
    VERIFY Average_Value = (the average of the monitored values so far),
    IF (X ≤ Window_Samples) THEN
        VERIFY Valid_Samples = X
    ELSE
        VERIFY Valid_Samples = Window_Samples

```

[Change **Clause 7.3.2.8**, p. 62]

### 7.3.2.8 Calendar ~~Test~~*Object Tests*

[Change **Clause 7.3.2.8.1**, p. 62]

[Reviewer Note: Modified test to allow for non-configurable Date\_List properties.]

#### 7.3.2.8.1 Single Date Rollover Test

~~Dependencies: ReadProperty Service Execution Tests, 9.18; TimeSynchronization Service Execution Tests, 9.30; UTCTimeSynchronization Service Execution Tests, 9.31.~~

~~BACnet Reference Clause: 12.9.~~

Purpose: To verify the ability to represent the Calendar status when the Date\_List is in the form of an individual date. Either execution of the TimeSynchronization or the UTCTimeSynchronization service must be supported or another means must be supplied to reset the IUT's clock during the test.

Test Concept: The Calendar object is configured with a Date\_List containing a single date. The IUT's clock is set to the date that immediately precedes the one specified in Date\_List and a time near the end of the day. The test verifies that the Present\_Value of the Calendar object is initially FALSE and that as the time rolls over to the next day the Present\_Value changes to TRUE.

Configuration Requirements: The IUT shall be configured with a Calendar object that contains a Date\_List with a single BACnetCalendarEntry in the form of a Date. *If Date\_List property cannot be configured with a BACnetCalendarEntry in the form of a Date, then this test shall be skipped.*

Test Steps:

1. (TRANSMIT TimeSynchronization-Request,  
     'Time' = (the day preceding the one specified in Date\_List,  
     24:00:00 – **Schedule Evaluation Fail Time** - 1 minute)) |  
 (TRANSMIT UTCTimeSynchronization-Request,  
     'Time' = (the day preceding the one specified in Date\_List,  
     24:00:00 - **Schedule Evaluation Fail Time** - 1 minute converted to UTC)) |  
 MAKE (the local time = 24:00:00 - **Schedule Evaluation Fail Time** - 1 minute)
2. WAIT **Schedule Evaluation Fail Time**
3. VERIFY Present\_Value = FALSE
4. WAIT (**Schedule Evaluation Fail Time** + 2 minutes)
5. VERIFY Present\_Value = TRUE

[Change **Clause 7.3.2.8.2**, p. 63]

[Reviewer Note: Modified test to allow for non-configurable Date\_List properties.]

### 7.3.2.8.2 Date Range Test

~~Dependencies: ReadProperty Service Execution Tests, 9.18; TimeSynchronization Service Execution Tests, 9.30; UTCTimeSynchronization Service Execution Tests, 9.31.~~

~~BACnet Reference Clause: 12.9.~~

Purpose: To verify the ability to represent the Calendar status when the Date\_List is in the form of a BACnetDateRange. Either execution of the TimeSynchronization or the UTCTimeSynchronization service must be supported or another means must be supplied to reset the IUT's clock during the test.

Test Concept: The Calendar object is configured with a Date\_List containing a single BACnetDateRange. The IUT's clock is set to a time and date that is outside of the date range. The Present\_Value is read and verified to be FALSE. The clock is reset to a value within the date range and the Present\_Value is read again to verify that it has the value TRUE. If the IUT can be configured with wildcard fields in the date range then it shall be tested with and without wildcards.

Configuration Requirements: The IUT shall be configured with a Calendar object that contains a Date\_List with a single BACnetCalendarEntry in the form of a BACnetDateRange.—*If Date\_List property cannot be configured with a BACnetCalendarEntry in the form of a BACnetDateRange, then this test shall be skipped.*

Test Steps:

1. (TRANSMIT TimeSynchronization-Request,  
    'Time' = (any day and time outside of the specified date range selected by the tester) ) |  
    (TRANSMIT UTCTimeSynchronization-Request,  
    'Time' = (any day and time outside of the specified date range selected by the tester) ) |  
    MAKE (the local time = any day and time outside of the specified date range selected by the tester)
2. **WAIT Schedule Evaluation Fail Time**
3. **VERIFY Present\_Value = FALSE**
4. (TRANSMIT TimeSynchronization-Request,  
    'Time' = (any day and time inside the specified date range selected by the tester) ) |  
    (TRANSMIT UTCTimeSynchronization-Request,  
    'Time' = (any day and time inside the specified date range selected by the tester) ) |  
    MAKE (the local time = any day and time inside the specified date range selected by the tester)
5. **WAIT Schedule Evaluation Fail Time**
6. **VERIFY Present\_Value = TRUE**

[Change **Clause 7.3.2.8.3**, p. 63]

[Reviewer Note: Modified test to allow for non-configurable Date\_List properties.]

### 7.3.2.8.3 WeekNDay Test

~~Dependencies: ReadProperty Service Execution Tests, 9.18; TimeSynchronization Service Execution Tests, 9.30; UTCTimeSynchronization Service Execution Tests, 9.31.~~

~~BACnet Reference Clause: 12.9.~~

Purpose: To verify the ability to represent the Calendar status when the Date\_List is in the form of a BACnetWeekNDay. Either execution of the TimeSynchronization or the UTCTimeSynchronization service must be supported or another means must be supplied to reset the IUT's clock during the test.

Test Concept: The Calendar object is configured with a Date\_List containing a single BACnetWeekNDay. The IUT's clock is set to a time and date, T1, that matches the BACnetWeekNDay mask. The Present\_Value is read and verified to be TRUE. The clock is reset to a value, T2, that matches the BACnetWeekNDay mask except for the month. The Present\_Value is read and

verified to be FALSE. The clock is reset again to a value, T3, that matches the BACnetWeekNDay mask except for the week of the month. The Present\_Value is read and verified to be FALSE. The clock is reset again to a value, T4, that matches the BACnetWeekNDay mask except for the day of the week. The Present\_Value is read and verified to be FALSE. In between each change, the clock is reset to T1 to force the Present\_Value back to TRUE.

Configuration Requirements: The IUT shall be configured with a Calendar object that contains a Date\_List with a single BACnetCalendarEntry in the form of a BACnetWeekNDay. The BACnetWeekNDay shall be the <sup>1</sup>1th month, last seven days, and Saturday. *If Date\_List property cannot be configured with a BACnetCalendarEntry in the form of a BACnetWeekNDay, then this test shall be skipped.*

Test Steps:

1. (TRANSMIT TimeSynchronization-Request,  
    'Time' = (T1) |  
    (TRANSMIT UTCTimeSynchronization-Request,  
        'Time' = (T1))  
    MAKE (the local time = T1)
2. **WAIT Schedule Evaluation Fail Time**
3. VERIFY Present\_Value = TRUE
4. (TRANSMIT TimeSynchronization-Request,  
    'Time' = (T2) |  
    (TRANSMIT UTCTimeSynchronization-Request,  
        'Time' = (T2)) |  
    MAKE (the local time = T2)
5. **WAIT Schedule Evaluation Fail Time**
6. VERIFY Present\_Value = FALSE
7. (TRANSMIT TimeSynchronization-Request,  
    'Time' = (T1)) |  
    (TRANSMIT UTCTimeSynchronization-Request,  
        'Time' = (T1)) |  
    MAKE (the local time = T1)
8. **WAIT Schedule Evaluation Fail Time**
9. VERIFY Present\_Value = TRUE
10. (TRANSMIT TimeSynchronization-Request,  
    'Time' = (T3) |  
    (TRANSMIT UTCTimeSynchronization-Request,  
        'Time' = (T3)) |  
    MAKE (the local time = T3)
11. **WAIT Schedule Evaluation Fail Time**
12. VERIFY Present\_Value = FALSE
13. (TRANSMIT TimeSynchronization-Request,  
    'Time' = (T1)) |  
    (TRANSMIT UTCTimeSynchronization-Request,  
        'Time' = (T1)) |  
    MAKE (the local time = T1)
14. **WAIT Schedule Evaluation Fail Time**
15. VERIFY Present\_Value = TRUE
16. (TRANSMIT TimeSynchronization-Request,  
    'Time' = (T4) |  
    (TRANSMIT UTCTimeSynchronization-Request,  
        'Time' = (T4)) |  
    MAKE (the local time = T4)
17. **WAIT Schedule Evaluation Fail Time**
18. VERIFY Present\_Value = FALSE

### 7.3.2.9 Command Object Tests

[Change **Clause 7.3.2.9.3**, p. 64]

[Reviewer Note: The purpose of the test is updated after removing the duplicate information. The Test Concept and Configuration Requirements are updated with generic test parameters.]

### 7.3.2.9.3 External Writes Test

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

~~BACnet Reference Clause: 12.10.8.~~

Purpose: To verify that a Command object can write to external objects. ~~If the IUT does not support writing to external objects from a Command object this test shall be omitted.~~

Test Concept: The IUT is configured with a Command object having an action list *X*, that includes writing to an object *O1* in the TD. The TD invokes this action list by writing the appropriate value to the Command object. The TD verifies that the IUT transmits the appropriate WriteProperty-Request.

Configuration Requirements: The IUT shall be configured with a Command object that has an Action property that contains an action list, *X*, that includes a command to write to the ~~Present\_Value~~ *property P1* of ~~(Analog\_Value, 0)~~ *object O1 with the value V1* in the TD.

*Notes to Tester: Any WriteProperty request generated by the IUT may have a Priority parameter. If included, it shall be in the range 1-16, excluding 6.*

Test Steps:

1. WRITE Present\_Value = *X*
2. RECEIVE ~~Simple ACK-PDU~~ BACnet-SimpleACK-PDU
3. VERIFY In\_Process = TRUE
- ~~4.3. BEFORE External Command Fail Time~~  
RECEIVE WriteProperty-Request,  
    'Object Identifier' = ~~(Analog\_Value, 0)~~ *O1*,  
    'Property Identifier' = ~~Present\_Value~~ *P1*,  
    'Property Value' = ~~(any Real value)~~ *V1*
4. ~~VERIFY In\_Process = TRUE~~ TRANSMIT BACnet-SimpleACK-PDU
5. IF (*Post\_Delay* is present)  
    WAIT (*Post\_Delay*)
6. VERIFY In\_Process = FALSE

~~Note to Tester: Any WriteProperty request generated by the IUT may have a Priority parameter. If included, it shall be in the range 1-16, excluding 6.~~

[Change **Clause 7.3.2.9.7**, p. 67]

[Reviewer Note: Changed step 2 to use WRITE. Fixed errata in step 3 to show correct comparison to Protocol\_Revision.]

### 7.3.2.9.7 Write While In\_Process is TRUE Test.

Purpose: To verify that an action list continues to completion if a second action list is commanded while In\_Process is TRUE and that the second action list is not executed.

Test Concept: The IUT is configured with two action lists that include a sequence of externally visible outputs with post delays for each action. The TD triggers the first action list. The external outputs are observed in order to trigger the second action list during the post delay of the first list. The TD triggers the second action list. The external outputs are observed to verify that the second action list is not executed. If the IUT does not support Post Delay, then this test shall be omitted. If the IUT does not support action list configuration, then this test shall be omitted.

Configuration Requirements: The IUT shall be configured with a Command object having two distinct action lists, X and Y, that include writing to a sequence of externally visible outputs. There shall be a post delay between writes to the externally visible outputs that is long enough for the tester to observe the delay (This ensures In\_Process remains TRUE long enough to command the second action list).

Test Steps:

1. WRITE Present\_Value = X
2. ~~TRANSMIT WriteProperty Request,~~  
~~'Object Identifier' = O,~~  
~~'Property Identifier' = Present\_Value,~~  
~~'Property Value' = Y~~
2. WRITE Present\_Value = Y
3. IF (Protocol\_Revision exists and Protocol\_Revision  $\geq$  10) THEN  
RECEIVE BACnet-Error-PDU  
'Error Class' = OBJECT,  
'Error Code' = BUSY  
ELSE  
(RECEIVE (BACnet-Error PDU  
'Error Class' = OBJECT,  
'Error Code' = BUSY)  
|  
(BACnet-Error-PDU  
'Error Class' = SERVICES,  
'Error Code' = SERVICE\_REQUEST\_DENIED | OTHER)
4. CHECK (that the externally visible actions of X took place)
5. CHECK (that the externally visible actions of Y did not take place)
6. VERIFY In\_Process = FALSE,
7. VERIFY All\_Writes\_Successful = TRUE

[Change **Clause 7.3.2.10**, p. 70]

### 7.3.2.10 Device Object Tests

~~Most necessary tests for functionality of the Device object are covered by tests for the application service or special functionality to which they correspond. These are the tests for the Device object. Other tests for functionality of the Device object are covered by tests for the application service or special functionality to which they correspond.~~

[Change **Clause 7.3.2.10.1**, p. 70]

[Reviewer Note: IC135-2012-18 ruled that the increment shall not be in the Active\_COV\_Subscriptions property value if the property is not numeric; present if a valid Increment was provided in the subscription; and optionally present otherwise.]

#### 7.3.2.10.1 Active\_COV\_Subscriptions SubscribeCOV Test

Purpose: This test case verifies that the IUT correctly updates the Active\_COV\_Subscriptions property when COV subscriptions are created, cancelled and timed-out using SubscribeCOV.

*Test Concept: INC<sub>1</sub>, INC<sub>2</sub>, and INC<sub>3</sub> are each not present if the property is not numeric; present if a valid Increment was provided in the subscription; and optionally present otherwise.*

Configuration Requirements: In this test, the tester shall choose three standard objects, O<sub>1</sub>, O<sub>2</sub>, and O<sub>3</sub>, for which the device supports SubscribeCOV. O<sub>1</sub>, O<sub>2</sub>, and O<sub>3</sub> are not required to refer to different objects. The tester shall also choose three non-zero unique process identifiers, P<sub>1</sub>, P<sub>2</sub>, and P<sub>3</sub>, and three non-zero lifetimes L<sub>1</sub>, L<sub>2</sub>, and L<sub>3</sub>. Lifetime L<sub>1</sub> shall be long enough to allow the initial part of the test to run through to step 14. Lifetimes L<sub>2</sub> and L<sub>3</sub> shall be long enough for the whole test to be completed without expiring.



The IUT shall start the test with no entries in its Active\_COV\_Subscriptions property.

Test Steps:

1. TRANSMIT SubscribeCOV-Request,  
     'Subscriber Process Identifier' = P<sub>1</sub>,  
     'Monitored Object Identifier' = O<sub>1</sub>,  
     'Issue Confirmed Notifications' = TRUE,  
     'Lifetime' = L<sub>1</sub>
2. RECEIVE BACnet-SimpleACK-PDU
3. BEFORE **Notification Fail Time**  
     RECEIVE ConfirmedCOVNotification-Request,  
     'Subscriber Process Identifier' = P<sub>1</sub>,  
     'Initiating Device Identifier' = IUT,  
     'Monitored Object Identifier' = O<sub>1</sub>,  
     'Time Remaining' = (a value approximately equal to L<sub>1</sub>),  
     'List of Values' = (values appropriate to the object type of the monitored object)
4. TRANSMIT BACnet-SimpleACK-PDU
5. IF P<sub>1</sub> is numeric THEN  
     VERIFY Active\_COV\_Subscriptions = { { {TD, P<sub>1</sub>}, {O<sub>1</sub>, Present\_Value }, TRUE, (a value less than L<sub>1</sub>),  
     (INC<sub>1</sub> : not present or a valid Increment if the property is REAL) } }  
     ELSE  
     VERIFY Active\_COV\_Subscriptions = { { {TD, P<sub>1</sub>, { O<sub>1</sub>, Present\_Value }, TRUE, (a value less than L<sub>1</sub>), (INC<sub>1</sub>: not present)}}  
     present)}}  
- 6. TRANSMIT SubscribeCOV-Request,  
     'Subscriber Process Identifier' = P<sub>2</sub>,  
     'Monitored Object Identifier' = O<sub>2</sub>,  
     'Issue Confirmed Notifications' = FALSE,  
     'Lifetime' = L<sub>2</sub>
- 7. RECEIVE BACnet-SimpleACK-PDU
- 8. BEFORE **Notification Fail Time**  
     RECEIVE UnconfirmedCOVNotification-Request,  
     'Subscriber Process Identifier' = P<sub>2</sub>,  
     'Initiating Device Identifier' = IUT,  
     'Monitored Object Identifier' = O<sub>2</sub>,  
     'Time Remaining' = (a value approximately equal to L<sub>2</sub>),  
     'List of Values' = (values appropriate to the object type of the monitored object)
- 9. VERIFY Active\_COV\_Subscriptions = { { {TD, P<sub>1</sub>}, {O<sub>1</sub>, Present\_Value}, TRUE, (a value less than L<sub>1</sub>),  
     INC<sub>1</sub>(a valid Increment if the property is REAL) },  
     { {TD, P<sub>2</sub>}, {O<sub>2</sub>, Present\_Value}, FALSE, (a value less than L<sub>2</sub>),  
     (INC<sub>2</sub>: not present if the property is not numeric; present  
     if a valid Increment was provided in the subscription;  
     optionally present otherwise if the property is REAL) } }  
- 10. TRANSMIT SubscribeCOV-Request,  
     'Subscriber Process Identifier' = P<sub>3</sub>,  
     'Monitored Object Identifier' = O<sub>3</sub>,  
     'Issue Confirmed Notifications' = FALSE,  
     'Lifetime' = L<sub>3</sub>
- 11. RECEIVE BACnet-SimpleACK-PDU
- 12. BEFORE **Notification Fail Time**  
     RECEIVE UnconfirmedCOVNotification-Request,  
     'Subscriber Process Identifier' = P<sub>3</sub>,  
     'Initiating Device Identifier' = IUT,  
     'Monitored Object Identifier' = O<sub>3</sub>,  
     'Time Remaining' = (a value approximately equal to L<sub>3</sub>),  
     'List of Values' = (values appropriate to the object type of the monitored object)
- 13. IF P<sub>3</sub> is numeric THEN  
     VERIFY Active\_COV\_Subscriptions = { { {TD, P<sub>1</sub>}, {O<sub>1</sub>, Present\_Value}, TRUE, (a value less than L<sub>1</sub>),  
     INC<sub>1</sub>(a valid Increment if the property is REAL) },  
     { {TD, P<sub>2</sub>}, {O<sub>2</sub>, Present\_Value}, FALSE, (a value less than L<sub>2</sub>),

~~*INC<sub>2</sub>(a valid Increment if the property is REAL)*~~,  
~~*{{TD, P<sub>3</sub>}, {O<sub>3</sub>, Present\_Value}, FALSE, (a value less than L<sub>3</sub>)}*~~,  
~~*INC<sub>3</sub>: not present or (a valid Increment if the property is REAL)*~~}}

*ELSE*  
*VERIFY Active\_COV\_Subscriptions* = *{{{{TD, P<sub>1</sub>}, {O<sub>1</sub>, Present\_Value}, TRUE, (a value less than L<sub>1</sub>)}, INC<sub>1</sub> },*  
*{{TD, P<sub>2</sub>}, {O<sub>2</sub>, Present\_Value}, FALSE, (a value less than L<sub>2</sub>)}, INC<sub>2</sub> },*  
*{{TD, P<sub>3</sub>}, {O<sub>3</sub>, Present\_Value}, FALSE, (a value less Than L<sub>3</sub>)}, (INC<sub>3</sub>: not present)}}*

14. WAIT L<sub>1</sub> + the IUT's timer granularity  
15. *VERIFY Active\_COV\_Subscriptions* = *{{{{TD, P 2 }, {O 2 , Present\_Value}, FALSE, (a value less than L 2 ),*  
*INC<sub>2</sub> (a valid Increment if the property is REAL)*},  
*{{TD, P 3 }, {O 3 , Present\_Value}, FALSE, (a value less than L 3 ),*  
*INC<sub>3</sub>(a valid Increment if the property is REAL)}}*

16. *TRANSMIT SubscribeCOV-Request,*  
*'Subscriber Process Identifier' = P<sub>3</sub> ,*  
*'Monitored Object Identifier' = O<sub>3</sub>*  
17. *RECEIVE BACnet-SimpleACK-PDU*  
18. *VERIFY Active\_COV\_Subscriptions* = *{{{{TD, P 2 }, {O 2 , Present\_Value}, FALSE, (a value less than L 2 ),*  
*INC<sub>2</sub>(a valid Increment if the property is REAL) }}*

19. *TRANSMIT SubscribeCOV-Request,*  
*'Subscriber Process Identifier' = P<sub>2</sub> ,*  
*'Monitored Object Identifier' = O<sub>2</sub>*  
20. *RECEIVE BACnet-SimpleACK-PDU*  
21. *VERIFY Active\_COV\_Subscriptions* = { }

[Change **Clause 7.3.2.10.6**, p. 73]

### 7.3.2.10.6 Successful Increment of the Database\_Revision Property after Changing the Object\_Identifier Property of an Object

Purpose: To verify that the Database\_Revision property of the Device object increments after changing the Object\_Identifier property of an object. If the Object\_Identifier property of an object cannot be changed, this test shall be omitted.

Test Concept: The Database\_Revision property of the Device object is read. An object's ~~name~~ *Object\_Identifier property* is changed. The Database\_Revision property of the Device object is read again to verify that it incremented.

Configuration Requirements: none.

Test Steps:

1. TRANSMIT ReadProperty-Request,  
'Object Identifier' = (the Device object),  
'Property Identifier' = Database\_Revision
2. RECEIVE ReadProperty-ACK,  
'Object Identifier' = (the Device object),  
'Property Identifier' = Database\_Revision,  
'Property Value' = (any value = initial value)
3. MAKE (the Object\_Identifier property of an object change)
4. TRANSMIT ReadProperty-Request,  
'Object Identifier' = (the Device object),  
'Property Identifier' = Database\_Revision
5. RECEIVE ReadProperty-ACK,  
'Object Identifier' = (the Device object),  
'Property Identifier' = Database\_Revision,  
'Property Value' = (greater than initial value)

[Add new **Clause 7.3.2.10.X2**, p. 74]

### **7.3.2.10.X2 Max\_Segments\_Accepted at least the minimum**

Purpose: To verify that the IUT correctly implements the Max\_Segments\_Accepted property value when it does support segmentation.

Configuration Requirements: If the IUT cannot be configured to support segmentation, then this test shall be skipped.

Test Steps:

1. IF Segmentation\_Supported == SEGMENTED\_TRANSMIT or Segmentation\_Supported == SEGMENTED\_NONE  
THEN  
VERIFY Max\_Segments\_Accepted = 1  
ELSE  
VERIFY (Max\_Segments\_Accepted > 1)

[Add new **Clause 7.3.2.10.X**, p. 74]

### **7.3.2.10.X Ensure UTC\_Offset is Configurable**

Purpose: This test verifies UTC\_Offset is configurable and accepts from -1440 to +1440.

Test Concept: For each value in a set of valid values across the acceptable range for UTC\_Offset, verify that the UTC\_Offset can be configured with the value.

Configuration Requirements: If the Protocol\_Revision of the device is less than 18, this test shall be skipped.

Test Steps:

1. REPEAT UO = (-1440, -780, 0, +780, +1440, and 2 other values which are multiples of 15 minutes) DO {  
IF (UTC\_Offset is writable) THEN  
WRITE UTC\_Offset = UO  
ELSE  
MAKE UTC\_Offset = UO  
VERIFY (Device Object) UTC\_Offset = UO  
}

[Add new **Clause 7.3.2.10.X3**, p. 74]

### **7.3.2.10.X3 Ensure Device Object\_Name is Configurable**

Purpose: This test verifies Device Object\_Name value is configurable as per BACnet Clause 12.1.1.4.

Configuration Requirements: none.

Test Steps:

1. READ S = (Device, IUT), Object\_Name
2. MAKE (Configure the device Object\_Name to a value other than S)
3. VERIFY (Device, IUT), Object\_Name <> S

[Add new **Clause 7.3.2.10.X4**, p. 74]

#### 7.3.2.10.X4 Ensure Device Object\_Identifier is Configurable

Purpose: This test verifies Device Object\_Identifier property value is configurable as per BACnet Clause 12.1.1.3.

Configuration Requirements: none.

Test Steps:

1. READ S = (Device, IUT), Object\_Identifier
2. MAKE (Configure the device Object\_Identifier to a value other than S)
3. VERIFY (Device, IUT), Object\_Identifier  $\neq$  S

#### 7.3.2.13 Global Group Object Tests

[Add new **Clause 7.3.2.13.X1**, p. 77]

##### 7.3.2.13.X1 Global Group Present\_Value, Out\_Of\_Service and Status\_Flags Test

Purpose: This test verifies the interrelationship between the Present\_Value, Out\_Of\_Service and Status\_Flags properties of a Global Group object.

Test Concept: Verify the Present\_Value stops updating when Out\_Of\_Service is TRUE.

Configuration Requirements: The IUT shall be configured with a Global Group object with the Group\_Members property containing a member M1 at index N1 that has a value that can be changed. W1 is the maximum time it takes for the Global Group to receive an update from M1.

Test Steps:

1. MAKE (Out\_Of\_Service = TRUE)
2. VERIFY Out\_Of\_Service = TRUE
3. VERIFY Status\_Flags = {?, ?, FALSE, TRUE}
4. X1 = READ Present\_Value, ARRAY INDEX = N1
5. MAKE (M1 value change)
6. WAIT (W1)
7. X2 = READ Present\_Value, ARRAY INDEX = N1
8. VERIFY X1 = X2

[Add new **Clause 7.3.2.13.X2**, p. 77]

##### 7.3.2.13.X2 Reliability MEMBER\_FAULT Test

Purpose: This test case verifies the FAULT flag of the Member\_Status\_Flags is TRUE, and the Reliability property is equal to MEMBER\_FAULT when a member of the Group\_Members property goes into FAULT.

Test Concept: Force a member of the Group\_Members property to enter a Fault condition and verify the Member\_Status\_Flags FAULT flag equals TRUE and Reliability equals MEMBER\_FAULT.

Configuration Requirements: The IUT shall be configured with a Global Group object with the Group\_Members property containing a member M1 at index N1 that has a value that can be made to indicate a fault condition (see Notes To Tester). The Out\_Of\_Service property of the Global Group object must remain FALSE throughout the test. W1 is the maximum time it takes for the Global Group to receive an update from M1.

Notes to Tester: Member\_Status\_Flags FAULT flag will remain TRUE and the Reliability property will change to MEMBER\_FAULT when a member of the Group\_Members property goes into fault.

Test Steps:

1. MAKE (M1 Status\_Flags = {?, TRUE, ?, ?})
2. WAIT (W1)
3. VERIFY Member\_Status\_Flags = {?, TRUE, ?, ?}
4. IF (Reliability is present) THEN  
    VERIFY Reliability = MEMBER\_FAULT

[Add new **Clause 7.3.2.13.X3**, p. 77]

#### **7.3.2.13.X3 Reliability COMMUNICATION\_FAILURE Test**

Purpose: This test case verifies that the Member\_Status\_Flags FAULT flag will remain FALSE while the Reliability property is COMMUNICATION\_FAILURE.

Test Concept: Force a member of the Group\_Members property to stop communicating and verify the Reliability property equals COMMUNICATION\_FAILURE and the Member\_Status\_Flags FAULT flag remains FALSE.

Configuration Requirements: The IUT shall be configured with a Global Group object with the Group\_Members containing a member M1 at index N1 that can be made to discontinue communications and also respond with an error such as OBJECT/UNKNOWN\_OBJECT. (See Notes To Tester). The Out\_Of\_Service property of the Global Group object must remain FALSE throughout the test. W1 is the maximum time it takes for the Global Group to receive an update from M1.

Notes to Tester: Reliability will change to COMMUNICATION\_FAILURE when a member is no longer able to communicate its Status\_Flags property. This can occur when the device goes offline or the object is deleted within the device.

Test Steps:

1. MAKE (M1 fail (communications or error))
2. WAIT (W1)
3. VERIFY Reliability = COMMUNICATION\_FAILURE
4. IF (Reliability is present) THEN  
    VERIFY Reliability = COMMUNICATION\_FAILURE
5. VERIFY Member\_Status\_Flags = {?, FALSE, ?, ?}

[Add new **Clause 7.3.2.13.X4**, p. 77]

#### **7.3.2.13.X4 Present\_Value Tracking and Reliability Test**

Purpose: This test verifies that the Global Group object continues to update its Present\_Value independent of the state of the Reliability property.

Test Concept: While the Reliability property is not NO\_FAULT\_DETECTED verify the Present\_Value continues to update.

Configuration Requirements: The IUT shall be configured with a Global Group object with its Reliability not equal to NO\_FAULT\_DETECTED and a Group\_Members member M1 at index N1 that can be changed. W1 is the maximum time it takes for the Global Group to receive an update from M1.

Notes to Tester: Reliability will change to COMMUNICATION\_FAILURE when a member is no longer able to communicate its Status\_Flags property. This can occur when the device goes offline or the object is deleted within the device. Also, the Reliability property will change to MEMBER\_FAULT when a member of the Group\_Members property goes into fault.

Test Steps:

1. VERIFY Reliability  $\diamond$  NO\_FAULT\_DETECTED
2. MAKE (M1 = X1)
3. WAIT (W1)
4. X2 = READ Present\_Value, ARRAY INDEX = N1
5. VERIFY X1 = X2

[Add new **Clause 7.3.2.13.X5**, p. 77]

#### **7.3.2.13.X5 Present\_Value Tracking Test**

Purpose: This test verifies that the Global Group object tracks the value of the monitored properties value and data type.

Test Concept: Make a member of the Group\_Members property change value and verify the Present\_Value updates to match that value.

Configuration Requirements: The IUT shall be configured with a Global Group object with the Group\_Members containing a member M1 at index N1 of the specified data type that can be changed. W1 is the maximum time it takes for the Global Group to receive an update from M1.

Test Steps:

1. MAKE (M1 = X1)
2. WAIT (W1)
3. X2 = READ Present\_Value, ARRAY INDEX = N1
4. VERIFY X1 = X2

[Add new **Clause 7.3.2.13.X6**, p. 77]

#### **7.3.2.13.X6 COVU\_Period and COVU\_Recipients Zero Test**

Purpose: To verify that object O1 does not initiate UnconfirmedCOVNotification service requests when COVU\_Period is zero or COVU\_Recipients contains an empty list.

Test Concept: Configure O1 to produce unsubscribed UnconfirmedCOVNotifications, set COVU\_Period to zero, and attempt to produce unsubscribed UnconfirmedCOVNotifications. Repeat with COVU\_Recipients containing an empty list.

Configuration Requirements: At the start of the test, O1 shall be configured with a non-zero COVU\_Period and a non-empty COV\_Recipient property.

Test Steps:

1. MAKE (O1 issue an unsubscribed UnconfirmedCOVNotification)
2. BEFORE **Notification Fail Time**  
RECEIVE UnconfirmedCOVNotification-Request,  
DESTINATION = (any valid address),  
'Subscriber Process Identifier' = 0,  
'Initiating Device Identifier' = IUT,  
'Monitored Object Identifier' = O1,  
'Time Remaining' = 0,  
'List of Values' = (any valid set of values)
3. WRITE (COVU\_Period = 0)
4. MAKE (O1 a condition that would normally cause the IUT to issue an unsubscribed UnconfirmedCOVNotification)
5. WAIT **Notification Fail Time** times 2
6. CHECK (that O1 has not transmitted an UnconfirmedCOVNotification-Request)
7. WRITE (COVU\_Period  $\diamond$  0)
8. MAKE (O1 issue an unsubscribed UnconfirmedCOVNotification)

#### 9. BEFORE **Notification Fail Time**

RECEIVE UnconfirmedCOVNotification-Request,  
 DESTINATION = (any valid address),  
 'Subscriber Process Identifier' = 0,  
 'Initiating Device Identifier' = IUT,  
 'Monitored Object Identifier' = O1,  
 'Time Remaining' = 0,  
 'List of Values' = (any valid set of values)

10. WRITE (COVU\_Recipients an empty list)

11. MAKE (O1 a condition that would normally cause the IUT to issue an unsubscribed UnconfirmedCOVNotification)

12. WAIT **Notification Fail Time** times 2

13. CHECK (that O1 has not transmitted an UnconfirmedCOVNotification-Request)

### 7.3.2.15 Life Safety Point Object Tests

[Add new **Clause 7.3.2.15.X5**, p. 78]

#### 7.3.2.15.X5 Writable Tracking\_Value

Purpose: This test case verifies that Tracking\_Value is writable when Out\_Of\_Service is TRUE.

Test Concept: It verifies the interrelationship between the Tracking\_Value, Status Flags, and Present\_Value properties. If the Out\_Of\_Service property of the object under test is not writable, and the value of the property cannot be changed by other means, then this test shall be omitted. This test applies to Life Safety Zone and Life Safety point object.

The TD will select one instance of each appropriate object type and test it as described.

Test Steps:

1. MAKE (Out\_Of\_Service TRUE)
2. VERIFY Out\_Of\_Service = TRUE
3. VERIFY Status\_Flags = (?, FALSE, ?, TRUE)
4. MAKE (Event\_State = Normal)
5. VERIFY Event\_State = Normal
6. WRITE Tracking\_Value = (X: any value that corresponds to an Event\_State of NORMAL)
7. VERIFY Tracking\_Value = X
8. VERIFY Present\_Value = X

[Add new **Clause 7.3.2.15.X6**, p. 78]

#### 7.3.2.15.X6 Supports Writable Mode Property

Purpose: To verify that the Mode property takes one of the values found in the Accepted\_Modes property.

Test Concept: It verifies the interrelationship between the Mode and Accepted\_Modes properties. This test applies to Life Safety Zone and Life Safety point object. The IUT will select one instance of each appropriate object type and test it as described.

Test Steps:

1. READ AM = Accepted\_Modes
2. TRANSMIT WriteProperty-Request  
 'Object Identifier' = (the object being tested),  
 'Property Identifier' = Mode,

- 'Property Value' = (X: Any valid value from list of AM)
3. RECEIVE BACnet-SimpleACK-PDU
  4. VERIFY Mode = X
  5. TRANSMIT WriteProperty-Request
    - 'Object Identifier' = (the object being tested),
    - 'Property Identifier' = Mode,
    - 'Property Value' = (X: Any invalid value, which is not present in AM)
  6. RECEIVE BACnet-Error-PDU,
    - 'Error Class' = PROPERTY,
    - 'Error Code' = VALUE\_OUT\_OF\_RANGE

[Add new **Clause 7.3.2.15.X7**, p. 78]

### 7.3.2.15.X7 Support Operation\_Expected Property

Purpose: To verify that the Operation\_Expected property takes on the value of ConfirmedEventNotification-Request.

Test Concept: It verifies the interrelationship between the Operation\_Expected property and ConfirmedEventNotification-Request. This test applies to Life Safety Zone and Life Safety point object. The IUT will select one instance of each appropriate object type and test it as described.

Test Steps:

1. MAKE (the IUT send an ConfirmedEventNotification)
2. RECEIVE ConfirmedEventNotification-Request,
  - 'Process Identifier' = (any valid process identifier),
  - 'Initiating Device Identifier' = TD,
  - 'Event Object Identifier' = (any Life-Safety object),
  - 'Time Stamp' = (the current local time),
  - 'Notification Class' = (any valid notification class),
  - 'Priority' = (any valid priority),
  - 'Event Type' = CHANGE-OF-LIFE-SAFETY,
  - 'Message Text' = (any character string),
  - 'Notify Type' = ALARM | EVENT,
  - 'AckRequired' = TRUE | FALSE,
  - 'From State' = NORMAL,
  - 'To State' = (any non-normal state appropriate to the event type),
  - 'Event Values' = (New State: (Any Valid State), New-Mode: (Any Valid Mode),  
Status-Flag: (TRUE, FALSE, ?, ?),  
Operation\_Expected: (X: Any Valid operation))
3. VERIFY Operation\_Expected = X ( operation expected in the step 2)

[Add new **Clause 7.3.2.15.X8**, p. 78]

### 7.3.2.15.X8 Support Writable Member\_Of Property

Purpose: To verify that the Member\_Of property takes only supported values of the life safety objects within the IUT.

Test Concept: If the property is writable and is restricted to referencing objects within the containing device, an attempt to write a reference to an object outside the containing device into this property shall cause a Result (-). If the property is not writable and if the value of the property cannot be changed by other means, then this test shall be omitted. The IUT will select one instance of each appropriate object type, O1, and test it as described.

Test Steps:



1. TRANSMIT WriteProperty-Request,  
    'Object Identifier' = (O1),  
    'Property Identifier' = Member\_Of  
    'Property Value' = (X: any valid life safety zone object reference)
2. RECEIVE BACnet-SimpleACK-PDU,
3. TRANSMIT ReadProperty-Request,  
    'Object Identifier' = (O1),  
    'Property Identifier' = Member\_Of
4. RECEIVE ReadProperty-ACK,  
    'Object Identifier' = (the object being tested),  
    'Property Identifier' = Member\_Of  
    'Property Value' = X

[Add new **Clause 7.3.2.15.X9**, p. 78]

### **7.3.2.15.X9 Silenced Property Test**

Purpose: This test verifies the behavior of Silenced property.

Test Concept: Verify the interrelationship between the Silenced property and any audible or visual indication that has been silenced by the receipt of a LifeSafetyOperation service request or a local process. If the Silenced property of the object under test is unchanging by means of a LifeSafetyOperation service requests because none of the silencing operations are supported, then this test shall be omitted. This test applies to Life Safety Zone and Life Safety Point object.

Since the result of any specific LifeSafetyOperation is a local matter, the expected actions when an operation is applied to an object is a local matter. In order to apply this test, the tester selects an initial Silenced state and a BACnetLifeSafetyOperation. The tester then verifies that the expected Silenced state, as specified by the vendor, is the result of the life safety operation on the object.

The tester will select one instance of each appropriate object type and test it as described.

Test Steps:

1. InitialSilencedState = READ Silenced
2. VERIFY Silenced = Other than InitialSilencedState
3. TRANSMIT LifeSafetyOperation-Request,  
    'Requesting Process Identifier' = (any valid identifier),  
    'Requesting Source' = (any valid character string),  
    'Request' = (any supported LifeSafetyOperation request transmitted to silence the sounder/strobe),  
    'Object Identifier' = (the selected object)
4. RECEIVE BACnet-SimpleACK-PDU
5. CHECK (Sounder/Strobe inactive)
6. ResultingSilencedState = READ Silenced
7. CHECK (the ResultingSilencedState is equal to the InitialSilencedState, modified by the LifeSafetyOperation request transmitted)

[Change **Clause 7.3.2.22**, p. 96]

### **7.3.2.22 Program Object Tests**

~~The Program object makes parameters of a custom program network visible. Since BACnet does not define the functionality of the program there are no standard tests to verify this functionality. The Program object utilizes parameter control through its writable Program\_Change property.~~

[Add new **Clause 7.3.2.22.1**, p. 96]

[Reviewer Note: No tests exist in Clause 7.3.2.22.]

#### **7.3.2.22.1 Program\_Change property test**

Purpose: To verify writability of Program\_Change property.

Test Concept: The Program\_Change property is set to a value other than READY and then it and the Program\_State property are verified to update correctly.

Configuration Requirements: The Program\_Change property of the program object being tested shows a value of READY.

Notes to Tester: In step 2, depending on the current Program\_State, and the implementation, certain requested values for Program\_Change may be invalid and would return a Result(-) if an attempt were made to write them.

Test Steps:

1. VERIFY Program\_Change = READY
2. WRITE Program\_Change = (a value other than READY)
3. WAIT (for the processing to consume that value written to Program\_Change)
4. VERIFY Program\_Change = READY
5. VERIFY Program\_State = the new state reflected, based upon value written to Program\_Change in step 2.

#### **7.3.2.23 Schedule Object Tests**

[Change **Clause 7.3.2.23.6**, p. 106]

[Reviewer Note: Corrected the Configuration Requirements to allow the test to be executed on devices greater than or equal to Protocol\_Revision 4.]

##### **7.3.2.23.6 Weekly\_Schedule Restoration Test**

~~Dependencies: ReadProperty Service Execution Tests, 9.18; ReinitializeDevice Service Execution Tests, 9.27; TimeSynchronization Service Execution Tests, 9.30; UTCTimeSynchronization Service Execution Tests, 9.31.~~

~~BACnet Reference Clause: 12.24.4, 12.24.7, 12.24.9.~~

Purpose: To verify the restoration behavior in a Weekly\_Schedule.

Test Concept: The IUT is configured with a Schedule object containing a Weekly\_Schedule with a BACnetDailySchedule that has multiple BACnetTimeValue entries that do not include the time 00:00. There shall be no Exception\_Schedule that overrides this Weekly\_Schedule during the date and time used for this test. The local date and time are changed to a value between 00:00 and the first entry in the BACnetDailySchedule. Present\_Value is read to verify that it contains the Schedule\_Default value, or V<sub>last</sub> for implementations with a Protocol\_Revision less than 4. The IUT is reset and the Present\_Value is checked again to verify that it contains the Schedule\_Default value, or V<sub>last</sub> for implementations with a Protocol\_Revision less than 4.

Configuration Requirements: The IUT shall be configured with a Schedule object that contains a Weekly\_Schedule that has more than one scheduled write operation for a particular day. None of the write operations shall be scheduled for time 00:00 and there shall be no higher priority coincident BACnetSpecialEvents. In the test description D<sub>1</sub> represents a time between 00:00 and the time of the first scheduled write operation in the BACnetDailySchedule. V<sub>last</sub> represents the value that is scheduled to be written in the last BACnetTimeValue pair for the day. ~~This test shall not be performed if the Protocol\_Revision property is present in the Device object and has a value of 4 or greater.~~

Test Steps:

1. (TRANSMIT TimeSynchronization-Request, 'Time' = D<sub>1</sub>) |  
(TRANSMIT UTCTimeSynchronization-Request 'Time' = D<sub>1</sub>) |  
MAKE (the local date and time = D<sub>1</sub>)
2. WAIT **Schedule Evaluation Fail Time**
3. IF (Protocol\_Revision is present and Protocol\_Revision ≥ 4) THEN  
VERIFY Present\_Value = Schedule\_Default  
ELSE  
VERIFY Present\_Value = V<sub>last</sub>
4. IF (ReinitializeDevice execution is supported) THEN  
TRANSMIT ReinitializeDevice-Request,  
'Reinitialized State of Device' = WARMSTART,  
'Password' = (any valid password)  
RECEIVE ~~BACnet-Simple-ACK-PDU~~ BACnet-SimpleACK-PDU  
ELSE  
MAKE (the IUT reinitialize)
5. CHECK (Did the IUT perform a WARMSTART reboot?)
6. WAIT **Schedule Evaluation Fail Time**
7. IF (Protocol\_Revision is present and Protocol\_Revision ≥ 4) THEN  
VERIFY Present\_Value = Schedule\_Default  
ELSE  
VERIFY Present\_Value = V<sub>last</sub>

### 7.3.2.23.10 Schedule Object Protocol\_Revision 4 Tests

#### 7.3.2.23.10.3 Revision 4 Exception\_Schedule Property Tests

[Change **Clause 7.3.2.23.10.3.6**, p. 117]

[Reviewer Note: New functionality testing WeekNDay new WeekOfMonth entries added in 135-2012bg-4 for Protocol\_Revision 18.]

#### 7.3.2.23.10.3.6 Revision 4 Calendar Entry WeekNDay ~~Last~~ Special Week Of Month Test

Purpose: To verify that a date matching a WeekNDay's WeekOfMonth field in an Exception\_Schedule enables the referencing Schedule object.

Test Concept: The IUT's local date and time are changed to values that are selected by the TD based on the criteria in Table 7-16. The value of the Present\_Value property is monitored to verify that the scheduled write operations occur.

Configuration Requirements: The IUT shall be configured to contain a Schedule object, S, with an Exception\_Schedule containing a BACnetCalendarEntry with a WeekNDay entry specifying the *one of the special week of month values, WM [6,7,8,9]* ~~last week of the month~~. The criteria for the dates used in the test are given in Table 7-16. The local date and time shall be set such that the Present\_Value property has a value other than V<sub>1</sub>.

**Table 7-16.** Criteria for Calendar Entry WeekNDay ~~LastSpecial~~ Week Of Month Test Dates and Values

Date	Criteria	Value
D <sub>1</sub>	1. Date occurs during Effective_Period, 2A. BACnetSpecialEvent incorporates calendarEntry: WeekNDay, 2B. calendarEntry: WeekNDay specifies WeekOfMonth, 2C. calendarEntry: WeekNDay: WeekOfMonth has the value <del>6WM</del> , 2D. Date is in the <del>last week of the month</del> <i>WeekOfMonth specified by 2C</i> , 2E. WeekNDay:Month matches the specified month, 2F. WeekNDay:dayOfWeek matches the specified day of the week, 2G. Time is on or after the time of the entry with V <sub>1</sub> , but before any other entry in the Exception_Schedule, and 2H. BACnetSpecialEvent has a higher eventPriority than any coincident BACnetSpecialEvent records.	V <sub>1</sub>
D <sub>2</sub>	1. Date occurs during Effective_Period, 2A. BACnetSpecialEvent incorporates calendarEntry: WeekNDay, 2B. calendarEntry: WeekNDay specifies WeekOfMonth, 2C. calendarEntry: WeekNDay: WeekOfMonth has the value <del>6WM</del> , and 2D. Date is not in <del>the last week of the month</del> <i>the WeekOfMonth specified by 2C</i> .	V <sub>2</sub> different from V <sub>1</sub>

Test Steps:

1. VERIFY (S), Present\_Value = any value other than V<sub>1</sub>
2. (TRANSMIT TimeSynchronization-Request, 'Time' = D<sub>1</sub>) |  
(TRANSMIT UTCTimeSynchronization-Request, 'Time' = D<sub>1</sub>) |  
MAKE (the local date and time = D<sub>1</sub>)
3. WAIT **Schedule Evaluation Fail Time**
4. VERIFY S, Present\_Value = V<sub>1</sub>
5. (TRANSMIT TimeSynchronization-Request, 'Time' = D<sub>2</sub>) |  
(TRANSMIT UTCTimeSynchronization-Request, 'Time' = D<sub>2</sub>) |  
MAKE (the local date and time = D<sub>2</sub>)
6. WAIT **Schedule Evaluation Fail Time**
7. VERIFY S, Present\_Value = V<sub>2</sub>

[Change **Clause 7.3.2.23.10.3.7**, p. 118]

[Reviewer Note: Added clarifying text to Table 7-16.1.]

#### 7.3.2.23.10.3.7 Revision 4 Calendar Entry WeekNDay Day Of Week Test

~~Dependencies: ReadProperty Service Execution Tests, 9.18; TimeSynchronization Service Execution Tests, 9.30, UTCTimeSynchronization Service Execution Tests, 9.31.~~

~~BACnet Reference Clause: 12.24.8.~~

Purpose: To verify that a date matching a WeekNDay's DayOfWeek field in an Exception\_Schedule enables the referencing Schedule object.

Test Concept: The IUT's local date and time are changed to values that are selected by the TD based on the criteria in Table 7-10. The value of the Present\_Value property is monitored to verify that the scheduled write operations occur.

Configuration Requirements: The IUT shall be configured to contain a Schedule object with an Exception\_Schedule containing a BACnetCalendarEntry with a WeekNDay entry specifying the day of the week. The criteria for the dates used in the test are given in Table 7-10. The local date and time shall be set such that the Present\_Value property has a value other than V<sub>1</sub>.

**Table 7-16.1.** Criteria for Calendar Entry WeekNDay Day of Week Test Dates and Values

Date	Criteria	Value
D <sub>1</sub>	1. Date occurs during Effective_Period, 2A. BACnetSpecialEvent incorporates calendarEntry: WeekNDay, 2B. calendarEntry: WeekNDay specifies only DayOfWeek, 2C. Date falls on the specified day of the week, and 2D. Higher eventPriority than any coincident BACnetSpecialEvents.	V <sub>1</sub>
D <sub>2</sub>	1. Date occurs during Effective_Period, 2A. BACnetSpecialEvent incorporates calendarEntry: WeekNDay, 2B. calendarEntry: WeekNDay specifies only DayOfWeek, and 2C. Date does not fall on the specified day of the week.	V <sub>2</sub> <i>(a value different from the Present_Value expected at D<sub>2</sub>)</i>

#### Test Steps:

1. VERIFY Present\_Value = (any value other than V<sub>1</sub>)
2. (TRANSMIT TimeSynchronization-Request, 'Time' = D<sub>1</sub>) |  
(TRANSMIT UTCTimeSynchronization-Request, 'Time' = D<sub>1</sub>) |  
MAKE (the local date and time = D<sub>1</sub>)
3. WAIT **Schedule Evaluation Fail Time**
4. VERIFY Present\_Value = V<sub>1</sub>
5. (TRANSMIT TimeSynchronization-Request, 'Time' = D<sub>2</sub>) |  
(TRANSMIT UTCTimeSynchronization-Request, 'Time' = D<sub>2</sub>) |  
MAKE (the local date and time = D<sub>2</sub>)
6. WAIT **Schedule Evaluation Fail Time**
7. VERIFY Present\_Value = (any value other than V<sub>2</sub>)

[Change **Clause 7.3.2.23.10.3.8**, p. 119]

[Reviewer Note: Added 'Notes to Tester' for clarity.]

#### 7.3.2.23.10.3.8 Revision 4 Event Priority Test

~~Dependencies: ReadProperty Service Execution Tests, 9.18; TimeSynchronization Service Execution Tests, 9.30.~~

~~BACnet Reference Clause: 12.24.8.~~

Purpose: To verify that a BACnetSpecialEvent of a higher priority takes precedence over one of lower priority when both are active at the same time, and that it relinquishes to the lower priority.

Configuration Requirements: The IUT shall be configured with a Schedule object containing two or more BACnetSpecialEvents, all active on the same date, with different eventPriority values (if possible, all 16 priority levels should be represented), and with overlapping BACnetTimeValue entries distributed thus: the entry with the lowest priority shall have the earliest time-value pair (D<sub>1</sub>) with a non-NULL value, and the last time-value pair (D<sub>N</sub>) with a NULL value; the next higher priority shall have a time-value pair D<sub>2</sub> occurring after D<sub>1</sub> with a different non-NULL value, and a time-value pair D<sub>N-1</sub> with a NULL value and occurring before D<sub>N</sub>; and so on. The result is that the time-value pairs shall be ordered chronologically thus: D<sub>1</sub>, D<sub>2</sub>, D<sub>3</sub>, ..., D<sub>N-1</sub>, D<sub>N</sub>. An example of such a configuration testing five priority levels is shown in Table 7-11.

**Table 7-11.** Example of event and value prioritization

Event Priority:	Time:								
	D <sub>1</sub>	D <sub>2</sub>	D <sub>3</sub>	D <sub>4</sub>	D <sub>5</sub>	D <sub>6</sub>	D <sub>7</sub>	D <sub>8</sub>	D <sub>9</sub>
1	-	-	-	-	V <sub>5</sub>	NULL	-	-	-
2	-	-	-	V <sub>4</sub>	-	-	NULL	-	-
3	-	-	V <sub>3</sub>	-	-	-	-	NULL	-
4	-	V <sub>2</sub>	-	-	-	-	-	-	NULL
5	V <sub>1</sub>	-	-	-	-	-	-	-	-
Present_Value:	V <sub>1</sub>	V <sub>2</sub>	V <sub>3</sub>	V <sub>4</sub>	V <sub>5</sub>	V <sub>4</sub>	V <sub>3</sub>	V <sub>2</sub>	V <sub>1</sub>

Note: Each event priority in the table above represents 1 BACnetSpecialEvent. The BACnetSpecialEvent should contain the time value pairs listed in the table (D<sub>x</sub>, V<sub>x</sub>). There should be only 1 BACnetSpecialEvent per priority for this test.

*Notes to Tester: Any WriteProperty request generated by the IUT may have a Priority parameter. If included, it shall be in the range 1-16, excluding 6. The Priority parameter for WriteProperty-Request may be left out if the target property is a standard property of a standard object for which commandability is not an option.*

Test Steps:

1. REPEAT D = (the times in the configured time-value pairs with non-NULL values) DO {  
 (TRANSMIT TimeSynchronization-Request, 'Time' = D) |  
 (TRANSMIT UTCTimeSynchronization-Request, 'Time' = D converted to UTC) |  
 MAKE (the local date and time = D)  
 WAIT Schedule Evaluation Fail Time  
 VERIFY Present\_Value = (the value corresponding to the time D)  
 }
2. REPEAT D = (the times in the configured time-value pairs with NULL values,  
 except the final DN) DO {  
 (TRANSMIT TimeSynchronization-Request, 'Time' = D) |  
 (TRANSMIT UTCTimeSynchronization-Request, 'Time' = D converted to UTC) |  
 MAKE (the local date and time = D)  
 WAIT Schedule Evaluation Fail Time  
 VERIFY Present\_Value = (the non-NULL value corresponding to the priority lower than that  
 associated with D)  
 }

### 7.3.2.23.11 Written Datatypes Tests

[Change **Clause 7.3.2.23.11.1**, p. 124]

[Reviewer Note: To ensure that datatype-specific testing is conducted even when there is no Schedule that can be made to reference another property. The effect should be observed in the Present\_Value property of the Schedule Object instead.]

#### 7.3.2.23.11.1 Internally Written Datatypes Test, non-NULL values

~~BACnet Reference Clauses: 12.24, 12.24.10~~

Purpose: This test verifies that the Schedule object within the IUT writes to properties in the same device for the non-NULL datatype being tested.

Test Concept: Two Date/Time, values, D1 and D2, are chosen by the TD based on the criteria in Table 7-17 such that D1 is sufficiently different from, and later than, the current time to cause a Schedule evaluation when the time is changed to D1, and such that setting the time to D2 (later than D1) from D1 will cause a Schedule evaluation that will cause it to write value V2. These values may be chosen based on the Schedule object's existing configuration, or the Schedule object, S, may be configured with such values.

Configuration Requirements: The IUT shall be configured with a Schedule object, S, such that the time periods defined in Table 7-17 can be configured with uniquely scheduled values. The Schedule object shall be configured with a List\_Of\_Object\_Property\_References, including at least one reference to a writable property within the device, *if possible. Step 4 and step 8 would REPEAT zero times if the referenced property is empty or not present. If the IUT cannot be configured to these requirements, then this test shall be omitted. Other properties in the Schedule object shall be consistent in both datatypes and values in a manner permitting this test to be executed.*

**Table 7-17. Criteria for Test Date and Times**

Date and Time:	Value:
D <sub>1</sub>	V <sub>1</sub>
D <sub>2</sub>	V <sub>2</sub> different from V <sub>1</sub> .

*Notes to Tester: In the context of this test definition, writable means that the Schedule object is capable of modifying the property. It does not necessarily indicate that the property is modifiable via BACnet services.*

#### Test Steps:

1. (TRANSMIT TimeSynchronization-Request, 'Time' = D<sub>1</sub>)  
     | (TRANSMIT UTCTimeSynchronization-Request, 'Time'=D<sub>1</sub>)  
     | MAKE (the local date and time = D<sub>1</sub>)
2. WAIT (**Schedule Evaluation Fail Time**)
3. VERIFY S, Present\_Value = V<sub>1</sub>
4. REPEAT P = (writable property in List\_Of\_Property\_References)  
     VERIFY P = V<sub>1</sub>
5. (TRANSMIT TimeSynchronization-Request, 'Time' = D<sub>2</sub>)  
     | (TRANSMIT UTCTimeSynchronization-Request, 'Time'=D<sub>12</sub>)  
     | MAKE (the local date and time = D<sub>2</sub>)
6. WAIT(**Schedule Evaluation Fail Time**)
7. VERIFY S, Present\_Value = V<sub>2</sub>
8. REPEAT P = (writable property in List\_Of\_Property\_References)  
     VERIFY P = V<sub>2</sub>

~~Notes to Tester: In the context of this test definition, writable means that the Schedule object is capable of modifying the property. It does not necessarily indicate that the property is modifiable via BACnet services.~~

#### 7.3.2.24 Logging Object Tests

[Change **Clause 7.3.2.24.3**, p. 131]

[Reviewer Note: Errata change to correct the Test Concept to reference correct property.]

##### 7.3.2.24.3 Stop\_Time Test

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

~~BACnet Reference Clause: 12.25.7.~~

Purpose: To verify that logging is disabled at the time specified by Stop\_Time.

Test Concept: The logging object is configured to acquire data by each means available to the implementation. The test is begun at some time prior to the time specified in *Stop\_Time*~~Start\_Time~~ and collection of records is confirmed. Non-collection of records after the time specified by Stop\_Time is then confirmed.

Configuration Requirements: Stop\_Time shall be configured with a date and time such that steps 1 through 9 will be concluded before that time. Start\_Time, if present shall be configured with date and time preceding the initiation of the test. Stop\_When\_Full, if configurable, shall be set to FALSE.

*Notes to Tester: For each MAKE (IUT collect another record), perform the following actions:*

```

IF (Event Log Object) THEN
    MAKE (Event Log Object collect another record)
ELSE
    IF (COV subscription in use) THEN
        MAKE (monitored value change sufficient to generate another record)
    ELSE IF (interval or period logging is in use) THEN
        WAIT (Log_Interval)
    ELSE
        MAKE (Trend Log or Trend Log Multiple Object collect another record)
    
```

#### Test Steps:

1. WRITE Enable = FALSE
2. WAIT **Internal Processing Fail Time**
3. WRITE Record\_Count = 0
4. WRITE Enable = TRUE
5. READ X = Total\_Record\_Count
6. WAIT **Internal Processing Fail Time**
7. MAKE (IUT collect another record)
8. WAIT (**Notification Fail Time + Internal Processing Fail Time**)
9. VERIFY Total\_Record\_Count > X
10. WHILE (IUT clock is earlier than Stop\_Time) DO {}
11. WAIT (**Notification Fail Time + Internal Processing Fail Time**)
12. READ X = Total\_Record\_Count
13. MAKE (IUT collect another record)
14. WAIT (**Notification Fail Time + Internal Processing Fail Time**)
15. VERIFY Total\_Record\_Count = X

~~Notes to Tester: For each MAKE (IUT collect another record), perform the following actions:~~

~~IF (Event Log Object) THEN~~  
~~MAKE (Event Log Object collect another record)~~  
~~ELSE~~  
~~IF (COV subscription in use) THEN~~  
~~MAKE (monitored value change sufficient to generate another record)~~  
~~ELSE IF (interval or period logging is in use) THEN~~  
~~WAIT (Log\_Interval)~~  
~~ELSE~~  
~~MAKE (Trend Log or Trend Log Multiple Object collect another record)~~

[Change **Clause 7.3.2.24.4**, p. 132]

[Reviewer Note: The Configuration Requirements are enhanced, and a Notes to Tester is added.]

#### **7.3.2.24.4 Log\_Interval Test**

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

~~Purpose: To verify that the logging period is controlled by Log\_Interval.~~

Test Concept: The logging object is configured to acquire data by polling. Polling is done at two different intervals, defined by Log\_Interval, with about 10 records acquired at each rate. The timestamps of the records are inspected to verify the polling rate.

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. Stop\_When\_Full, if configurable, shall be set to FALSE. Enable shall be set to TRUE. *Logging\_Type is not equal to TRIGGERED*. Non-zero values shall be chosen for Log\_Interval in accordance with the range and resolution specified by the manufacturer for this property.

*Notes to Tester: The step 1 write of Logging\_Interval to a non-zero value will make a change in Logging\_Type from COV to POLLED, if Logging\_Type was initially COV.*

Test Steps:

1. WRITE Log\_Interval = (some non-zero value)
2. WRITE Record\_Count = 0
3. WAIT (**Internal Processing Fail Time + 10\* Log\_Interval hundredths-seconds**)
4. VERIFY (Log\_Buffer record timestamp intervals, on average, are as written in step 1)
5. WRITE Log\_Interval = (a non-zero value different from the one written in step 1)
6. WRITE Record\_Count = 0



7. WAIT (**Internal Processing Fail Time** + 10\* Log\_Interval hundredths-seconds)
8. VERIFY (Log\_Buffer record timestamp intervals, on average, are as written in step 5)

### 7.3.2.24.6 Stop\_When\_Full Tests

[Change **Clause 7.3.2.24.6.1**, p. 134]

[Reviewer Note: This test was revised for negative response.]

#### 7.3.2.24.6.1 Stop\_When\_Full TRUE Test

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

Purpose: To verify that Stop\_When\_Full set to TRUE properly indicates that the logging object ceases collecting data when its Log\_Buffer acquires Buffer\_Size data items.

Test Concept: The logging object is configured to acquire data by whatever means. Data is collected until more than Buffer\_Size records have been collected and Enable is verified to be FALSE. *Attempt to write TRUE to Enable and verify that the IUT does not accept it due to Log\_Buffer being full.*

Configuration Requirements: *The IUT shall be configured with Object1 where 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. Stop\_When\_Full, if configurable, shall be set to TRUE. Enable shall be set to FALSE.*

Test Steps:

1. WRITE Record\_Count = 0
2. WRITE Enable = TRUE
3. WHILE (Record\_Count < Buffer\_Size) DO {}
4. WAIT **Internal Processing Fail Time**
5. VERIFY Enable = FALSE
6. TRANSMIT WriteProperty-Request,  
    'Object Identifier' = Object1,  
    'Property Identifier' = Enable,  
    'Property Value' = TRUE
7. RECEIVE BACnet-Error-PDU,  
    'Error Class' = OBJECT,  
    'Error Code' = LOG\_BUFFER\_FULL.
8. VERIFY Enable = FALSE

[Change **Clause 7.3.2.24.13**, p. 139]

[Reviewer Note: Modified test with a completely different, less prescriptive approach.]

### 7.3.2.24.13 Log-Status Test

~~Dependencies: ReadRange Service Execution Tests, 9.21; WriteProperty Service Execution Tests, 9.22.~~

~~BACnet Reference Clause: 12.25.14~~

Purpose: To verify proper logging of log-disabled and buffer-purged events.

Test Concept: The buffer is cleared. Then the Enable property is changed and it is verified that the Record\_Count property is changed and it is verified that the status entry is made correctly in the Log\_Buffer. The Record\_Count is also set to zero while the Enable property is FALSE and it is verified that the buffer-purged event is recorded into the Log\_Buffer.

Test Configuration: The logging object is configured to acquire data by whatever means available. Configure the logging such that the entire test may be run without the trend buffer overflowing.

*Notes to Tester: When the IUT's Protocol\_Revision < 7, the length of BACnetLogStatus shall be 2; otherwise, it shall be 3.*

Test Steps:

1. ~~WRITE Enable = FALSE~~
2. ~~WRITE Record\_Count = 0~~
3. ~~VERIFY Record\_Count = 1~~
4. ~~TRANSMIT ReadRange~~  
~~'Object Identifier' = 01,~~  
~~'Property Identifier' = Log\_Buffer,~~  
~~'Reference Index' = 1,~~  
~~'Count' = 1~~
5. ~~RECEIVE ReadRangeAck~~  
~~'Object Identifier' = 01,~~  
~~'Property Identifier' = Log\_Buffer,~~  
~~'Result Flags' = (True, True, False),~~  
~~'Item Count' = 1~~  
~~'Item Data' = ((a buffer purged record))~~
6. ~~WRITE Enable = TRUE~~
7. ~~WRITE Enable = FALSE~~
8. ~~TRANSMIT ReadRange~~  
~~'Object Identifier' = 01,~~  
~~'Property Identifier' = Log\_Buffer,~~  
~~'Reference Index' = 1,~~  
~~'Count' = 2~~
9. ~~RECEIVE ReadRangeAck~~  
~~'Object Identifier' = 01,~~  
~~'Property Identifier' = Log\_Buffer,~~  
~~'Result Flags' = (True, False, False),~~  
~~'Item Count' = 2~~  
~~'Item Data' = ((a buffer purged record), (a log-enable record))~~
10. ~~TRANSMIT ReadRange~~  
~~'Object Identifier' = 01,~~  
~~'Property Identifier' = Log\_Buffer,~~  
~~'Reference Time' = (2154 12 31, 23:59:59.99),~~  
~~'Count' = 1~~
11. ~~RECEIVE ReadRangeAck~~  
~~'Object Identifier' = 01,~~  
~~'Property Identifier' = Log\_Buffer,~~  
~~'Result Flags' = (False, True, False),~~  
~~'Item Count' = 1~~  
~~'Item Data' = ((a log-disable record))~~
1. ~~WRITE Enable = FALSE~~
2. ~~WRITE Record\_Count = 0~~
3. ~~VERIFY (Log\_Buffer contains 1 entries, and it is the buffer-purged event)~~
4. ~~WRITE Enable = TRUE~~
5. ~~WRITE Enable = FALSE~~
6. ~~VERIFY (Record\_Count => 3 and the first entry is the buffer-purged event, the second entry is the log-enable TRUE event and the last entry is the log-enable FALSE event)~~

*Notes to Tester: When the IUT's Protocol\_Revision < 7, the length of BACnetLogStatus shall be 2; otherwise, it shall be 3.*

[Change **Clause 7.3.2.24.14**, p. 140]

[Reviewer Modified test with a completely different, less prescriptive approach.]

### 7.3.2.24.14 Time\_Change Test

Purpose: To verify proper logging of time-change events in the log buffer

Test Concept: Change the clock in the device and verify that a record is logged indicating the number of seconds that the clock changed by or indicating zero if unknown. This test shall be skipped if the device does not support the Local\_Time property in the device object or there is no way to change the time in the device.

Configuration Requirements: The log object is configured to acquire data by whatever means available. ~~The Log\_Buffer should be cleared such that the Record\_Count is 0.~~ Configure the logging such that the entire test may be run without the trend buffer overflowing.

Test Steps:

- ~~1. WRITE Enable = FALSE~~
- ~~2. WRITE Record\_Count = 0~~
- ~~3. READ currentTime = (Device Object of device that contains the log object), Local\_Time~~
- ~~4. WRITE Enable = TRUE~~
- ~~5. MAKE (the time change on the device by deltaTime where deltaTime >= 1 hour)~~
- ~~6. WRITE Enable = FALSE~~
- ~~7. READ N = Record\_Count~~
- ~~8. REPEAT X = (N down through 1) DO {~~
  - ~~TRANSMIT ReadRange~~
  - ~~'Object Identifier' = O1,~~
  - ~~'Property Identifier' = Log\_Buffer,~~
  - ~~'Reference Index' = X,~~
  - ~~'Count' = 1~~
  - ~~RECEIVE ReadRangeAck~~
  - ~~'Object Identifier' = O1,~~
  - ~~'Property Identifier' = Log\_Buffer,~~
  - ~~'Result Flags' = (False, True, False),~~
  - ~~'Item Count' = 1,~~
  - ~~'Item Data' = ( (a record. If the record is a time change record, save the timestamp into TS and the time change value into TC) )~~- ~~— }~~
- ~~9. CHECK ( TC == deltaTime )~~
- ~~10. CHECK ( TS == currentTime + deltaTime )~~
  1. WRITE Enable = FALSE
  2. WRITE Record\_Count = 0
  3. VERIFY (Log\_Buffer contains 1 entry, and it is the buffer-purged event)
  4. TRANSMIT ReadProperty-Request,
    - 'Object Identifier' = (device that contains log object)
    - 'Property Identifier' = Local\_Time
  5. RECEIVE ReadProperty-Ack,
    - 'Object Identifier' = (device that contains log object)
    - 'Property Identifier' = Local\_Time
    - 'Property Value' = (currentTime)
  6. WRITE Enable = TRUE
  7. MAKE (the time change on the device by a reasonable amount (deltaTime); change by one hour or more)
  8. WRITE Enable = FALSE
  9. VERIFY Record\_Count => 4
  10. CHECK (Log\_Buffer contains a log-status entry of time-change)
  11. VERIFY ( time-change value == deltaTime )
  12. VERIFY TimeStamp on the time-change entry == ( currentTime + deltaTime )

[Change Clause 7.3.2.24.15, p. 141]

[Reviewer Note: Modified test with a completely different, less prescriptive approach. The Test Concept is simplified. The Configuration Requirements are enhanced.]

### 7.3.2.24.15 COV-Sampling Verification Test

Purpose: To verify logged samples are based on COV rather than by interval.

Test Concept: The trend log is configured to log based on COV increment. Logging is enabled. After a period of time the buffer is checked to verify the data in the buffer is based on the COV values and not on the set interval.

~~Configuration Requirements: The IUT shall be configured such that the monitored object has a COV\_Increment property that is set to a value other than 0.0, the Client\_COV\_Increment is set to a value other than 0.0 or NULL, or the monitored property is not of datatype REAL.~~

~~Test Steps:~~

- ~~1. WRITE Enable = FALSE~~
- ~~2. WRITE Record\_Count = 0~~
- ~~3. WRITE Interval = 0~~
- ~~4. WRITE Enable = TRUE~~
- ~~5. WAIT ( 10 seconds )~~
- ~~6. MAKE (monitored property change its value)~~
- ~~7. WAIT ( 60 seconds )~~
- ~~8. MAKE (monitored property change its value)~~
- ~~9. WAIT ( 90 seconds )~~
- ~~10. MAKE (monitored property change its value)~~
- ~~11. WAIT ( 40 seconds )~~
- ~~12. MAKE (monitored property change its value)~~
- ~~13. WAIT Notification Fail Time~~
- ~~14. WRITE Enable = FALSE~~
- ~~15. READ N = RecordCount~~
- ~~16. REPEAT X = (1 through 4) {~~
  - ~~TRANSMIT ReadRange~~
  - ~~‘Object Identifier’ = O1,~~
  - ~~‘Property Identifier’ = Log\_Buffer,~~
  - ~~‘Reference Index’ = N-5+X,~~
  - ~~‘Count’ = 1~~
  - ~~RECEIVE ReadRangeAck~~
  - ~~‘Object Identifier’ = O1,~~
  - ~~‘Property Identifier’ = Log\_Buffer,~~
  - ~~‘Result Flags’ = (False, False, False),~~
  - ~~‘Item Count’ = 1~~
  - ~~‘Item Data’ = ( (one data record storing the timestamp in TS[X]))~~~~}~~
- ~~17. CHECK( TS[2] - TS[1] = 60 seconds )~~
- ~~18. CHECK( TS[3] - TS[2] = 90 seconds )~~
- ~~19. CHECK( TS[4] - TS[3] = 40 seconds )~~

*Configuration Requirements: The IUT shall be configured such that the monitored object has COV configured, or the Client\_COV\_Increment shall be configured, or it is not monitoring a REAL property. The Logging\_Type shall not have a value of TRIGGERED.*

*Test Steps:*

1. WRITE Enable = FALSE
2. WRITE Record\_Count = 0
3. WRITE Log\_Interval = 0
4. WRITE Enable = TRUE
5. MAKE (monitored property change its value)
6. WAIT ( 60 seconds )

7. *MAKE (monitored property change its value)*
8. *WAIT ( 90 seconds )*
9. *MAKE (monitored property change its value)*
10. *WAIT ( 40 seconds )*
11. *CHECK ( Log\_Buffer contains 3 or 4 data entries, and time between each sample is not equal )*

[Change **Clause 7.3.2.24.19**, p. 144]

[Reviewer Note: Modified test to check Log\_Interval shall be zero when Logging\_Type property has either of the values COV or TRIGGERED. Updated Configuration Requirements.]

### 7.3.2.24.19 Trigger Verification Test

~~Dependencies: ReadRange-Service-Execution, 9.21;~~

~~BACnet Reference Clause: 12.25.27, 12.30.12~~

Purpose: To verify logged samples are based on the triggered Logging\_Type.

Test Concept: The log,  $O_I$  is configured to log based on TRIGGERED. Logging is enabled. After a period of time the buffer is checked to verify the data in the buffer is based on triggered values.

Configuration Requirements: ~~The IUT shall be configured such that the monitored object's Logging\_Type is set to TRIGGERED.~~ *The object being tested shall be configured with Logging\_Type set to TRIGGERED.*

Test Steps:

1. *VERIFY Logging\_Type = TRIGGERED*
2. *VERIFY Log\_Interval = 0*
34. *WRITE Enable = FALSE*
42. *WRITE Record\_Count = 0* ~~results in a buffer purged record~~
53. *WRITE Enable = TRUE* ~~results in a logging enable record~~
64. *WAIT (10 seconds)*
75. *WRITE Trigger = TRUE*
86. *WAIT (20 seconds)*
97. *WRITE Trigger = TRUE*
108. *WAIT (40 seconds)*
119. *WRITE Trigger = TRUE*
124. *WAIT (30 seconds)*
134. *WRITE Enable = FALSE* ~~results in a logging disabled record~~
12. ~~VERIFY Record\_Count = 6~~
14. *READ N = Record\_Count*
1543. *REPEAT X = (1 through 34)*
  - TRANSMIT ReadRange-Request*
    - 'Object Identifier' =  $O_I$ ,
    - 'Property Identifier' = *Log\_Buffer*,
    - 'Reference Index' =  $N-4+X$ ,
    - 'Count' = 1
  - RECEIVE ReadRangeAckReadRange-ACK*
    - 'Object Identifier' =  $O_I$ ,
    - 'Property Identifier' = *Log\_Buffer*,
    - 'Result Flags' = (*?False, ?False, False*),
    - 'Item Count' = 1,
    - 'Item Data' =  $\leftarrow$ (one data record storing the timestamp in  $TS[X]$ ) $\rightarrow$
16. *TRANSMIT ReadRange-Request*
  - 'Object Identifier' =  $O_I$ ,
  - 'Property Identifier' = *Log\_Buffer*,

'Reference Index' = N,  
 'Count' = 1  
 RECEIVE ReadRange-ACK  
 'Object Identifier' = 01,  
 'Property Identifier' = Log\_Buffer,  
 'Result Flags' = (False, True, False),  
 'Item Count' = 1,  
 'Item Data' = (one data record storing the timestamp in TS[4])

14. CHECK( TS[3] - TS[2]  $\sim$  10 seconds )
15. CHECK( TS[4] - TS[3]  $\sim$  20 seconds )
16. CHECK( TS[5] - TS[4]  $\sim$  40 seconds )
17. CHECK( TS[6] - TS[5]  $\sim$  30 seconds )
17. CHECK( TS[2] - TS[1]  $\sim$  20 seconds )
18. CHECK( TS[3] - TS[2]  $\sim$  40 seconds )
19. CHECK( TS[4] - TS[3]  $\sim$  30 seconds )

[Change and renumber **Clause 9.21.1.12**, p. 409]

[Reviewer Note: Moved modified test from the Application Service Execution Tests section to the Object Support Tests section.]

#### **9.21.1.12 7.3.2.24.X1 Status/Failure logging**

Purpose: To verify that a failure is logged when an error is encountered in an attempt to read a data value from the monitored object. ~~If the error is conveyed by an error response from a remote device, verify that the Error Class and Error Code in the response is logged.~~

Test Concept: ~~Make the monitored object fail and respond with an error by setting the Configure Log\_DeviceObjectProperty of the logging object with an unknown object such that collection of records fails. to an invalid device or object.~~ Wait until the IUT attempts to read a sample for the Log\_Buffer. Then check the Log\_Buffer to verify that there is a failure entry that consists of the ErrorClass and ErrorCode of the error. Repeat with Log\_DeviceObjectProperty referencing an object in a device that does not exist.

Configuration Requirements: Configure the logging object so that collection of records will fail (such as by referencing a non-existent object).

Test Steps:

- ~~1. WRITE (Invalid object into the Log\_DeviceObjectProperty of the log object)~~
1. MAKE (Log\_DeviceObjectProperty reference a non-existent object in the local device or in an existing remote device)
2. WAIT (until IUT attempts to read a sample for the Log\_Buffer)
3. ~~VERIFY~~ CHECK(Log\_Buffer contains a failure entry of with an error class/error code of OBJECT/UNKNOWN\_OBJECT~~unknown object~~)
4. IF the IUT supports logging remote values THEN {  
     MAKE (Log\_DeviceObjectProperty reference an object in a non-existing device )  
     WAIT (until IUT attempts to read a sample for the Log\_Buffer)  
     CHECK (Log\_Buffer contains a failure entry with an error class/code  
         COMMUNICATION/UNKNOWN\_DEVICE)

[Add new **Clause 7.3.2.24.X8**, p. 144]

#### **7.3.2.24.X8 Clock-Aligned Logging**

Purpose: To verify that logged trend records have timestamps aligned to that interval, when Align\_Intervals is TRUE and Log\_Interval is a factor of (divides without remainder) a day.

Test Concept: For this test, select two evenly divisible factors. Write each to Log\_Interval in the test. Trend records are logged and checked that those are aligned to the Log\_Interval. This is done twice to ensure that different interval frequency behavior is verified. This test does not employ Log\_Interval values which are not one of the evenly divisible factors.

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 in order that it occurs after the end of the test. Stop\_When\_Full, if configurable, shall be set to FALSE. Enable is initially FALSE. Interval\_Offset is set to zero. Align\_Intervals is set to TRUE. Triggering of non periodic log records should not occur during this test. Logging\_Mode is POLLED. X1 and X2 are each a value which the IUT supports for which the standard mandates the alignment behavior.

Notes to Tester: The values for Log\_Interval which require alignment are those for which the standard mandates the alignment behavior, where 8,640,000 modulo Log\_Interval is zero.

Test Steps:

1. CHECK (Log\_Buffer contains 1 entry, and it is the buffer-purged event)
2. WRITE Log\_Interval = X1
3. WRITE Enable = TRUE
4. MAKE (logging object collect at least 2 records)
5. WRITE Enable = FALSE
6. CHECK (Log\_Buffer contains at least 5 entries, and at least two data records)
7. CHECK (that the timestamp of each data record, since the Log\_Interval was written, is a multiple of X1)
8. WRITE Log\_Interval = (X2, any value which requires alignment behavior, that was not already chosen)
9. WRITE Enable = TRUE
10. MAKE (logging object collect at least 2 more records)
11. WRITE Enable = FALSE
12. CHECK (Log\_Buffer has collected two or more additional data records and two or more log-status records)
13. CHECK (that the timestamp are multiples of X2 for all data records collected, since the write with X2)

[Add new **Clause 7.3.2.24.X9**, p. 144]

### **7.3.2.24.X9 Logging Interval\_Offset**

Purpose: To verify that timestamps abide by the Interval\_Offset.

Test Concept: Log\_Interval is set to a value which the IUT supports which is a factor of (divides without remainder) a day and which is greater than 3 seconds.

Interval\_Offset is first set to a non-zero value less than Log\_Interval. After logging some records, their timestamps are checked. The logging is stopped. Interval\_Offset is set to a value which the IUT supports greater than Log\_Interval, logging is re-enabled, and the timestamps again are checked.

Configuration Requirements: Align\_Intervals is set to TRUE. The Log\_DeviceObjectProperty property in a Trend Log or in a Trend Log Multiple, is configured to the property or properties monitored. 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 in order that it occur after the end of the test. Stop\_When\_Full, if configurable, shall be set to FALSE. Enable is initially FALSE. Align\_Intervals is set to TRUE. Triggering of non periodic log records should not occur during this test. If the Interval\_Offset cannot be set to a value which the IUT supports greater than Log\_Interval, then steps 11 through the end of this test are skipped. Logging\_Mode is POLLED. An evenly divisible value is a value for which the standard mandates the alignment behavior.

Notes to tester: Interval\_Offset in logging objects, and Log\_Interval are each an Unsigned number of hundredths of seconds. Excellent choices are 400, 500, 600, 1000, or 1200. When Interval\_Offset is larger than Log\_Interval, then Interval\_Offset modulo Log\_Interval, is smaller than Log\_Interval.

Test Steps:

1. WRITE Record\_Count = 0
2. CHECK (Log\_Buffer contains 1 entry, and it is the buffer-purged event)

3. WRITE Log\_Interval = (any evenly divisible value greater than 3 seconds)
4. WRITE Interval\_Offset = (any value, between 2 seconds and Log\_Interval - 1 seconds)
5. WRITE Enable = TRUE
6. MAKE (logging object collect at least 2 records)
7. WRITE Enable = FALSE
8. CHECK (Log\_Buffer contains two or more data records and at least three log-status)
9. CHECK (the timestamp for the data records have a fixed offset, determined by Log\_Interval and Interval\_Offset)
10. WRITE Interval\_Offset = (any value greater than Log\_Interval)
11. WRITE Enable = TRUE
12. MAKE (logging object collect at least 2 records)
13. WRITE Enable = FALSE
14. CHECK (Log\_Buffer has collected two or more additional data records and two or more log-status entries)
15. CHECK (the timestamp for data records collected since the Interval\_Offset was last written, have Log\_Interval between records, at a fixed offset of Interval\_Offset modulo Log\_Interval)

[Add new **Clause 7.3.2.24.X10**, p. 144]

#### **7.3.2.24.X10 Buffer\_Size Write Test**

Purpose: To verify the content of the log buffer after a write to the Buffer\_Size property.

Test Concept: The logging object, O1, is configured to acquire data by whatever means. Logging is disabled and Buffer\_Size set to a different valid value, V1. The content of the Log\_Buffer is read to confirm a single entry that is a buffer-purged event.

Configuration Requirements: The logging object, O1, is configured to acquire data by whatever means. If a write to the Buffer\_Size does not delete all records in the log, this test shall be skipped.

Test Steps:

1. WRITE Enable = FALSE
2. WRITE Buffer\_Size = V1
3. **WAIT Internal Processing Fail Time**
4. CHECK (Log\_Buffer contains one entry, and it is a buffer-purged event)

#### **7.3.2.25 Event Log Tests**

[Change **Clause 7.3.2.25.1**, p. 144]

[Reviewer Note: Fixed the Result Flags value in step 12]

##### **7.3.2.25.1 Internal Logging of Notifications**

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

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

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

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

Test Steps:



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

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

[Change **Clause 7.3.2.25.2**, p. 144]

[Reviewer Note: Fixed Result Flags in step 8. Add delay after Step 1 and renumbered subsequent test steps.]

### 7.3.2.25.2 Remote Logging of Notifications

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

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

Configuration Requirements: LO1 is an Event Log object in IUT which logs the event types which are sent. Stop\_When\_Full in LO1 shall be FALSE or absent.

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

Test Steps:

1. WRITE Enable = TRUE
2. WAIT Internal Processing Fail Time
32. 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' = (optional, any valid message text),
  - 'Notify Type' = ALARM | EVENT,
  - 'AckRequired' = TRUE | FALSE,
  - 'From State' = (state S1, any valid state for this event type),
  - 'To State' = (state S2, any valid state for this event type that can follow S1),
  - 'Event Values' = (any values appropriate to the event type)
43. RECEIVE BACnet-SimpleACK-PDU
54. 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' = (optional, any valid message text),
  - 'Notify Type' = ALARM | EVENT,
  - 'AckRequired' = TRUE | FALSE,
  - 'From State' = (state S3, any valid state for this event type),
  - 'To State' = (state S4, any valid state for this event type that can follow S3),
  - 'Event Values' = (any values appropriate to the event type)
65. RECEIVE BACnet-SimpleACK-PDU
76. READ RC = LO1, Record\_Count
87. TRANSMIT ReadRange-Request,
  - 'Object Identifier' = LO1,
  - 'Property Identifier' = Log\_Buffer,
  - 'Reference Index' = RC,
  - 'Count' = -2
98. RECEIVE ReadRange-ACK,
  - 'Object Identifier' = LO1,
  - 'Property Identifier' = Log\_Buffer,
  - 'Result Flags' = {FALSE?, ?, 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)

109. CHECK (that the events were logged in the order in which they were received)

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

[Change **Clause 7.3.2.25.3**, p. 145]

[Reviewer Note: Fixed the Result Flags in step 12. Moved step 2 to step 9. Removed unnecessary step 4.]

### 7.3.2.25.3 Internal Logging of ACK\_NOTIFICATIONs

Purpose: To verify the IUT correctly collects and represents an ACK\_NOTIFICATION which it initiates.

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

Configuration Requirements: O1 is an event initiating object in the IUT, which is configured to send event notifications to TD. LO1 is an Event Log object in IUT which logs ACK\_NOTIFICATIONs.

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

Test Steps:

1. WRITE Enable = TRUE
- ~~2. READ RC = LO1, Record\_Count~~
23. MAKE (the IUT generate a notification)
- ~~4. WAIT D1~~
35. 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' = (optional, any valid message text),
  - 'Notify Type' = ALARM | EVENT,
  - 'AckRequired' = TRUE | FALSE,
  - 'From State' = (S1, any valid state for this event type),
  - 'To State' = (S2, any valid state for this event type),
  - 'Event Values' = (any values appropriate to the event type)
46. TRANSMIT BACnet-SimpleACK-PDU
57. 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)
68. RECEIVE BACnet-SimpleACK-PDU
79. 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' = (optional, any valid message text),  
 'Notify Type' = ACK\_NOTIFICATION,  
 'From State' = S1

840. TRANSMIT BACnet-SimpleACK-PDU

9. *READ RC = LO1, Record\_Count*

1041. TRANSMIT ReadRange-Request,  
 'Object Identifier' = LO1,  
 'Property Identifier' = Log\_Buffer,  
 'Reference Index' = RC,  
 'Count' = -1

1142. RECEIVE ReadRange-ACK,  
 'Object Identifier' = LO1,  
 'Property Identifier' = Log\_Buffer,  
 'Result Flags' = {FALSE?, ?, FALSETRUE},  
 'Item Count' = 1,  
 'Item Data' = (logged data that matches the information received in step 74,  
 except that Process\_Identifier can be any value and is not  
 required to match)

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

[Change **Clause 7.3.2.25.4**, p. 148]

[Reviewer Note: Fixed Result Flags in step 6. Added delay after Step 1.]

### 7.3.2.25.4 Remote Logging of ACK\_NOTIFICATIONs

Purpose: To verify that the IUT correctly collects and represents ACK\_NOTIFICATIONs which it receives.

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

Configuration Requirements: LO1 is an Event Log object in IUT which logs ACK\_NOTIFICATIONs. Stop\_When\_Full in LO1 shall be FALSE or absent.

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

Test Steps:

1. WRITE Enable = TRUE
2. *WAIT Internal Processing Fail Time*
32. TRANSMIT ConfirmedEventNotification-Request,  
 'Process Identifier' = (any valid process identifier),  
 'Initiating Device Identifier' = ~~IUT~~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' = (optional, any valid message text),  
 'Notify Type' = ACK\_NOTIFICATION,  
 'From State' = (state S1, any valid state for this event type)
43. RECEIVE BACnet-SimpleACK-PDU
54. READ RC = LO1, Record\_Count
65. TRANSMIT ReadRange-Request,  
 'Object Identifier' = LO1,

'Property Identifier' = Log\_Buffer,  
 'Reference Index' = RC,  
 'Count' = -1

76. RECEIVE ReadRange-ACK,  
 'Object Identifier' = LO1,  
 'Property Identifier' = Log\_Buffer,  
 'Result Flags' = {~~FALSE~~?, ?, ~~FALSE~~TRUE},  
 'Item Count' = 1,  
 'Item Data' = (logged data that matches the information received in step 2,  
 except that Process\_Identifier can be any value and is not required to match)

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

### 7.3.2.30 Notification Forwarder Object Tests

[Change **Clause 7.3.2.30.6**, p. 158]

[Reviewer Note: Fixed Status\_Flags expected values.]

#### 7.3.2.30.6 Out\_Of\_Service Property Test

Purpose: This test case verifies that event forwarding is not done while Out\_Of\_Service is TRUE.

Test Concept: Set up both Recipient\_List and Subscribed\_Recipient recipient entries with no filters specified and then send event notifications to the Notification Forwarder while the value of the Out\_Of\_Service property is TRUE. Subscribed\_Recipients are configured as part of base setup 2 for Notification Forwarder object tests. Verify that forwarding of the event notifications is not performed.

If the Out\_Of\_Service property of the object under test is not writable, and if the value of the property cannot be changed by other means, then this test shall be omitted.

Configuration Requirements: Base setup 2 for Notification Forwarder object tests with TR lifetime sufficient for this test.

Test Steps:

1. MAKE (Recipient\_List = { (all), --Valid Days  
 (all), --From Time, To Time  
 DEST\_OBJ\_ID2, --Recipient D2  
 DEST\_PROCESS\_ID, --Process Identifier  
 FALSE, --Issue Confirmed Notifications  
 {T, T, T} --Transitions  
 }) --One list element
2. MAKE (Out\_Of\_Service = TRUE)
3. VERIFY Out\_Of\_Service = TRUE
4. VERIFY Status\_Flags = (~~?FALSE~~, FALSE, ~~?FALSE~~, TRUE)
5. TRANSMIT SOURCE = DS, UnconfirmedEventNotification-Request,  
 'Process Identifier' = SRC\_PROCESS\_ID,  
 'Initiating Device Identifier' = SRC\_NOTIF\_DEV,  
 'Event Object Identifier' = SRC\_NOTIF\_OBJ,  
 'Time Stamp' = (any valid time stamp),  
 'Notification Class' = SRC\_NOTIF\_CLS,  
 'Priority' = (any valid priority),  
 'Event Type' = (any valid event type),  
 'Message Text' = (optional, any valid message text),  
 'Notify Type' = SRC\_NOTIF\_TYP,  
 'AckRequired' = (any valid value), -- absent if 'Notify Type' is ACK\_NOTIFICATION  
 'From State' = (any valid From\_State), -- absent if 'Notify Type' is ACK\_NOTIFICATION

- 'To State' = (any valid To\_State),  
 'Event Values' = (any valid event values) -- absent if 'Notify Type' is ACK\_NOTIFICATION
6. **WAIT Notification Fail Time**
  7. CHECK (the IUT did not transmit an event notification)
  8. MAKE (Out\_Of\_Service = FALSE)
  9. VERIFY Out\_Of\_Service = FALSE
  10. VERIFY Status\_Flags = (?FALSE, ?, ?FALSE, FALSE)
  11. TRANSMIT SOURCE = DS, UnconfirmedEventNotification-Request,
    - 'Process Identifier' = SRC\_PROCESS\_ID,
    - 'Initiating Device Identifier' = SRC\_NOTIF\_DEV,
    - 'Event Object Identifier' = SRC\_NOTIF\_OBJ,
    - 'Time Stamp' = (any valid time stamp),
    - 'Notification Class' = SRC\_NOTIF\_CLS,
    - 'Priority' = (any valid priority),
    - 'Event Type' = (any valid event type),
    - 'Message Text' = (optional, any valid message text),
    - 'Notify Type' = SRC\_NOTIF\_TYP,
    - 'AckRequired' = (any valid value), -- absent if 'Notify Type' is ACK\_NOTIFICATION
    - 'From State' = (any valid From\_State), -- absent if 'Notify Type' is ACK\_NOTIFICATION
    - 'To State' = (any valid To\_State),
    - 'Event Values' = (any valid event values) -- absent if 'Notify Type' is ACK\_NOTIFICATION
  12. **BEFORE Notification Fail Time** --The following can be in any order
    - RECEIVE DESTINATION = D1, UnconfirmedEventNotification-Request
    - RECEIVE DESTINATION = D2, UnconfirmedEventNotification-Request

[Add new **Clause 7.3.2.X37**, p. 185]

[Reviewer Note: Clause 7.3.2 (Object Specific Tests) contains no tests for the Accumulator object type.]

### 7.3.2.X37 Accumulator Object Tests

#### 7.3.2.X37.1 Present\_Value Remains In-Range Test

Purpose: To verify the correct wrapping operation of the Accumulator Present\_Value.

Test Concept: The IUT shall be configured with a Max\_Pres\_Value which is attainable, within reasonable testing time, after Present\_Value is preset to a value slightly less than that, then incremented. The Present\_Value shall remain in range from one to Max\_Pres\_Value, by wrapping back to 1 when it would exceed Max\_Pres\_Value.

Test Steps:

1. IF (Value\_Set is writable) THEN
  - WRITE Value\_Set = (a value slightly less than Max\_Pres\_Value)
- ELSE
  - MAKE (Present\_Value equal a value slightly less than Max\_Pres\_Value)
2. MAKE (the Accumulator increase its Present\_Value until it rolls over Max\_Pres\_Value)
3. CHECK (Present\_Value < Max\_Pres\_Value)

#### 7.3.2.X37.2 Prescale in Accumulator Test

Purpose: To verify the correct effect of Prescale on the increment of the Present\_Value in Accumulator.

Test Concept: The IUT shall be configured with a Prescale whose effect when incrementing Present\_Value is testable. Three readings of the Present\_Value are observed, then the math is checked to ensure that it increments at the rate expected given Prescale.

Configuration Requirements: If there is no Prescale property present in any Accumulator object, then this test shall be skipped.

Test Steps:

1. IF (Value\_Set is writable) THEN  
    WRITE Value\_Set = (any valid value  $V_1$ )  
ELSE  
    MAKE (Present\_Value equal any valid value  $V_1$ )
2. MAKE (the Accumulator increase its Present\_Value)
3. READ  $V_2$  = Present\_Value)
4. READ  $V_3$  = Present\_Value)
5. IF (the Accumulator is stopped) THEN  
    CHECK ( $V_3 = V_2 = ((\text{Prescale-multiplier}) * \text{pulse-count of signals generated by the measuring instrument}) / (\text{Prescale-moduloDivide}) + V_1$ )  
ELSE  
    CHECK ( $V_1 < V_2 < V_3$ )

### 7.3.2.X37.3 Logging\_Record in Accumulator Test

Purpose: To verify the correct values represented in Logging\_Record of Accumulator.

Test Concept: Two readings of the Logging\_Object acquiring the Logging\_Record are performed,  $Pv_{\text{prior}}$  being the value from the first, and Present\_Value matching what is observed in the second Logging\_Record. Then all fields are checked to ensure these match the values expected.

Configuration Requirements: The IUT shall be configured so that Logging\_Record capture is testable. If there is no Logging\_Record property present in any Accumulator object, then this test shall be skipped.

Test Steps:

1. MAKE (the Logging\_Object acquire the Logging\_Record)
2.  $Pv_{\text{prior}}$  = present-value parameter in the Logging\_Record
3. MAKE (the Logging\_Object acquire another Logging\_Record)
4. CHECK (Logging\_Record list of values are:  
    timestamp: the local date and time,  
    present-value: Present\_Value,  
    accumulated-value: Present\_Value -  $Pv_{\text{prior}}$ ,  
    accumulated-status: NORMAL)

### 7.3.2.X37.4 Logging\_Record in Accumulator RECOVERED Test

Purpose: To verify the correct values represented in Logging\_Record of Accumulator after one or more writes to Value\_Before\_Change or Value\_Set.

Test Concept: The effect of the Logging\_Object acquiring the Logging\_Record is checked to ensure that after one or more writes to Value\_Before\_Change or Value\_Set, it matches the values expected.

Configuration Requirements: The IUT shall be configured so that Logging\_Record capture is testable. If there is no Logging\_Record property present in any Accumulator object, or if neither Value\_Before\_Change nor Value\_Set is writable in an object which does have a Logging\_Record property, then this test shall be skipped.

Test Steps:

1. MAKE (the Logging\_Object acquire the Logging\_Record)
2.  $Pv_{\text{prior}}$  = present-value parameter in the Logging\_Record
3. WRITE (either Value\_Before\_Change or Value\_Set in the object that contains Logging\_Record)
4. MAKE (the Logging\_Object acquire another Logging\_Record)

5. CHECK (Logging\_Record list of values are:  
 timestamp: the local date and time,  
 present-value: Present\_Value,  
 accumulated-value:  $(\text{Present\_Value} - \text{Value\_Set}) + (\text{Value\_Before\_Change} - P_{v_{\text{prior}}})$ ,  
 accumulated-status: RECOVERED)

### 7.3.2.X37.5 Logging\_Record in Accumulator STARTING Test

Purpose: To verify the correct values represented in Logging\_Record of Accumulator when no data has been acquired since startup by the object identified by Logging\_Object.

Test Concept: The Logging\_Record is observed when no data has been acquired by the object identified by Logging\_Object, to ensure that it matches the values expected.

Configuration Requirements: The IUT shall be in a state when no data has been acquired since startup by the object identified by Logging\_Object. If there is no Logging\_Record property present in any Accumulator object, then this test shall be skipped.

Test Steps:

1. CHECK (Logging\_Record list of values are:  
 timestamp: unspecified,  
 present-value: Present\_Value,  
 accumulated-value: 0,  
 accumulated-status: STARTING)
2. MAKE (the Logging\_Object acquire the Logging\_Record)
3. CHECK (Logging\_Record list of values are:  
 timestamp: the local date and time,  
 present-value: Present\_Value,  
 accumulated-value: same as present-value,  
 accumulated-status: STARTING)

### 7.3.2.X37.6 Out\_Of\_Service Accumulator Test

Purpose: This test case verifies that Present\_Value, Pulse\_Rate, and the Reliability property are writable when Out\_Of\_Service is TRUE.

Test Concept: Select one instance of each appropriate object type and test it as described. Verify the interrelationship between the Out\_Of\_Service, Status\_Flags, and Reliability properties. If the Out\_Of\_Service property of the object under test is not writable, and the value of the property cannot be changed by other means, then this test shall be omitted. If the Reliability property is not supported, then step 5 shall be omitted.

Test Steps:

1. IF (Out\_Of\_Service is writable) THEN  
 WRITE Out\_Of\_Service = TRUE  
 ELSE  
 MAKE (Out\_Of\_Service TRUE)
2. VERIFY Out\_Of\_Service = TRUE
3. VERIFY Status\_Flags = (?, ?, ?, TRUE)
4. REPEAT X = (all values meeting the functional range requirements of 7.2.1) DO {  
 WRITE Present\_Value = X  
 VERIFY Present\_Value = X  
 }
5. IF (Reliability is present and writable) THEN  
 REPEAT X = (all values of the Reliability enumeration appropriate to the object type except  
 NO\_FAULT\_DETECTED) DO {  
 WRITE Reliability = X  
 VERIFY Reliability = X  
 VERIFY Status\_Flags = (?, TRUE, ?, TRUE)  
 WRITE Reliability = NO\_FAULT\_DETECTED



```

        VERIFY Reliability = NO_FAULT_DETECTED
        VERIFY Status_Flags = (?, FALSE, ?, TRUE)
    }
6. IF (the object has a Pulse_Rate property) THEN {
    REPEAT X = (all values meeting the functional range requirements of 7.2.1) DO {
        WRITE Pulse_Rate = X
        VERIFY Pulse_Rate = X
    }
}
7. IF (Out_Of_Service is writable) THEN
    WRITE Out_Of_Service = FALSE
ELSE
    MAKE (Out_Of_Service FALSE)
8. VERIFY Out_Of_Service = FALSE
9. VERIFY Status_Flags = (?, ?, ?, FALSE)

```

### 7.3.2.X37.7 Value\_Set Writing Test

Purpose: Verifying that writes to the Value\_Set are reflected atomically into the object's properties.

Test Concept: Writing the Value\_Set shall be reflected atomically in the Value\_Set and Present\_Value properties, while the old Present\_Value is stored into the Value\_Before\_Change property, and the Value\_Change\_Time shall update.

Test Steps:

1. READ OldV = Present\_Value
2. WRITE Value\_Set = (NewV, any valid value)
3. VERIFY Value\_Set = NewV
4. VERIFY Present\_Value = NewV
5. VERIFY Value\_Before\_Change = OldV
6. VERIFY Value\_Change\_Time = (approximately the current local time)

### 7.3.2.X37.8 Value\_Before\_Change Writing Test

Purpose: To verify the correct atomic operations of writing the Accumulator Value\_Before\_Change.

Test Concept: Write the Value\_Before\_Change and verify that it is reflected atomically in the Value\_Before\_Change property, while the old Present\_Value is stored into the Value\_Set property, and the Value\_Change\_Time shall update.

Test Steps:

1. READ OldV = Present\_Value
2. WRITE Value\_Before\_Change = (NewV, any valid value)
3. VERIFY Value\_Before\_Change = NewV
4. VERIFY Value\_Set = OldV
5. VERIFY Value\_Change\_Time = (approximately the current local time)

[Add new **Clause 7.3.2.X38**, p. 185]

[Reviewer Note: Clause 7.3.2 (Object Specific Tests) contains no tests for the Pulse Converter object type.]

### 7.3.2.X38 Pulse Converter Object Tests

### 7.3.2.X38.1 Adjust\_Value Write Test

Purpose: To verify the correct write operation of a Pulse Converter's several properties, when writing the Adjust\_Value. Count\_Before\_Change reflects the prior Count before a write to the Adjust\_Value property.

Test Configuration: Select a Pulse Converter object for which the pulse can be stopped so that Count remains unchanged during the test.

Test Steps:

2. READ OldC = Count
3. WRITE Adjust\_Value = (NewA, any valid value)
4. VERIFY Present\_Value = Count \* Scale\_Factor5.      VERIFY Count\_Change\_Time  $\approx$  (the current local time)
6. VERIFY Count\_Before\_Change = OldC

### 7.3.2.X38.2 Scale\_Factor Test

Purpose: To verify the correct effect of Scale\_Factor on the Present\_Value in Pulse Converter.

Test Concept: The IUT shall be configured with a Scale\_Factor whose influence on the behavior of Present\_Value is observable. After Present\_Value is read, then the value derived from Count and Scale\_Factor is compared to the expected Present\_Value.

Test Steps:

1. IF (Scale\_Factor is writable) THEN  
    WRITE Scale\_Factor = (any valid value V<sub>1</sub>)  
ELSE  
    MAKE (Scale\_Factor equal any valid value V<sub>1</sub>)
2. VERIFY (Present\_Value = conversion specified by Scale\_Factor V<sub>1</sub> coefficient times the Count property)

### 7.3.2.X38.3 Out\_Of\_Service Pulse Converter Test

Purpose: This test case verifies that Present\_Value and the Reliability property are writable when Out\_Of\_Service is TRUE. It also verifies the interrelationship between the Out\_Of\_Service, Status\_Flags, and Reliability properties. If the PICS indicates that the Out\_Of\_Service property of the object under test is not writable, and if the value of the property cannot be changed by other means, then this test shall be omitted.

Test Concept: The IUT will select one instance of each appropriate object type and test it as described. If the Reliability property is not supported, then step 5 shall be omitted.

Test Steps:

1. IF (Out\_Of\_Service is writable) THEN  
    WRITE Out\_Of\_Service = TRUE  
ELSE  
    MAKE (Out\_Of\_Service TRUE)
2. VERIFY Out\_Of\_Service = TRUE
3. VERIFY Status\_Flags = (?, FALSE, ?, TRUE)
4. REPEAT X = (any values meeting the functional range requirements of 7.2.1) DO {  
    WRITE Present\_Value = X  
    VERIFY Present\_Value = X  
}
5. IF (Reliability is present and writable) THEN  
    REPEAT X = (any values of the Reliability enumeration appropriate to the object type except NO\_FAULT\_DETECTED) DO {  
        WRITE Reliability = X  
        VERIFY Reliability = X  
        VERIFY Status\_Flags = (?, TRUE, ?, TRUE)  
        WRITE Reliability = NO\_FAULT\_DETECTED  
        VERIFY Reliability = NO\_FAULT\_DETECTED

```

        VERIFY Status_Flags = (?, FALSE, ?, TRUE)
    }
6. IF (Out_Of_Service is writable) THEN
    WRITE Out_Of_Service = FALSE
ELSE
    MAKE (Out_Of_Service FALSE)
7. VERIFY Out_Of_Service = FALSE
8. VERIFY Status_Flags = (?, ?, ?, FALSE)

```

#### 7.3.2.X38.4 Update\_Time Reflects Change to the Count and is Updated Atomically Test

Purpose: To verify the correct atomic operations of change to the Pulse Converter's several properties, for an inherent change in Count.

Test Steps:

1. READ OldV = Present\_Value
2. READ OldC = Count
3. READ OldU = Update\_Time
4. READ OldT = Count\_Change\_Time
5. READ OldA = Adjust\_Value
6. READ OldS = Scale\_Factor
7. READ OldB = Count\_Before\_Change
8. WAIT (for a change in Count to any valid value, different from OldC so that it can be distinguished)
9. CHECK Present\_Value is recalculated, increasing in proportion to the change in Count multiplied by OldS (or such that Present\_Value minus OldA is still the same fixed difference)
10. VERIFY Update\_Time = (approximately the current local time, and different from OldU)
11. VERIFY Count\_Change\_Time = OldT

#### 7.3.2.X38.5 Adjust\_Value Out-of-Range WriteProperty Test

Purpose: To verify the correct atomic operations of change to the Pulse Converter Count property, when an attempt is made to write Adjust\_Value with a value that would cause an overflow or underflow condition in Count. The test is performed once using WriteProperty and once using WritePropertyMultiple, if IUT supports both services.

Test Steps:

1. READ OldV = Present\_Value
2. READ OldC = Count
3. READ OldU = Update\_Time
4. READ OldT = Count\_Change\_Time
5. READ OldA = Adjust\_Value
6. READ OldS = Scale\_Factor
7. READ OldB = Count\_Before\_Change
8. TRANSMIT WriteProperty-Request
  - 'Property Identifier' = Adjust\_Value
  - 'Property Value' = (NewA, a valid value that would cause an overflow or underflow condition in Count)
9. RECEIVE BACnet-Error-PDU
  - 'Error Class' = PROPERTY
  - 'Error Code' = VALUE\_OUT\_OF\_RANGE
10. VERIFY Update\_Time = OldU
11. VERIFY Adjust\_Value = OldA
12. VERIFY Count\_Before\_Change = OldB

[Add new **Clause 7.3.2.X40**, p. 185]

[Reviewer Note: Clause 7.3.2 (Object Specific Tests) contains no tests for the Channel object type.]

### 7.3.2.X40 Channel Object Tests

#### 7.3.2.X40.2 Last\_Priority Test

Purpose: To verify that the initial value of Last\_Priority is 16. To verify that a Channel object correctly retains the priority of written values. To verify that a Last\_Priority will have a default priority of 16 if the last attempt to write to the Present\_Value was done without specifying the priority.

Test Concept: First, confirm that the default value of Last\_Priority is 16. Next, write a valid value to a Channel object using a valid priority level other than 16 and then check the value of Last\_Priority to make sure it shows the specified priority level. Finally, write a valid value to a Channel object again but without specifying priority and then check the value of Last\_Priority to make sure that it now shows 16 again.

Configuration Requirements: Configure entry X of the Channel object's List\_Of\_Object\_Property\_References to refer to a writable Present\_Value property which is either on local device or remote device.

Test Steps:

1. READ LEN = List\_Of\_Object\_Property\_References, ARRAY INDEX = 0
2. VERIFY Last\_Priority = 16
3. WRITE Present\_Value = (Any valid value), PRIORITY = (Y: Any valid value < 16)
4. WAIT (**Channel Write Fail Time** \* LEN)
5. VERIFY Last\_Priority = Y
6. WRITE Present\_Value = (Any valid value)
7. WAIT (**Channel Write Fail Time** \* LEN)
8. VERIFY Last\_Priority = 16

#### 7.3.2.X40.3 WriteGroup Service Support Test

Purpose: To verify that the Present\_Value of the Channel object can be written by WriteGroup Service

Test Concept: The Channel object, O1, is written to via WriteGroup and it is verified that the Present\_Value of the object is updated correctly.

Test Steps:

1. READ X = O1, Present\_Value
2. TRANSMIT WriteGroup-Request,  
'Group Number' = (one of the Control\_Group values configured in O1),  
'Write Priority' = (any valid value),  
'Change List' = (O1's channel number, no overriding priority, Y: a value different than X)
3. VERIFY Present\_Value = Y

#### 7.3.2.X40.4 Propagation Entirety Test

Purpose: To verify that the Channel object keeps propagating the value to each local/remote object property reference in its List\_Of\_Object\_Property\_References after propagating the value fails for one or more target.

Test Concept: The Channel object, O1, is configured with at least 1 referenced target which the Channel object won't send the write to due to an invalid datatype coercion error. The Channel object's Present\_Value is written, and the writes to remote targets are checked. Once all writes complete, the Write\_Status is verified to be FAILED and all targets the Channel object sent the write to are verified to have accepted the written value.

Configuration Requirements: Configure the Channel object's List\_Of\_Object\_Property\_References so that at least one of the target references (contained in entry X of List\_Of\_Object\_Property\_References) will result in an invalid datatype coercion when the value WrittenValue is written to the Channel object. The rest of the target reference(s) shall be selected such that they will accept the written value as is without coercion. Refer to the Table 12-63 – Datatype Coercion Rules from ASHRAE 135 for the invalid datatypes.

Test Steps:

1. READ LEN = List\_Of\_Object\_Property\_References, ARRAY INDEX = 0
2. READ N1 = List\_Of\_Object\_Property\_References, ARRAY INDEX = X
3. WRITE Present\_Value = WRITTEN\_VALUE
4. WAIT (**Channel Write Fail Time** \* LEN)
5. REPEAT REF = (each reference in O1.List\_Of\_Object\_Property\_References) {
  - IF (REF is not contained in the IUT) THEN
    - RECEIVE WriteProperty-Request
      - 'Object Identifier' = (Object Identifier of N1),
      - 'Property Identifier' = (Property Identifier of N1),
      - 'Property Value' = WrittenValue
    - IF (REF <> N1) THEN
      - TRANSMIT BACnet-SimpleACK-PDU
6. VERIFY Write\_Status = FAILED
7. REPEAT REF = (each reference in O1.List\_Of\_Object\_Property\_References) {
  - IF (REF <> N1) THEN
    - VERIFY REF = WRITTEN\_VALUE

### 7.3.2.X40.5 Write\_Status Test

Purpose: To verify that the Write\_Status of the Channel object is IDLE when it's List\_Of\_Object\_Property\_References is empty and is IN\_PROGRESS while the Channel object's Present\_Value is being propagated, FAILED when propagation failed, and SUCCESSFUL when propagation succeeds.

Test Concept: The Channel object's List\_Of\_Object\_Property\_References is read and verified to be empty. Then, the Channel object's Write\_Status is verified to be IDLE. Next, any valid value is written to the Channel object's Present\_Value and Write\_Status is verified to be IDLE still. An object property reference, R1, that the IUT cannot reach is set into the List\_Of\_Object\_Property\_References and then any valid value is written to the Present\_Value and the Write\_Status is verified to be IN\_PROGRESS. After the IUT determines that the referenced device is offline, the Write\_Status is verified to be FAILED. Finally, any valid reference, R2, that will cause successful value propagation is set to the List\_Of\_Object\_Property\_References and the test is repeated to verify that Write\_Status becomes SUCCESSFUL.

Test Steps:

1. READ LEN = List\_Of\_Object\_Property\_References, ARRAY INDEX = 0
- IDLE test
2. READ L = List\_Of\_Object\_Property\_References
  3. VERIFY L = (empty)
  4. VERIFY Write\_Status = IDLE
  5. WRITE Present\_Value = (any valid value)
  6. VERIFY Write\_Status = IDLE

-- IN\_PROGRESS and FAIL test

7. WRITE List\_Of\_Object\_Property\_References = (R1)-- write the whole array
8. WRITE Present\_Value = (any valid value)
9. VERIFY Write\_Status = IN\_PROGRESS
10. WAIT (**Channel Write Fail Time** \* LEN)
11. VERIFY Write\_Status = FAILED

-- SUCCESSFUL test

12. WRITE List\_Of\_Object\_Property\_References = (R2)-- write the whole array
13. WRITE Present\_Value = (any valid value)
14. WAIT (**Channel Write Fail Time** \* LEN)
15. VERIFY Write\_Status = SUCCESSFUL

### 7.3.2.X40.6 Allow\_Group\_Delay\_Inhibit Test

Purpose: To verify that no Execution\_Delay will be applied to any of the writes if Allow\_Group\_Delay\_Inhibit is TRUE.

Test Concept: Setup List\_Of\_Object\_Property\_References to contain 2 valid entries PR1, PR2 and provide each with an execution delay (ED1 and ED2). Set Allow\_Group\_Delay\_Inhibit to TRUE so that no delays will occur between writes to referenced properties. Write to the Channel object's Present\_Value and verify that no delay occurs between writes to the referenced properties.

Set Allow\_Group\_Delay\_Inhibit to FALSE so that delays will occur between writes to referenced properties. Write to the Channel object's Present\_Value and verify that delays occur between writes to the referenced properties.

Configuration Requirements: PR1 and PR2 shall be references to writable properties and shall be the same datatype. ED1 and ED2 shall be values which are large enough that the delay between writes is sufficient for the test. V1 and V2 shall be of the expected datatype for PR1 and PR2 so that no coercion occur and shall be different values. This test shall be skipped if the Channel object does not support at least 2 entries in the List\_Of\_Object\_Property\_References.

Test Steps:

1. READ LEN = List\_Of\_Object\_Property\_References, ARRAY INDEX = 0
- Setup the Channel object
2. WRITE List\_Of\_Object\_Property\_References = (PR1, PR2)
3. WRITE Execution\_Delay = (ED1, ED2)

-- Test that delays are inhibited

4. WRITE Allow\_Group\_Delay\_Inhibit = TRUE
5. WRITE Present\_Value = V1
6. WAIT (**Channel Write Fail Time** \* LEN)
7. VERIFY PR1 = V1
8. VERIFY PR2 = V1
9. VERIFY Write\_Status = SUCCESSFUL

-- Test that delays are not inhibited

10. WRITE Allow\_Group\_Delay\_Inhibit = FALSE
11. WRITE Present\_Value = V2
12. WAIT (**Channel Write Fail Time** \* LEN)
13. VERIFY PR1 = V1
14. VERIFY PR2 = V1
15. VERIFY Write\_Status = IN\_PROGRESS
16. WAIT (ED1)
17. VERIFY PR1 = V2
18. VERIFY PR2 = V1
19. VERIFY Write\_Status = IN\_PROGRESS
20. WAIT (ED2 - ED1)
21. VERIFY PR2 = V2
22. VERIFY Write\_Status = SUCCESSFUL

### 7.3.2.X40.7 Numeric to BOOLEAN Coercion Rule Test

Purpose: To verify that the Channel object correctly propagates all numeric datatype values to a BOOLEAN target based on Coercion Rule 1 – Numeric to BOOLEAN.

Test Concept: Write a value of 0 to the Present\_Value of the Channel object with a BOOLEAN target object property reference, verify that a target object property has a value of FALSE, and that Write\_Status of the Channel object shows SUCCESSFUL. When any non-zero numeric value is written to the Present\_Value of the same Channel object, verify that a target object property has a value of TRUE, and a Write\_Status shows SUCCESSFUL.

Configuration Requirements: Configure entry X of the Channel object's List\_Of\_Object\_Property\_References to refer to a writable BOOLEAN object property.

Test Steps:

1. READ LEN = List\_Of\_Object\_Property\_References, ARRAY INDEX = 0
2. READ B = List\_Of\_Object\_Property\_References, ARRAY INDEX = X
3. WRITE Present\_Value = 0
4. WAIT (Channel Write Fail Time \* LEN)
5. VERIFY B = FALSE
6. VERIFY Write\_Status = SUCCESSFUL
7. WRITE Present\_Value = (Any non-zero numeric value)
8. WAIT (Channel Write Fail Time \* LEN)
9. VERIFY B = TRUE
10. VERIFY Write\_Status = SUCCESSFUL

### 7.3.2.X40.8 BOOLEAN to Numeric Coercion Rule Test

Purpose: To verify that the Channel object can correctly propagate BOOLEAN values to a numeric target object property reference based on Coercion Rule 2 – BOOLEAN to Numeric defined in ASHRAE 135.

Test Concept: When a value of FALSE is written to the Present\_Value of the Channel object with a numeric target object property reference, verify that the target object property has a value of 0, and a Write\_Status of the Channel object shows SUCCESSFUL. When a value of TRUE is written to the present value of the same Channel object, verify that a target object property has a value of 1, and a Write\_Status shows SUCCESSFUL.

Configuration Requirements: Configure entry X of the Channel object's List Of Object Property References to refer to a writable numeric object property on the IUT. The referenced property shall not be 0 at the start of the test.

Test Steps:

1. READ LEN = List\_Of\_Object\_Property\_References, ARRAY INDEX = 0
2. READ N = List\_Of\_Object\_Property\_References, ARRAY INDEX = X
3. VERIFY N <> 0 -- non-zero so that coercion is verified in the following write
4. WRITE Present\_Value = FALSE
5. WAIT (Channel Write Fail Time \* LEN)
6. VERIFY N = 0
7. VERIFY Write\_Status = SUCCESSFUL
8. WRITE Present\_Value = TRUE
9. WAIT (Channel Write Fail Time \* LEN)
10. VERIFY N = 1
11. VERIFY Write\_Status = SUCCESSFUL

### 7.3.2.X40.9 Unsigned/INTEGER/REAL/Double to Numeric Coercion Rule Test

Purpose: To verify that the Channel object correctly propagates Unsigned, INTEGER, REAL or Double datatype values to a numeric target object property reference.

Test Concept: Select a Channel object with a numeric target property, N. Select an Unsigned, INTEGER, REAL, or Double value, V1, which is in the acceptable range for N, and which coerces to value V2 based on N's datatype. Write V1 to the Present\_Value of the Channel object. Verify that the N changes to V2 and that Write\_Status of the Channel object is SUCCESSFUL.

Configuration Requirements: Configure entry X of the Channel object's List\_Of\_Object\_Property\_References to refer to the selected numeric property N. Configure the Channel object with no execution delays.

Test Steps:

1. READ LEN = List\_Of\_Object\_Property\_References, ARRAY INDEX = 0
2. VERIFY List\_Of\_Object\_Property\_References = N, ARRAY INDEX = X
3. WRITE Present\_Value = V1
4. WAIT (**Channel Write Fail Time** \* LEN)
5. VERIFY N = V2
6. VERIFY Write\_Status = SUCCESSFUL

### 7.3.2.X40.10 Invalid Datatype Coercion Test

Purpose: To check that the Channel object does not write to a target object property reference and the Write\_Status indicates FAILED when invalid datatype coercion occur.

Test Concept: When an invalid data type value is written to a Present\_Value of the Channel object, verify that a target object reference value does not change.

Configuration Requirements: Configure entry X of the Channel object's List\_Of\_Object\_Property\_References to refer to a writable object property of a specific data type on the IUT such that the Channel object will fail to propagate InvalidDataTypeValue. Refer to the Table 12-63 - Datatype Coercion Rules from ASHRAE 135 for the invalid datatypes.

Test Steps:

1. READ LEN = List\_Of\_Object\_Property\_References, ARRAY INDEX = 0
2. READ N = List\_Of\_Object\_Property\_References, ARRAY INDEX = X
3. WRITE Present\_Value = (InvalidDataTypeValue: Any invalid data type value)
4. WAIT (**Channel Write Fail Time** \* LEN)
5. VERIFY N <> InvalidDataTypeValue
6. VERIFY Write\_Status = FAILED

### 7.3.2.X40.11 No Coercion Test

Purpose: To check that the Channel object can successfully write to a target object property reference without any value conversions and Write\_Status indicates SUCCESSFUL when no coercion occurs.

Test Concept: When a valid data type value is written to a Present\_Value of the Channel object using a value that require no coercion, verify that a written value is directly mapped to a target object reference and a Write\_Status of the Channel object shows SUCCESSFUL.

Configuration Requirements: Configure entry X of the Channel object's List\_Of\_Object\_Property\_References to refer to a writable object property of a specific data type on the IUT such that no coercion will occur. Refer to the Table 12-63 – Datatype Coercion Rules from ASHRAE 135 for the data types that require no coercion.

Test Steps:



1. READ LEN = List\_Of\_Object\_Property\_References, ARRAY INDEX = 0
2. READ N = List\_Of\_Object\_Property\_References, ARRAY INDEX = X
3. WRITE Present\_Value = (ValidDataTypeValue: Any value of a datatype that requires no coercion)
4. WAIT (**Channel Write Fail Time** \* LEN)
5. VERIFY N = ValidDataTypeValue
6. VERIFY Write\_Status = SUCCESSFUL

### 7.3.2.X40.12 Write Priority Test

Purpose: To check that the Channel object uses a priority level specified by a write service when the Channel object propagates its Present\_Value to (a) target object property reference(s). If no priority level is specified, check that 16 is used by default.

Test Concept: When a valid data type value is written to a Present\_Value of the Channel object by a WriteProperty request and a 'Priority' is provided in the write, the Channel object will use this same priority to command the referenced properties. When another value is written to a Present\_Value of the Channel object by a WriteProperty request with no 'Priority' specified, the Channel object will use a priority 16 by default to command the referenced properties.

Configuration Requirements: Configure entry X of the Channel object's List\_Of\_Object\_Property\_References to refer to a writable object property of a specific data type on the IUT such that no coercion will occur. Refer to the Table 12-63 – Datatype Coercion Rules from ASHRAE 135 for the data types that require no coercion. The referenced property must contain a Priority\_Array property.

Test Steps:

1. READ LEN = List\_Of\_Object\_Property\_References, ARRAY INDEX = 0
2. READ N = List\_Of\_Object\_Property\_References, ARRAY INDEX = X
3. TRANSMIT WriteProperty-Request,
  - 'Object Identifier' = (Object Identifier of the Channel object)
  - 'Property Identifier' = Present\_Value
  - 'Property Value' = (V1: Any value of a datatype that requires no coercion)
  - 'Priority' = (P1: Any valid value but 16)
4. WAIT (**Channel Write Fail Time** \* LEN)
5. RECEIVE BACnet-SimpleACK-PDU
6. TRANSMIT ReadProperty-Request,
  - 'Object Identifier' = (Object Identifier of N)
  - 'Property Identifier' = Priority\_Array
  - 'Property Array Index' = P1
7. RECEIVE ReadProperty-ACK,
  - 'Object Identifier' = (Object Identifier of N)
  - 'Property Identifier' = Priority\_Array
  - 'Property Array Index' = P1
  - 'Property Value' = V1
8. TRANSMIT WriteProperty-Request,
  - 'Object Identifier' = (Object Identifier of the Channel object)
  - 'Property Identifier' = Present\_Value
  - 'Property Value' = (V2: Any value of a datatype that requires no coercion)
9. WAIT (**Channel Write Fail Time** \* LEN)
10. TRANSMIT ReadProperty-Request,
  - 'Object Identifier' = (Object Identifier of N)
  - 'Property Identifier' = Priority\_Array
  - 'Property Array Index' = 16
11. RECEIVE ReadProperty-ACK,
  - 'Object Identifier' = (Object Identifier of N)
  - 'Property Identifier' = Priority\_Array
  - 'Property Array Index' = 16
  - 'Property Value' = V2

### 7.3.2.X40.13 Writing with a NULL Value Test

Purpose: To check that the Channel object ignores datatype errors when writing a NULL value to a non-commandable target.

Test Concept: This test is to check a special exception of Write\_Status reporting SUCCESSFUL after propagating a NULL value to both a commandable and non-commandable property. Writing a NULL value to a commandable property will result in a property relinquishing a value to a Relinquish\_Default value. However, writing a NULL value to a non-commandable property will result in a property remaining a current value. If a non-commandable target property is on a remote device, the IUT will receive either an ERROR\_INVALID\_DATATYPE or REJECT\_INVALID\_PARAMETER\_DATA\_TYPE. The Write\_Status after such events should report SUCCESSFUL instead of FAILED.

Configuration Requirements: Configure entry X of the Channel object's List\_Of\_Object\_Property\_References to refer to a commandable property that accepts a NULL value, and configure entry Y of the List\_Of\_Object\_Property\_References to a non-commandable property that rejects a NULL value with either ERROR\_INVALID\_DATATYPE or REJECT\_INVALID\_PARAMETER\_DATA\_TYPE. For a commandable property, all prioritized commands have to be relinquished, and any minimum on/off time has to be accounted for prior to the test. An initial value of a commandable property, N1, must be different from a value of its Relinquish\_Default. N1's Priority\_Array has only one non-Null value in it and it is in the priority array level that the Channel object is targeting.

Test Steps:

1. READ LEN = List\_Of\_Object\_Property\_References, ARRAY INDEX = 0  
-- Read an initial value of target properties
2. READ P1 = List\_Of\_Object\_Property\_References: ARRAY INDEX = X, which is a commandable property
3. READ P2 = List\_Of\_Object\_Property\_References: ARRAY INDEX = Y, which is a non-commandable property
- Find expected values of the target properties after a NULL value is written to them
4. READ V1 = P1.Relinquish\_Default
5. READ V2 = P2
- Make the Channel object to propagate NULL value to targets
6. WRITE Present\_Value = NULL
7. WAIT (**Channel Write Fail Time** \* LEN)
8. IF (P2 is on external) THEN  
    RECEIVE WritePropertyMultiple-Error  
    'Error Class' = PROPERTY,  
    'Error Code' = INVALID\_DATATYPE |  
    RECEIVE BACnet-Reject-PDU  
    'Reject Reason' = INVALID\_PARAMETER\_DATATYPE
- Check that P1 has Relinquish\_Default value and P2 remains the same
9. VERIFY P1 = V1
10. VERIFY P2 = V2
- Check that the Channel object ignores the datatype error and Write\_Status is SUCCESSFUL
11. VERIFY Write\_Status = SUCCESSFUL

[Add new **Clause 7.3.2.X45**, p. 185]

[Reviewer Note: Clause 7.3.2 (Object Specific Tests) contains no tests for the Elevator Group object type.]

### 7.3.2.X45 Elevator Group Object Tests

#### 7.3.2.X45.1 Machine\_Room\_ID property references a Positive Integer Value Object

Purpose: To verify that the Machine\_Room\_ID property of an Elevator Group object can only reference a Positive Integer Value object or an object with instance number of 4194303.

Test Concept: The Machine\_Room\_ID property of an Elevator Group object, EG1, is read to verify that it contains an object reference to a Positive Integer Value object, PIV, or an object with instance number of 4194303. If the property is writable, an attempt is made to write an object reference, O1, that is not a Positive Integer Value object and has an instance number 0-4194302 (inclusive) to verify that an error is returned.

Test Steps:

1. IF (Machine\_Room\_ID contains room identification number) THEN  
     VERIFY (EG1), Machine\_Room\_ID = (PIV)  
   ELSE  
     VERIFY (EG1), Machine\_Room\_ID = (any object type, 4194303)
2. IF (Machine\_Room\_ID is writeable) THEN  
     Transmit WriteProperty-Request  
       'Object Identifier'= EG1,  
       'Property Identifier'= Machine\_Room\_ID,  
       'Property Value'= O1  
     Receive BACnet-Error-PDU  
       'Error Class'= PROPERTY,  
       'Error Code'= VALUE\_OUT\_OF\_RANGE

#### 7.3.2.X45.2 Linking of Lift and Escalator Objects under Group\_Members property of the Elevator Group Object

Purpose: This test verifies that objects in the Group\_Members property of Elevator Group objects contain a reference back to the Elevator Group that has it listed as a member.

Test Concept: The Group\_Members property of each Elevator Group object is read to identify member Lift and Escalator objects. The Elevator\_Group property is read from each member Lift object and Escalator object to verify it contains a reference back to the appropriate Elevator Group object. The Elevator\_Group property of the remaining Lift and Escalator objects are read to verify that it contains an object identifier instance of 4194303.

Configuration Requirements: If the IUT supports a Group\_Members property that can be made to contain a reference to one or more Lift objects, then it shall be configured as such. If the IUT supports a Group\_Members property that can be made to contain a reference to one or more Escalator objects, it shall be configured as such.

Test Steps:

1. REPEAT EGO = (each Elevator Group object in the IUT) {  
     READ L1 = (EGO, Group\_Members)  
     IF (L1 is not empty) THEN  
       REPEAT O1 = (each Lift or Escalator object in L1) {  
         READ EGP = (O1, Elevator\_Group)  
         VERIFY EGP = EGO  
       }  
     }  
   }
3. REPEAT O1 = (each remaining Lift or Escalator object in the IUT) {  
     READ EGP = (O1, Elevator\_Group)  
     VERIFY EGP = (any object type, 4194303)  
   }

### 7.3.2.X45.3 Landing\_Call\_Control test

Purpose: To verify that writing to the Landing\_Call\_Control property updates the Landing\_Call\_Control and Landing\_Calls properties in the Elevator Group object and updates the Assigned\_Landing\_Calls property of a linked Lift object

Test Concept: The Landing\_Call\_Control property of an Elevator Group object (EG1) is written with a value that represents a request to travel upwards from FN1. The Landing\_Call\_Control and Landing\_Calls properties of EG1 and the Assigned\_Landing\_Calls property of the linked Lift object (L1) are checked to verify they updated correctly. The Landing\_Call\_Control property is written with a value that represents a request to travel downwards from FN2 and the aforementioned properties are checked again. The optional 'floor-text' parameter is used in one of the WRITE steps to verify the server will ignore this parameter when present. In the test steps, DF represents a valid destination floor.

Configuration Requirements: Lift object (L1) is contained in the Group\_Members property of the Elevator Group object (EG1) and has a door at array index Y on the same side of the landing call. FN1 and FN2 values should be sufficiently far away from the current position of L1 to allow for reading of the property values. No other processes shall be generating landing calls during this test.

Notes to Tester: If the Elevator Group contains more than 1 lift, the value written to Landing\_Call\_Control may get assigned to any other lift in the group based on the lift algorithm.

Test Steps:

1. WRITE EG1, Landing\_Call\_Control = (FN1, UP | DF (DF > FN1), "test string")
2. VERIFY EG1, Landing\_Call\_Control = (FN1, UP | DF, floor-text (optional))
3. VERIFY EG1, Landing\_Calls = (FN1, UP | DF, floor-text (optional))
4. IF (L1 contains the Assigned\_Landing\_Calls property)  
    VERIFY L1, Assigned\_Landing\_Calls, ARRAY\_INDEX (Y) = (FN1, UP)
5. WAIT (a time interval sufficient for the car to complete the call + **Internal Processing Fail Time**)
6. VERIFY EG1, Landing\_Calls = ()
7. IF (L1 contains the Assigned\_Landing\_Calls property)  
    VERIFY L1, Assigned\_Landing\_Calls, ARRAY\_INDEX (Y) = ()
8. WRITE EG1, Landing\_Call\_Control = (FN2, DOWN | DF (DF < FN2))
9. VERIFY EG1, Landing\_Call\_Control = (FN2, DOWN | DF, floor-text (optional))
10. VERIFY EG1, Landing\_Calls = (FN2, DOWN | DF, floor-text (optional))
11. IF (L1 contains the Assigned\_Landing\_Calls property)  
    VERIFY L1, Assigned\_Landing\_Calls = (FN1, DOWN)
12. WAIT (a time interval sufficient for the car to complete the call + **Internal Processing Fail Time**)
13. VERIFY EG1, Landing\_Calls = ()
14. IF (L1 contains the Assigned\_Landing\_Calls property)  
    VERIFY L1, Assigned\_Landing\_Calls, ARRAY\_INDEX (Y) = ()

[Add new **Clause 7.3.2.X46**, p. 185]

[Reviewer Note: Clause 7.3.2 (Object Specific Tests) contains no tests for the Lift object type.]

### 7.3.2.X46 Lift Object Tests

#### 7.3.2.X46.1 Array Size of the Lift Object Properties Based on Car Door size.

Purpose: To verify that the size of the arrays for the Car\_Door\_Text, Assigned\_Landing\_Calls, Making\_Car\_Call, Registered\_Car\_Call, Car\_Door\_Status, Car\_Door\_Command, and Landing\_Door\_Status properties are the same.

Test Concept: The array size for each of the above properties, if present, is read and the sizes are compared to verify they are all equal.

Test Steps:

1. VERIFY (L1), Car\_Door\_Text = (Number of car doors present in the Lift), ARRAY INDEX = 0
2. VERIFY (L1), Assigned\_Landing\_Calls = (Number of car doors present in Lift), ARRAY INDEX = 0
3. VERIFY (L1), Making\_Car\_Call = (Number of car doors present in the Lift), ARRAY INDEX = 0
4. VERIFY (L1), Registered\_Car\_Call = (Number of car doors present in the Lift), ARRAY INDEX = 0
5. VERIFY (L1), Car\_Door\_Status = (Number of car doors present in the Lift), ARRAY INDEX = 0
6. VERIFY (L1), Car\_Door\_Command = (Number of car doors present in the Lift), ARRAY INDEX = 0
7. VERIFY (L1), Landing\_Door\_Status = (Number of car doors present in the Lift), ARRAY INDEX = 0
8. CHECK (Array index 0 of all these properties shall be same)

### 7.3.2.X46.2 Lift Properties Operational Test

Purpose: To verify that the property values in the Lift object update when it responds to a call.

Test Concept: The test starts with the Lift object, L1, in the lowest floor that it serves, LF, and property values are checked. A request is made to move the lift to the highest floor that it serves, HF, and property values are checked while the lift is moving and again when the lift arrives at HF. If the IUT does not contain the property specified in the test step, that step shall be skipped. In the test steps, DSR is a specific array index corresponding to the car door servicing the request.

Configuration Requirements: At the start of the test, the lift corresponding to L1 is at LF, and there are no active calls for L1. Throughout the test, L1 is in a normal operating state such that Car\_Mode = NORMAL, Out\_Of\_Service = FALSE, and no other processes shall be attempting to control L1.

Test Steps:

1. READ LF = Car\_Position
2. READ DS1 = Car\_Door\_Status
3. VERIFY Floor\_Text = (any value), ARRAY INDEX = LF
4. VERIFY Floor\_Text = (any value), ARRAY INDEX = HF
5. REPEAT N = (each array element) DO {
  - VERIFY Assigned\_Landing\_Calls = {}, ARRAY INDEX = N
6. REPEAT N = (each array element) DO {
  - VERIFY Registered\_Car\_Calls = {}, ARRAY INDEX = N
7. VERIFY Car\_Moving\_Direction <> UP | DOWN
8. VERIFY Car\_Mode = NORMAL
9. VERIFY Next\_Stopping\_Floor = LF
10. VERIFY Passenger\_Alarm = FALSE
11. VERIFY Reliability = NO\_FAULT\_DETECTED
12. VERIFY Out\_Of\_Service = FALSE
13. VERIFY Car\_Drive\_Status = STATIONARY | UNKNOWN
14. REPEAT N = (each array element) DO {
  - VERIFY Landing\_Door\_Status = (a list containing an entry {LF, DS1[N]}), ARRAY INDEX = N
15. MAKE (A command that will cause L1 to travel to HF)
  - Complete steps 16 – 19 before L1 reaches HF
16. IF (command was generated via Landing call) THEN
  - VERIFY Assigned\_Landing\_Calls = (HF, DOWN), ARRAY INDEX = DSR
- ELSE --command was generated via Car call
  - VERIFY Making\_Car\_Call = (HF), ARRAY INDEX = DSR
  - VERIFY Registered\_Car\_Calls = (HF), ARRAY INDEX = DSR
  - VERIFY Car\_Assigned\_Direction = (UP)
17. VERIFY Car\_Moving\_Direction = UP
18. VERIFY Next\_Stopping\_Floor = HF
19. VERIFY Car\_Drive\_Status <> STATIONARY
20. WAIT (for L1 to reach HF) + **Internal Processing Fail Time**

```

21. REPEAT N = (each array element) DO {
    VERIFY Registerd_Car_Calls = {}, ARRAY INDEX = N
}
22. VERIFY Car_Position = HF
23. VERIFY Car_Moving_Direction <> UP | DOWN
24. VERIFY Next_Stopping_Floor = HF
25. READ DS2 = Car_Door_Status
26. REPEAT (N = each array element) DO{
    VERIFY Landing_Door_Status = (a list containing an entry {LF, DS2[N]}), ARRAY INDEX = N
}

```

### 7.3.2.X46.3 Out\_Of\_Service, Status\_Flags for Lift Object

Purpose: To verify the interrelationship between Out\_Of\_Service and Status\_Flags and that properties dictated by the standard to be writable when Out\_Of\_Service is TRUE are writable when Out\_Of\_Service is TRUE.

Test Concept: Out\_Of\_Service is set to TRUE and Status\_Flags is checked to verify the Out\_Of\_Service flag is set. While Out\_Of\_Service is TRUE, each of the properties (represented by LP), if present in the object, is read to obtain the current property value, X, and written with a different property value, Y. The property value is read again to verify it changed to Y.

LP = (Assigned\_Landing\_Calls, Registered\_Car\_Call, Car\_Position, Car\_Moving\_Direction, Car\_Assigned\_Direction, Car\_Door\_Status, Car\_Door\_Zone, Car\_Load, Next\_Stopping\_Floor, Passenger\_Alarm, Energy\_Meter, Car\_Drive\_Status, Fault\_Signals, Landing\_Door\_Status, Making\_Car\_Call, Car\_Door\_Command, and Car\_Mode)

Test Steps:

```

1. WRITE Out_Of_Service = TRUE
2. VERIFY Out_Of_Service = TRUE
3. VERIFY Status_Flags = (?, ?, ?, TRUE)
4. REPEAT P = (each property in LP present in the object) DO{
    READ X = P
    WRITE P = Y
    WAIT Internal Processing Fail Time
    VERIFY (P=Y)
}

```

### 7.3.2.X46.4 Energy\_Meter\_Ref Property Tests

Purpose: To verify linking of Energy\_Meter property and Energy\_Meter\_Ref property.

Test Concept: If the Energy\_Meter\_Ref property of an object (O1) is present and initialized (contains an instance other than 4194303), then the Energy\_Meter property, if present, shall have a value of 0.0. If Energy\_Meter\_Ref is present and is uninitialized, then the value of Energy\_Meter property shall have any valid value.

Test Steps:

```

1. IF (Energy_Meter_Ref is present and initialized with instance other than 4194303) THEN
    VERIFY Energy_Meter = 0.0
ELSE
    VERIFY Energy_Meter = (Any Valid Value)

```

[Add new **Clause 7.3.2.X47**, p. 185]

[Reviewer Note: Clause 7.3.2 (Object Specific Tests) contains no tests for the Escalator object type.]

### 7.3.2.X47 Escalator Object Tests

#### 7.3.2.X47.1 Out\_Of\_Service, Status\_Flags for Escalator object

Purpose: To verify the interrelationship between Out\_Of\_Service and Status\_Flags and that properties dictated by the standard to be writable when Out\_Of\_Service is TRUE are writable when Out\_Of\_Service is TRUE.

Test Concept: Out\_Of\_Service is set to TRUE and Status\_Flags is checked to verify the Out\_Of\_Service flag is set. While Out\_Of\_Service is TRUE, each of the properties (represented by EP), if present in the object, is read to obtain the current property value, X, and written with a different property value, Y. The property value is read again to verify it changed to Y.

EP = (Power\_Mode, Operation\_Direction, Escalator\_Mode, Energy\_Meter, Fault\_Signals, and Passenger\_Alarm)

Test Steps:

1. WRITE Out\_Of\_Service = TRUE
2. VERIFY Out\_Of\_Service = TRUE
3. VERIFY Status\_Flags = (?, ?, ?, TRUE)
4. REPEAT P = (each property in LP present in the object) DO {
  - READ X = P
  - WRITE (P = Y)
  - WAIT **Internal Processing Fail Time**
  - VERIFY (P=Y)

[Add new **Clause 7.3.2.X53**, p. 185]

[Reviewer Note: Clause 7.3.2 (Object Specific Tests) contains no tests for the Load Control object type.]

### 7.3.2.X53 Load Control Object Tests

The Load Control object defines a standardized object whose properties represent the externally visible characteristics of a mechanism for controlling load requirements. A BACnet device can use a Load Control object to allow external control over the shedding of a load that it controls. The mechanisms by which the loads are shed are not visible to the BACnet client. The Load Control Object utilizes parameter control through its writable Requested\_Shed\_Level, Start\_Time, Shed\_Duration, Duty\_Window, Enable, and Shed\_Levels properties.

#### 7.3.2.X53.1 Requested\_Shed\_Level property test with LEVEL choice

Purpose: To verify that the IUT can accept and execute a shed request with LEVEL choice.

Test Concept: The Requested\_Shed\_Level property of the Load Control object is set to a LEVEL choice and it is verified that the Load Control object behaves as per the Load Control state machine.

Configuration Requirements: The IUT shall be configured so that Present\_Value is equal to SHED\_INACTIVE, at the start of the test. Writing Start\_Time and/or Shed\_Duration with values such that current time is after ST+SD forces Present\_Value to become equal to SHED\_INACTIVE.

Notes to Tester: The writing of Duty\_Window can be skipped, for the tester to see that the VERIFY Duty\_Window = DW during a pending or active shed event, that property takes on PAV, the configured pre-agreed upon value.

Test Steps:

1. VERIFY Requested\_Shed\_Level = (DRSL : one of the default Requested\_Shed\_Level values for a previous shed request, not necessarily the LEVEL default of 0)
2. VERIFY Expected\_Shed\_Level = DRSL
3. VERIFY Actual\_Shed\_Level = DRSL
4. VERIFY Present\_Value = SHED\_INACTIVE
5. VERIFY Shed\_Duration = 0
6. VERIFY Start\_Time = (the fully unspecified datetime value)
7. VERIFY Duty\_Window = (PAV, the pre-agreed upon value)
8. WRITE Enable = TRUE
9. WRITE Shed\_Duration = (SD, any value appropriate to the object)
10. WRITE Start\_Time = (ST, any valid start time including values in the past, present or future, but limited to such that current\_time is before ST + SD)
11. WRITE Duty\_Window = (DW, any value appropriate to the object)
12. WRITE Requested\_Shed\_Level = (a value appropriate to the object with a LEVEL choice, that is not equal to the default value: 0)
13. IF (current time is before ST) THEN  
     VERIFY Present\_Value = (SHED\_REQUEST\_PENDING,  
         SHED\_COMPLIANT or  
         SHED\_NON\_COMPLIANT)  
     WAIT (until Start\_Time)
14. VERIFY Present\_Value = (SHED\_COMPLIANT or SHED\_NONCOMPLIANT)
15. IF (ST + DW < ST + SD and ST + DW is in the future) THEN  
     WAIT (until ST+DW)
16. IF (current time is after ST+DW) THEN  
     IF (Actual\_Shed\_Level does not comply with Requested\_Shed\_Value) THEN  
         VERIFY Present\_Value = SHED\_NONCOMPLIANT
17. VERIFY Shed\_Duration = SD
18. VERIFY Start\_Time = ST
19. VERIFY Duty\_Window = DW
20. VERIFY Expected\_Shed\_Level = (any value appropriate to the choice, that is not equal to the default value)
21. VERIFY Actual\_Shed\_Level = (any value appropriate to the choice, that is not equal to the default value)
- the above VERIFY statements apply all through the time that there is a pending or active shed event
22. WAIT (until the shed request has completed, at ST+SD)
23. VERIFY Requested\_Shed\_Level = 0 -- the default LEVEL value
24. VERIFY Expected\_Shed\_Level = 0 -- the default LEVEL value
25. VERIFY Actual\_Shed\_Level = 0 -- the default LEVEL value
26. VERIFY Shed\_Duration = 0
27. VERIFY Start\_Time = (the fully unspecified datetime value)
28. VERIFY Duty\_Window = PAV

### 7.3.2.X53.2 Shed\_Levels property test

Purpose: To verify writability of Shed\_Levels property and verify that when commanded with the LEVEL choice, the Load Control object shall take a shedding action described by the corresponding element in the Shed\_Level\_Descriptions array.

Test Concept: The Shed\_Levels property of the Load Control object being tested is written to BACnetARRAY of unsigned integers representing the shed levels for the LEVEL choice of BACnetShedLevel that have meaning for this particular Load Control object. Verify that is updating correctly. The array shall be ordered by increasing shed amount.

Test Steps:

1. READ N1 = Shed\_Levels, ARRAY INDEX = 0
2. VERIFY (Shed\_Level\_Descriptions = N1, ARRAY INDEX = 0)
3. WRITE Shed\_Levels = (any content that is different from the current value, but nonetheless still ordered by increasing shed amount)
4. READ N2 = Shed\_Levels, ARRAY INDEX = 0 -- obtaining the length of the new value
5. VERIFY (Shed\_Level\_Descriptions = N2, ARRAY INDEX = 0)



### 7.3.2.X53.3 Load Control Status\_Flags and Reliability Test

Purpose: To ensure Status\_Flags reflects the Reliability property value.

Test Concept: Write to Reliability and verify the interrelationship between the Status\_Flags and Reliability.

Configuration Requirements: The selected object is configured such that its Reliability is NO\_FAULT\_DETECTED before execution of this test. If the Reliability property is not present or not writable, then this test shall be skipped.

Test Steps:

1. VERIFY Reliability = NO\_FAULT\_DETECTED
2. VERIFY Status\_Flags = (?, FALSE, ?, FALSE)
3. REPEAT X = (all values of the Reliability enumeration appropriate to the object type except NO\_FAULT\_DETECTED) DO {  
    WRITE Reliability = X  
    VERIFY Reliability = X  
    VERIFY Status\_Flags = (TRUE, TRUE, ?, FALSE)  
    WRITE Reliability = NO\_FAULT\_DETECTED  
    VERIFY Reliability = NO\_FAULT\_DETECTED  
    VERIFY Status\_Flags = (? FALSE, ?, FALSE)  
}

### 7.3.2.X53.4 Requested\_Shed\_Level property test with PERCENT choice

Purpose: To verify the performance of a shed request with PERCENT choice.

Test Concept: The Requested\_Shed\_Level property of the Load Control object is set to a PERCENT choice, and it is verified that the series of required actions which that sets into operation occur correctly.

Test Steps: The test steps defined in test 7.3.2.X53.1 shall be followed except that the Requested\_Shed\_Level property of the Load Control object is written to a PERCENT choice, and the default value for a shed request with PERCENT choice in Requested\_Shed\_Level, Expected\_Shed\_Level, and Actual\_Shed\_Level properties is 100

### 7.3.2.X53.5 Requested\_Shed\_Level property test with AMOUNT choice

Purpose: To verify the performance of a shed request with AMOUNT choice.

Test Concept: The Requested\_Shed\_Level property of the Load Control object is set to an AMOUNT choice and it is verified that the series of required actions which that sets into operation occur correctly.

Test Steps: The test steps defined in test 7.3.2.X53.1 shall be followed except that the Requested\_Shed\_Level property of the Load Control object is written to an AMOUNT choice, and the default value for a shed request with AMOUNT choice in Requested\_Shed\_Level, Expected\_Shed\_Level, and Actual\_Shed\_Level properties is 0.0

[Add new **Clause 7.3.2.X54**, p. 185]

[Reviewer Note: Clause 7.3.2 (Object Specific Tests) contains no tests for the Lighting Output object type.]

## 7.3.2.X54 Lighting Output Object Tests

### 7.3.2.X54.21 Lighting Output Tracking Test

Purpose: To verify that the Tracking\_Value property follows the Present\_Value property.

Test Concept: Write to the Present\_Value of a Lighting Output object, O1, and verify that the Tracking\_Value property follows Present\_Value once In-Progress returns to IDLE.

Configuration Requirements: The IUT shall be configured with a lighting output, O1, that can be observed during the test. O1 shall be configured such that all slots in the Priority\_Array numerically less than PTY1 have a value of NULL and no internal algorithms are issuing commands to O1 at a priority numerically less than or equal to PTY1 and Out\_Of\_Service = FALSE.

Test Steps:

1. WRITE Present\_Value = 100, PRIORITY = PTY1
2. VERIFY Present\_Value = 100
3. WHILE (In\_Progress  $\neq$  IDLE) DO {  
}
4. VERIFY Tracking\_Value = 100
5. WRITE Present\_Value = 1, PRIORITY = PTY1
6. VERIFY Present\_Value = 1
7. WHILE (In\_Progress  $\neq$  IDLE) DO {  
}
8. VERIFY Tracking\_Value = 1
9. WRITE Present\_Value = 0, PRIORITY = PTY1
10. VERIFY Present\_Value = 0
11. WHILE (In\_Progress  $\neq$  IDLE) DO {  
}
12. VERIFY Tracking\_Value = 0

### 7.3.2.X54.22 Lighting Output Present Value between 0.0 and 1.0 Test

Purpose: To verify that writing a value numerically greater than 0.0 but less than 1.0 to Present\_Value shall result in Present\_Value taking on the value 1.0.

Test Concept: Select a value, V1, which is numerically greater than 0.0 and less than 1.0. Write V1 to Present\_Value and verify that Present\_Value takes on the value 1.0.

Configuration Requirements: The Lighting Output object, O1, shall be configured such that all slots in the Priority\_Array numerically less than PTY1 have a value of NULL and no internal algorithms are issuing commands to O1 at a priority numerically less than or equal to PTY1. Present\_Value shall be different from 1.0.

Test Steps:

1. VERIFY Present\_Value  $\neq$  1.0
2. WRITE Present\_Value = a value numerically greater than 0.0 but less than 1.0
3. VERIFY Present\_Value = 1.0

### 7.3.2.X54.31 Lighting Command Operation NONE Test

Purpose: To verify that the IUT can execute WriteProperty service requests when an attempt is made to write a value that is outside of the supported range.

Test Concept: The TD writes the Lighting Command Operation NONE to the IUT, and expects Error Class of PROPERTY and an Error Code of VALUE\_OUT\_OF\_RANGE

Test Steps:

1. VERIFY (Object1), P1 = (the value defined for this property in the EPICS);

2. TRANSMIT WriteProperty-Request,  
     'Object Identifier' = O1  
     'Property Identifier' = Lighting\_Command  
     'Property Value' = NONE
3. RECEIVE BACnet-Error PDU,  
     'Error Class' = PROPERTY,  
     'Error Code' = VALUE\_OUT\_OF\_RANGE
4. VERIFY (Object1), Lighting\_Command = (the value defined for this property in the EPICS)

### 7.3.2.X54.32 Lighting Command Operation FADE\_TO Test

Purpose: To verify the correct operation of FADE\_TO lighting command by observing the value of Present\_Value, In\_Progress and Tracking\_Value.

Test Concept: The TD writes to the Present\_Value at each end of the range (i.e., 0% or 100%), and then writes to the Lighting Command Operation with FADE\_TO with a long enough fade-time to allow In\_Progress and Tracking\_Value to be observed while set to FADE\_ACTIVE. The Tracking\_Value will be checked at the end of the fade to verify that it tracked the target level. The IUT shall be tested for fade up (0% to 100%) and fade down (100% to 0%).

Configuration Requirements: O1 shall be configured such that all slots in the Priority\_Array numerically less than PTY1 have a value of NULL and no internal algorithms are issuing commands to O1 at a priority numerically less than or equal to PTY1. V1 > 1 and V2 < 100%

Test Steps:

- Start with 0% Present\_Value to test fade up
  1. WRITE Present\_Value = 0, ARRAY INDEX = PTY1
  2. VERIFY Present\_Value = 0
  3. **WAIT Internal Processing Fail Time**
  4. VERIFY Tracking\_Value = 0
- Write a FADE\_TO command (operation, target-level, priority, fade-time)
  5. WRITE Lighting\_Command = (FADE\_TO, V1, PTY1, FT)
  6. **WAIT Internal Processing Fail Time**
  7. VERIFY Priority\_Array = V1, ARRAY INDEX = PTY1
  8. VERIFY Present\_Value = V1
- In a half way of fading up, check In\_Progress and Tracking\_Value
  9. WAIT FT/2
  10. VERIFY In\_Progress = FADE\_ACTIVE,
  11. VERIFY Tracking\_Value  $\approx$  V1 / 2
  12. WAIT FT/2
- When fading up is completed, check In\_Progress and Tracking\_Value
  13. VERIFY In\_Progress = IDLE
  14. VERIFY Tracking\_Value = V1
- Now repeat the test with 100% Present\_Value to test fade down
  15. WRITE Present\_Value = 100, ARRAY INDEX = PTY1
  16. VERIFY Present\_Value = 100
  17. **WAIT Internal Processing Fail Time**
  18. VERIFY Tracking\_Value = 100
- Write a FADE\_TO command (operation, target-level, priority, fade-time)
  19. WRITE Lighting\_Command = (FADE\_TO, V2, PTY1, FT)
  20. **WAIT Internal Processing Fail Time**
  21. VERIFY Priority\_Array = V2, ARRAY INDEX = PTY1
  22. VERIFY Present\_Value = V2

-- In a half way of fading down, check In\_Progress and Tracking\_Value

23. WAIT FT/2

24. VERIFY In\_Progress = FADE\_ACTIVE,

25. VERIFY Tracking\_Value  $\approx$  V1 / 2

26. WAIT FT/2

-- When fading down is completed, check In\_Progress and Tracking\_Value

27. VERIFY In\_Progress = IDLE

28. VERIFY Tracking\_Value = V2

### 7.3.2.X54.33 Lighting Command Operation RAMP\_TO Test

Purpose: To verify the correct operation of RAMP\_TO lighting command by observing the value of Present\_Value, In\_Progress and Tracking\_Value.

Test Concept: The TD writes to Present\_Value at each end of the range (i.e., 0% or 100%), and then writes to the Lighting Command Operation with RAMP\_TO with a slow enough ramp rate to allow In\_Progress and Tracking\_Value to be observed while set to RAMP\_ACTIVE. The Tracking\_Value will be checked at the end of the ramp to verify that it tracked the target level. The IUT shall be tested for ramp up (0% to 100%) and ramp down (100% to 0%).

Configuration Requirements: O1 shall be configured such that all slots in the Priority\_Array numerically less than PTY1 have a value of NULL and no internal algorithms are issuing commands to O1 at a priority numerically less than or equal to PTY1. V1 > 1 and V2 < 100%

Test Steps:

-- Start with 0% Present\_Value to test ramp up

1. WRITE Present\_Value = 0, ARRAY INDEX = PTY1

2. VERIFY Present\_Value = 0

3. WAIT **Internal Processing Fail Time**

4. VERIFY Tracking\_Value = 0

-- Write a RAMP\_TO command (operation, target-value, priority, ramp-rate)

5. WRITE Lighting\_Command = (RAMP\_TO, V1, PTY1, any valid rate)

6. WAIT **Internal Processing Fail Time**

7. VERIFY Priority\_Array = V1, ARRAY INDEX = PTY1

8. VERIFY Present\_Value = V1

-- Check In\_Progress while ramping up

9. VERIFY In\_Progress = RAMP\_ACTIVE

-- Make sure that Tracking\_Value increases with the ramp-rate

10. WHILE (In\_Progress  $\neq$  IDLE) DO {

11. VERIFY Tracking\_Value > 0 < V1

12. CHECK (Tracking\_Value is increasing with the ramp-rate)}

-- When ramping up is completed, check In\_Progress and Tracking\_Value

13. VERIFY In\_Progress = IDLE

14. VERIFY Tracking\_Value = V1

-- Now repeat the test with 100% Present\_Value to test ramp down

15. WRITE Present\_Value = 100, ARRAY INDEX = PTY1

16. VERIFY Present\_Value = 100

17. WAIT **Internal Processing Fail Time**

18. VERIFY Tracking\_Value = 100

-- Write a RAMP\_TO command (operation, target-value, priority, ramp-rate)

19. WRITE Lighting\_Command = (RAMP\_TO, V2, PTY1, any valid rate)

20. WAIT **Internal Processing Fail Time**

```

21. VERIFY Priority_Array = V2, ARRAY INDEX = PTY1
22. VERIFY Present_Value = V2

-- Check In_Progress while ramping up
23. VERIFY In_Progress = RAMP_ACTIVE,

-- Make sure that Tracking_Value decreases with the ramp-rate
24. WHILE (In_Progress <> RAMP_ACTIVE) DO {
25. VERIFY Tracking_Value < 0
26. VERIFY Tracking_Value > V2
27. CHECK (Tracking_Value is decreasing with the ramp-rate)}

-- Check In_Progress and Tracking_Value
28. VERIFY In_Progress = IDLE
29. VERIFY Tracking_Value = V2

```

### 7.3.2.X54.34 Lighting Command Operation STEP\_UP Test

Purpose: To verify the correct operation of STEP\_UP lighting command by observing the value of Present\_Value, In\_Progress and Tracking\_Value.

Test Concept: The TD writes to Present\_Value at 0%, and then writes to the Lighting Command Operation with STEP\_UP and any step increment. The Tracking\_Value shall remain at 0% to ignore the operation. Next, the TD writes to Present\_Value at 1%, and then writes to the Lighting Command Operation with STEP\_UP and a step increment greater than 99%, the Tracking\_Value shall be 100%. The TD writes to Present\_Value at 1%, and then writes to the Lighting Command Operation with STEP\_UP and a step increment less than 99%, the Tracking\_Value shall be 1% plus the step increment.

Configuration Requirements: O1 shall be configured such that all slots in the Priority\_Array numerically less than PTY1 have a value of NULL and no internal algorithms are issuing commands to O1 at a priority numerically less than or equal to PTY1.

Test Steps:

```

-- Start with 0% Present_Value
1. WRITE Present_Value = 0, ARRAY INDEX = PTY1
2. VERIFY Present_Value = 0
3. WAIT Internal Processing Fail Time
4. VERIFY Tracking_Value = 0

-- Write a STEP_UP command (operation, priority, step-increment)
5. WRITE Lighting_Command = (STEP_UP, PTY1, any valid value)
6. WAIT Internal Processing Fail Time

-- Confirm that the command was ignored since Tracking_Value was 0
7. VERIFY Priority_Array = 0, ARRAY INDEX = PTY1
8. VERIFY Present_Value = 0
9. VERIFY Tracking_Value = 0

-- Now test with Tracking_Value > 0
10. WRITE Present_Value = 1, ARRAY INDEX = PTY1
11. VERIFY Present_Value = 1
12. WAIT Internal Processing Fail Time
13. VERIFY Tracking_Value = 1

-- Keep stepping up while continuously checking Priority_Array, Present_Value and Tracking_Value
14. REPEAT X = (1 through (100 - step-increment) by step-increment) DO {
    WRITE Lighting_Command = (STEP_UP, PTY1, any valid value)
    WAIT Internal Processing Fail Time
    VERIFY Priority_Array = X + step-increment, ARRAY INDEX = PTY1
    VERIFY Present_Value = X + step-increment

```

```

    VERIFY Tracking_Value = X + step-increment
  }
-- Now step up one more time to confirm that the values will not exceed 100
15. WRITE Lighting_Command = (STEP_UP, PTY1, any valid value)
16. WAIT Internal Processing Fail Time
17. VERIFY Priority_Array = 100, ARRAY INDEX = PTY1
18. VERIFY Present_Value = 100
19. VERIFY Tracking_Value = 100

```

### 7.3.2.X54.35 Lighting Command Operation STEP\_DOWN Test

Purpose: To verify that writing this Lighting Command Operation is reflected in the Tracking\_Value, that writes resulting in a step below 1% are limited to 1%, and that this command is ignored if the Tracking\_Value is 0.0%.

Test Concept: The TD writes to Present\_Value at 0%, and then writes to the Lighting Command Operation with STEP\_DOWN and any step increment. The Tracking\_Value shall remain at 0%. The TD writes to Present\_Value at 100%, and then writes to the Lighting Command Operation with STEP\_DOWN and a step increment greater than 99%, the Tracking\_Value shall be 1%. The TD writes to Present\_Value at 100%, and then writes to the Lighting Command Operation with STEP\_DOWN and a step increment less than 99%, the Tracking\_Value shall be 100% minus the step increment.

Configuration Requirements: O1 shall be configured such that all slots in the Priority\_Array numerically less than PTY1 have a value of NULL and no internal algorithms are issuing commands to O1 at a priority numerically less than or equal to PTY1.

Test Steps:

```

-- Start with 0% Present_Value
1.  WRITE Present_Value = 0, ARRAY INDEX = PTY1
2.  VERIFY Present_Value = 0
3.  WAIT Internal Processing Fail Time
4.  VERIFY Tracking_Value = 0

-- Write a STEP_DOWN command (operation, priority, step-increment)
5.  WRITE Lighting_Command = (STEP_DOWN, PTY1, any valid value)
6.  WAIT Internal Processing Fail Time

-- Confirm that the command was ignored since Tracking_Value was 0
7.  VERIFY Priority_Array = 0, ARRAY INDEX = PTY1
8.  VERIFY Present_Value = 0
9.  VERIFY Tracking_Value = 0

-- Now test with Tracking_Value = 100
10. WRITE Present_Value = 100, ARRAY INDEX = PTY1
11. VERIFY Present_Value = 100
12. WAIT Internal Processing Fail Time
13. VERIFY Tracking_Value = 100

-- Keep stepping down while continuously checking Priority_Array, Present_Value and Tracking_Value
14. REPEAT X = (100 through (1 + step-increment) by step-increment) DO {
    WRITE Lighting_Command = (STEP_DOWN, PTY1, any valid value)
    WAIT Internal Processing Fail Time
    VERIFY Priority_Array = X - step-increment, ARRAY INDEX = PTY1
    VERIFY Present_Value = X - step-increment
    VERIFY Tracking_Value = X - step-increment
  }

-- Now step down one more time to confirm that the values will not go down below 1
15. WRITE Lighting_Command = (STEP_DOWN, PTY1, any valid value)
16. WAIT Internal Processing Fail Time
17. VERIFY Priority_Array = 1, ARRAY INDEX = PTY1

```

18. VERIFY Present\_Value = 1
19. VERIFY Tracking\_Value = 1

### 7.3.2.X54.36 Lighting Command Operation STEP\_ON Test

Purpose: To verify that writing this Lighting Command Operation is reflected in the Tracking\_Value, that this command will set the Tracking\_Value to 1% if the Tracking\_Value is 0.0%, and that it otherwise adheres to STEP\_UP.

Test Concept: The TD writes to Present\_Value at 0%, and then writes to the Lighting Command Operation with STEP\_UP and any step increment. The Tracking\_Value shall be 1%. The TD writes to Present\_Value at 1%, and then writes to the Lighting Command Operation with STEP\_UP and a step increment greater than 99%, the Tracking\_Value shall be 100%. The TD writes to Present\_Value at 1%, and then writes to the Lighting Command Operation with STEP\_UP and a step increment less than 99%, the Tracking\_Value shall be 1% plus the step increment.

Configuration Requirements: O1 shall be configured such that all slots in the Priority\_Array numerically less than PTY1 have a value of NULL and no internal algorithms are issuing commands to O1 at a priority numerically less than or equal to PTY1.

Test Steps:

- Start with 0% Present\_Value
  1. WRITE Present\_Value = 0, ARRAY INDEX = PTY1
  2. VERIFY Present\_Value = 0
  3. WAIT **Internal Processing Fail Time**
  4. VERIFY Tracking\_Value = 0
- Write a STEP\_ON command (operation, priority, step-increment)
  5. WRITE Lighting\_Command = (STEP\_ON, PTY1, any valid values)
  6. WAIT **Internal Processing Fail Time**
- Confirm that the Present\_Value and Tracking\_Value became 1
  7. VERIFY Priority\_Array = 1, ARRAY INDEX = PTY1
  8. VERIFY Present\_Value = 1
  9. VERIFY Tracking\_Value = 1
- Keep stepping on while continuously checking Priority\_Array, Present\_Value and Tracking\_Value
  10. REPEAT X = (1 through (100 – step-increment)) DO {
    - WRITE Lighting\_Command = (STEP\_ON, PTY1, any valid values)
    - WAIT **Internal Processing Fail Time**
    - VERIFY Priority\_Array = X + step-increment, ARRAY INDEX = PTY1
    - VERIFY Present\_Value = X + step-increment
    - VERIFY Tracking\_Value = X + step-increment}
- Now step on one more time to confirm that the values will not exceed 100
  11. WRITE Lighting\_Command = (STEP\_ON, PTY1, any valid values)
  12. WAIT **Internal Processing Fail Time**
  13. VERIFY Priority\_Array = 100, ARRAY INDEX = PTY1
  14. VERIFY Present\_Value = 100
  15. VERIFY Tracking\_Value = 100

### 7.3.2.X54.37 Lighting Command Operation STEP\_OFF Test

Purpose: To verify that writing this Lighting Command Operation is reflected in the Tracking\_Value, that writes resulting in a step below 1% are limited to 1%, and that this command is ignored if the Tracking\_Value is 0.0%.

Test Concept: The TD writes to Present\_Value at 0%, and then writes to the Lighting Command Operation with STEP\_DOWN and any step increment. The Tracking\_Value shall remain at 0%. The TD writes to Present\_Value at 100%, and then writes to the Lighting Command Operation with STEP\_DOWN and a step increment greater than 99%, the Tracking\_Value shall be 1%.

The TD writes to Present\_Value at 100%, and then writes to the Lighting Command Operation with STEP\_DOWN and a step increment less than 99%, the Tracking\_Value shall be 100% minus the step increment.

Configuration Requirements: O1 shall be configured such that all slots in the Priority\_Array numerically less than PTY1 have a value of NULL and no internal algorithms are issuing commands to O1 at a priority numerically less than or equal to PTY1.

Test Steps:

-- Start with 0% Present\_Value

1. WRITE Present\_Value = 0, ARRAY INDEX = PTY1
2. VERIFY Present\_Value = 0
3. **WAIT Internal Processing Fail Time**
4. VERIFY Tracking\_Value = 0

-- Write a STEP\_OFF command (operation, priority, step-increment)

5. WRITE Lighting\_Command = (STEP\_OFF, PTY1, step-increment)
6. **WAIT Internal Processing Fail Time**

-- Confirm that the command was ignored since Tracking\_Value was 0

7. VERIFY Priority\_Array = 0, ARRAY INDEX = PTY1
8. VERIFY Present\_Value = 0
9. VERIFY Tracking\_Value = 0

-- Now test with Tracking\_Value = 100

10. WRITE Present\_Value = 100, ARRAY INDEX = PTY1
11. VERIFY Present\_Value = 100
12. **WAIT Internal Processing Fail Time**
13. VERIFY Tracking\_Value = 100

-- Keep stepping off while continuously checking Priority\_Array, Present\_Value and Tracking\_Value

14. REPEAT X = (100 through (1 + step-increment)) DO {
  - WRITE Lighting\_Command = (STEP\_OFF, PTY1, step-increment)
  - WAIT Internal Processing Fail Time**
  - VERIFY Priority\_Array = X - step-increment, ARRAY INDEX = PTY1
  - VERIFY Present\_Value = X - step-increment
  - VERIFY Tracking\_Value = X - step-increment

-- Confirm that the Present\_Value and Tracking\_Value become 0 when STEP OFF command is executed while Tracking\_Value is 1

15. WRITE Lighting\_Command = (STEP\_OFF, PTY1, step-increment)
16. **WAIT Internal Processing Fail Time**
17. VERIFY Priority\_Array = 0, ARRAY INDEX = PTY1
18. VERIFY Present\_Value = 0
19. VERIFY Tracking\_Value = 0

### 7.3.2.X54.41 Transition None Test

Purpose: To verify that the Tracking\_Value property immediately follows the Present\_Value property if Transition is NONE.

Test Concept: Setup a Lighting Output object, O1, to use its complete supported value range. Set Present\_Value to the highest supported value, and then to the lowest supported value, verifying that there is no delay in the transitions.

Configuration Requirements: O1 shall be configured such that all slots in the Priority\_Array numerically less than PTY1 have a value of NULL and no internal algorithms are issuing commands to O1 at a priority numerically less than or equal to PTY1. If present, Min\_Actual\_Value shall be set to 1, and Max\_Actual\_Value shall be set to 100. Transition shall be set to NONE.

Test Steps:



1. VERIFY Transition = NONE
2. VERIFY In\_Progress = IDLE
3. WRITE Present\_Value = 100, ARRAY INDEX = PTY1
4. VERIFY In\_Progress = IDLE
5. VERIFY Tracking\_Value = 100
6. WRITE Present\_Value = 1, ARRAY INDEX = PTY1
7. VERIFY In\_Progress = IDLE
8. VERIFY Tracking\_Value = 1

### 7.3.2.X54.42 Transition Test

Purpose: To verify that the Lighting Output object transitions using the configured function and transitions at the configured speed when Transition is set to either FADE or RAMP.

Test Concept: Setup a Lighting Output object, O1, to use fading or ramping as the default transition method. Present\_Value is changed to V1 which is larger than the initial Present\_Value, V0, so that the output will fade or ramp up. Halfway through the process, verify that Tracking\_Value is approximately equal to the value halfway between V0 and V1. The physical output shall also be verified that it is fading or ramping from V0 to V1. When the process completes, verify that Tracking\_Value reached V1. Repeat the process fading or ramping down from V1 to V2.

Configuration Requirements: O1 shall be configured such that all slots in the Priority\_Array numerically less than PTY1 have a value of NULL and no internal algorithms are issuing commands to O1 at a priority numerically less than or equal to PTY1. The Transition property is set to FADE or RAMP, Present\_Value is V0 and In\_Progress is IDLE.

To test FADE functionality, T is FADE, A is FADE\_ACTIVE, W1 and W2 are (Default\_Fade\_Time / 2), and Default\_Fade\_Time is sufficiently large so as to allow the intermediate progress checks.

To Test RAMP functionality, T is RAMP, A is RAMP\_ACTIVE, W1 is  $((V1 - V0) / \text{Default\_Ramp\_Rate}) / 2$ , W2 is  $((V1 - V2) / \text{Default\_Ramp\_Rate}) / 2$ , and Default\_Ramp\_Rate is sufficiently small so as to allow the intermediate progress checks.

Test Steps:

1. VERIFY Transition = T
2. VERIFY In\_Progress = IDLE
3. V0 = READ Present\_Value
4. WRITE Present\_Value = V1, ARRAY INDEX = PTY1
5. VERIFY Present\_Value = V1
6. WAIT W1
7. VERIFY Tracking\_Value  $\approx (V1 + V0) / 2$
8. VERIFY In\_Progress = A
9. CHECK (the physical output is fading from V0 to V1 )
10. WAIT W1
11. VERIFY In\_Progress = IDLE
12. VERIFY Tracking\_Value = V1
13. WRITE Present\_Value = V2, ARRAY INDEX = PTY1
14. VERIFY Present\_Value = V2
15. WAIT W2
16. VERIFY Tracking\_Value  $\approx (V2 + V1) / 2$
17. VERIFY In\_Progress = A
18. CHECK (the physical output is fading V1 to V2 )
19. WAIT W2
20. VERIFY In\_Progress = IDLE
21. VERIFY Tracking\_Value = V2

### 7.3.2.X54.51 Feedback\_Value Clamping Test

Purpose: To verify that the Feedback\_Value remains in the normalized range when the physical lighting output is outside the normalized range.

Test Concept: Set the normalized range to be the largest range supported by the device. Make the physical output be above the normalized range by setting it to the maximum supported value and then shrinking the normalized range. The Feedback\_Value is immediately tested to verify that it takes on the value 100.

Reset the normalized range. Make the physical output be below the normalized range by setting it to the minimum supported value and then shrinking the normalized range. The Feedback\_Value is immediately tested to verify that it takes on the value 1.

Configuration Requirements: The Lighting Output object, O1, shall be configured to transition slowly when Present\_Value changes, such as by ramping, fading or stepping, if possible.

O1 shall be configured such that all slots in the Priority\_Array numerically less than PTY1 have a value of NULL and no internal algorithms are issuing commands to O1 at a priority numerically less than or equal to PTY1.

Test Steps:

-- Verify Feedback\_Value when output is above Max\_Actual\_Value

1. WRITE Max\_Actual\_Value = 100
2. WRITE Min\_Actual\_Value = 1
3. WRITE Present\_Value = 100, PRIORITY = PTY1
4. WHILE In\_Progress <> IDLE {}
5. WRITE Max\_Actual\_Value = (Lowest supported Max\_Actual\_Value)
6. VERIFY Feedback\_Value = 100

-- Verify Feedback\_Value when output is below Min\_Actual\_Value

7. WRITE Max\_Actual\_Value = 100
8. WRITE Min\_Actual\_Value = 1
9. WRITE Present\_Value = 1, PRIORITY = PTY1
10. WHILE In\_Progress <> IDLE {}
11. WRITE Min\_Actual\_Value = (Highest supported Min\_Actual\_Value)
12. VERIFY Feedback\_Value = 1

### 7.3.2.X54.61 Min\_Actual\_Value and Max\_Actual\_Value Test

Purpose: To verify that Min\_Actual\_Value remains less than Max\_Actual\_Value and within the allowable range when either is written to a value that would violate these conditions.

Test Concept: Write a value to Min\_Actual\_Value which is larger than Max\_Actual\_Value. Verify that Max\_Actual\_Value became equal to Min\_Actual\_Value. Next, write a value to Max\_Actual\_Value which is less than Min\_Actual\_Value. Verify that Min\_Actual\_Value became equal to Max\_Actual\_Value.

Verify that neither Min\_Actual\_Value nor Max\_Actual\_Value will accept a value outside the range 1.0 to 100.0.

Configuration Requirements: The IUT shall be configured with a lighting output, O1. Min\_Actual\_Value shall be set to a value less than Max\_Actual\_Value, and Max\_Actual\_Value shall be within the allowable range for Min\_Actual\_Value and not equal to Min\_Actual\_Value's maximum supported value. If the IUT cannot be configured to meet these requirements, then this test shall be skipped.

Test Steps:

1. V1 = READ Max\_Actual\_Value

2. WRITE Min\_Actual\_Value = V2, a value greater than V1
3. VERIFY Max\_Actual\_Value = V2
4. WRITE Max\_Actual\_Value = V3, a value less than V2
5. VERIFY Min\_Actual\_Value = V3
6. TRANSMIT WritePropertyRequest  
     'Object Identifier' = O1,  
     'Property Identifier' = Min\_Actual\_Value,  
     'Property Value' = (any value outside the range 1.0 to 100.0)
7. RECEIVE BACnet-Error-PDU,  
     'Error Class' = PROPERTY,  
     'Error Code' = VALUE\_OUT\_OF\_RANGE
8. TRANSMIT WritePropertyRequest  
     'Object Identifier' = O1,  
     'Property Identifier' = Max\_Actual\_Value,  
     'Property Value' = (any value outside the range 1.0 to 100.0)
9. RECEIVE BACnet-Error-PDU,  
     'Error Class' = PROPERTY,  
     'Error Code' = VALUE\_OUT\_OF\_RANGE

### 7.3.2.X54.62 Min\_Actual\_Value and Max\_Actual\_Value Scaling Test

Purpose: To verify that the physical output level changes to the expected scaled value as Present\_Value changes.

Test Concept: Set Min\_Actual\_Value to a value other than the lowest supported minimum value, and set Max\_Actual\_Value to a value other than the highest support value but larger than Min\_Actual\_Value.

Then write 1.0 to Present\_Value and measure the physical output. Repeat the procedure to measure the physical output after writing 100.0 to Present\_Value. After obtaining these upper and lower bound values, write a value between 1.0 and 100.0, measure the physical output, and confirm that the measured value is approximately the same as the expected scaled value.

Configuration Requirements: The IUT shall be configured with a lighting output, O1 that can be observed during the test. O1 shall be configured such that all slots in the Priority\_Array numerically less than PTY1 have a value of NULL and no internal algorithms are issuing commands to O1 at a priority numerically less than or equal to PTY1 and Out\_Of\_Service = FALSE.

Test Steps:

1. WRITE Min\_Actual\_Value = (a supported value that is not the lowest supported value)
2. WRITE Max\_Actual\_Value = (a supported value which is not the highest support value)
3. WRITE Present\_Value = 1.0, ARRAY INDEX = PTY1
4. CHECK(the value of the physical output is Min\_Actual\_Value)
5. WRITE Present\_Value = 100.0, ARRAY INDEX = PTY1
6. CHECK(the value of the physical output is Max\_Actual\_Value)
7. WRITE Present\_Value = (V1, a value between 1.0 and 100.0 exclusive), ARRAY INDEX = PTY1
8. MAKE(measure the value of the physical output and record in MV)
9. CHECK ( $MV \approx \text{Min\_Actual\_Value} + (V1 / 100) * (\text{Max\_Actual\_Value} - \text{Min\_Actual\_Value})$ )

[Add new **Clause 7.3.2.X55**, p. 185]

[Reviewer Note: Clause 7.3.2 (Object Specific Tests) contains no tests for the Access Door object type.]

### 7.3.2.X55 Access Door Object Tests

#### 7.3.2.X55.1 Commandable Present\_Value Test

Purpose: To verify that writing to the Present\_Value will cause a corresponding change to the physical output.

Test Concept: The IUT shall be configured with a door control output that can be observed during the test. The Present\_Value property is written with each of the following values: UNLOCK, LOCK, PULSE\_UNLOCK, EXTENDED\_PULSE\_UNLOCK, and the Access Door object is monitored to ensure that the door locks and unlocks appropriately.

Configuration Requirements: The Relinquish\_Default shall have the value LOCK. All writes are at a priority higher than any internal algorithms writing to this property. Out\_Of\_Service shall be set to FALSE. Prior to the test the Present\_Value shall have the value LOCK and the IUT is in a state that would cause the door to be locked.

Test Steps:

-- Test UNLOCK value

1. WRITE Present\_Value = UNLOCK
2. WAIT (**Internal Processing Fail Time**)
3. IF (Lock\_Status is present) THEN  
    VERIFY Lock\_Status = UNLOCKED
4. CHECK (that the door control output is in a state that would cause the door to be unlocked)

-- Test LOCK value

5. WRITE Present\_Value = LOCK
6. WAIT (**Internal Processing Fail Time**)
7. IF (Lock\_Status is present) THEN  
    VERIFY Lock\_Status = LOCKED
8. CHECK (that the door control output is in a state that would cause the door to be locked)

-- Test PULSE\_UNLOCK value

9. WRITE Present\_Value = PULSE\_UNLOCK
10. WAIT (**Internal Processing Fail Time** + Door\_Unlock\_Delay\_Time if present)
11. IF (Lock\_Status is present) THEN  
    VERIFY Lock\_Status = UNLOCKED
12. CHECK (that the IUT is in a state that would cause the door to be unlocked)
13. WAIT (Door\_Pulse\_Time)
14. VERIFY Present\_Value = LOCK
15. IF (Lock\_Status is present) THEN  
    VERIFY Lock\_Status = LOCKED
16. CHECK (that the door control output is in a state that would cause the door to be locked)

-- Test EXTENDED\_PULSE\_UNLOCK value

17. WRITE Present\_Value = EXTENDED\_PULSE\_UNLOCK
18. WAIT (**Internal Processing Fail Time** + Door\_Unlock\_Delay\_Time if present)

### 7.3.2.X55.2 Door\_Status, Lock\_Status and Door\_Alarm\_State Tests

Purpose: This test case verifies that Door\_Status, Lock\_Status, and Door\_Alarm\_State properties are writable when Out\_Of\_Service is TRUE.

Test Concept: Set Out\_Of\_Service to TRUE and then make sure one at a time that Door\_Status, Lock\_Status, and Door\_Alarm\_State, if present, are writable.

Configuration Requirements: If the Out\_Of\_Service property of this object is not writable, and if the Out\_Of\_Service property cannot be changed by other means, then this test shall be omitted. All writes to the Present\_Value shall be performed at a priority higher (numerically smaller) than any internal algorithms writing to this property. For testing Door\_Alarm\_State, test only values listed in either the Alarm\_Values or Fault\_Values.

Test Steps:

1. MAKE (Out\_Of\_Service TRUE)

2. VERIFY Status\_Flags = (?, ?, ?, TRUE)
3. IF (Door\_Status is present) THEN
  - REPEAT X = (all values of the Door\_Status enumeration values supported by the property)
  - DO {
    - WRITE Door\_Status = X
    - VERIFY Door\_Status = X
4. IF (Lock\_Status is present) THEN
  - REPEAT X = (all values of the Lock\_Status enumeration values supported by the property)
  - DO {
    - WRITE Lock\_Status = X
    - VERIFY Lock\_Status = X
5. IF (Door\_Alarm\_State is present) THEN
  - REPEAT X = (all values of the Door\_Alarm\_State enumeration values supported by the property)
  - DO {
    - WRITE Door\_Alarm\_State = X
    - VERIFY Door\_Alarm\_State = X

### 7.3.2.X55.3 Door\_Status with Physical Door Status Tests

Purpose: To verify that the Door\_Status property reflects the state of the physical door (CLOSED, OPENED, UNUSED, and DOOR\_FAULT if the object supports detecting door faults).

Test Concept: The IUT is configured to monitor the state of a physical door. The physical door may be represented by a BACnet input object or through some proprietary method.

Configuration Requirements: The IUT shall be configured such that it can determine the state of a door. The Access\_Door object associated with this physical door shall be configured with Out\_Of\_Service = FALSE.

Test Steps:

1. MAKE (set physical door to the closed state)
2. VERIFY Door\_Status = CLOSED
3. MAKE (set physical door to the opened state)
4. VERIFY Door\_Status = OPENED
5. IF (the object supports detecting door faults) THEN
  - MAKE (set the physical door to a state that would cause the Door\_Status to take on a value of DOOR\_FAULT)
  - VERIFY Door\_Status = DOOR\_FAULT
6. IF (possible to remove a door status input associated with the door) THEN
  - MAKE (remove a door status input associated with the door)
7. VERIFY Door\_Status = UNUSED | UNKNOWN

### 7.3.2.X55.4 Lock\_Status Tests

Purpose: To verify that the Lock\_Status property reflects the state of the physical lock. (LOCKED, UNLOCKED, and LOCK\_FAULT if the object supports detecting lock faults).

Test Concept: The IUT monitors the state of a physical lock. The state of the physical lock may be represented by a BACnet input object or through some proprietary method.

Configuration Requirements: The IUT shall be configured such that it can monitor the state of the physical lock. The Access\_Door object associated with this physical door shall be configured with Out\_Of\_Service = FALSE. The physical lock shall be manipulated other than through the Access Door object.

Notes to tester: The physical lock shall be manipulated other than through the Access Door object.

#### Test Steps:

1. MAKE (set the physical lock to a state that would cause the Lock\_Status to take on a value of LOCKED)
2. VERIFY Lock\_Status = LOCKED
3. MAKE (set the physical lock to a state that would cause the Lock\_Status to take on a value of UNLOCKED)
4. VERIFY Lock\_Status = UNLOCKED
5. IF (the object and the lock support detecting lock faults) THEN
  - MAKE (set the physical lock to a state that would cause the Lock\_Status to take on a value of LOCK\_FAULT)
  - VERIFY Lock\_Status = LOCK\_FAULT

### 7.3.2.X55.5 Secured\_Status Tests

Purpose: To verify that the Secured\_Status property reflects the state of the physical lock, the physical door, and the state of the Access Door object.

Test Concept: Start the test by creating a condition where the Secured\_Status = SECURED. Then create various conditions one at a time to verify that the Secured\_Status becomes UNSECURED when it should.

Configuration Requirements: All writes to the Present\_Value shall be performed at a priority higher than any internal algorithms writing to this property. If this object supports intrinsic reporting, then the Alarm\_Values property shall be empty. If this object supports the Masked\_Alarm\_Values property, then it shall be empty. Out\_Of\_Service is FALSE.

#### Test Steps:

- Create a condition where the Secured\_Status becomes SECURED
  1. WRITE Present\_Value = LOCK
  2. WAIT (**Internal Processing Fail Time**)
  3. VERIFY Status\_Flags = (FALSE ?, ?, ?)
  4. IF (Lock\_Status property is present) THEN
    - MAKE (Lock\_Status = LOCKED or UNUSED)
  5. MAKE (Door\_Status = CLOSED or UNUSED)
- Verify that the Secured\_Status is SECURED when it should
  6. VERIFY Secured\_Status = SECURED
- Verify that Secured\_Status is UNSECURED when Present\_Value is anything other than LOCKED
  7. REPEAT X = (UNLOCK, PULSE\_UNLOCK, EXTENDED\_PULSE\_UNLOCK) DO {
    - WRITE Present\_Value = X
    - WAIT (**Internal Processing Fail Time**)
    - VERIFY Secured\_Status = UNSECURED
- Recreate a condition where the Secured\_Status becomes SECURED again
  8. WRITE Present\_Value = LOCK
  9. WAIT (**Internal Processing Fail Time**)
  10. VERIFY Secured\_Status = SECURED
- Verify that Secured\_Status is UNSECURED when Masked\_Alarm\_Value, if exist, is NOT empty
  11. IF (Masked\_Alarm\_Values is present) THEN
    - MAKE (Masked\_Alarm\_Values = (any valid BACnetDoorAlarmState enumeration))
    - WAIT(**Internal Processing Fail Time**)
    - VERIFY Secured\_Status = UNSECURED
- Recreate a condition where the Secured\_Status becomes SECURED again
  - MAKE (Masked\_Alarm\_Values = {})
  - WAIT (**Internal Processing Fail Time**)

VERIFY Secured\_Status = SECURED

-- Verify that Secured\_Status is UNSECURED when Lock\_Status, if present, is anything other than LOCKED or UNUSED

```
12. IF (Lock_Status property is present) THEN
    REPEAT X = (UNLOCKED, UNKNOWN, LOCK_FAULT) DO {
        MAKE (Lock_Status = X)
        WAIT (Internal Processing Fail Time)
        VERIFY Secured_Status = UNSECURED
    }
    REPEAT X = (LOCKED, UNUSED) DO {
        MAKE (Lock_Status = X)
        VERIFY Secured_Status = SECURED
    }
}
```

-- Verify that Secured\_Status is UNSECURED when Door\_Status, is anything other than CLOSED or UNUSED

```
13. REPEAT X = (OPEN, UNKNOWN, DOOR_FAULT) DO {
    MAKE (Door_Status = X)
    WAIT (Internal Processing Fail Time)
    VERIFY Secured_Status = UNSECURED
}
REPEAT X = (CLOSED, UNUSED) DO {
    MAKE (Door_Status = X)
    WAIT (Internal Processing Fail Time)
    VERIFY Secured_Status = SECURED
}
```

-- Verify that Secured\_Status is UNSECURED when In\_Alarm bit of Status\_Flag is True

```
14. IF (Alarming is supported) THEN
    IF (Alarm_Values is writable) THEN
        WRITE Alarm_Values = { AV: any valid value}
        MAKE (trigger an alarm by using a physical door/lock to create the door alarm state AV)
        WAIT (Internal Processing Fail Time + Time_Delay)
        VERIFY Status_Flags = (TRUE, FALSE, ?, ?)
        VERIFY Secured_Status = UNSECURED
    }
```

### 7.3.2.X55.6 Door\_Unlock\_Delay\_Time Test

Purpose: To verify that when the Door\_Unlock\_Delay\_Time property has a non-zero value, the output is delayed in unlocking when a PULSE\_UNLOCK or EXTENDED\_PULSE\_UNLOCK is written to the Present\_Value and not when UNLOCK is written.

Test Concept: When unlocking the door by writing PULSE\_UNLOCK to the Present\_Value of the Access Door object, it is verified that the door is still locked for the specified Door\_Pulse\_Time, then the door is unlocked. The same test is done for EXTENDED\_PULSE\_UNLOCK, but this time, it is verified that the door is still locked for the specified Door\_Extended\_Pulse\_Time then the door is unlocked.

Configuration Requirements: The IUT shall be configured with a door control output that can be observed during the test. The Relinquish\_Default shall have the value LOCK. All writes to the Present\_Value shall be performed at a priority higher than any internal algorithms writing to this property. Door\_Unlock\_Delay\_Time shall be set to a non-zero value which is sufficient to observe the delay and check the status of the lock. Out\_Of\_Service shall be set to FALSE. Prior to the test the Present\_Value shall have the value LOCK and the IUT is in a state that would cause the door to be locked.

Test Steps:

```
-- Test PULSE_UNLOCK
1. WRITE Present_Value = PULSE_UNLOCK
2. WAIT (Internal Processing Fail Time)
3. BEFORE Door_Unlock_Delay_Time
    IF (Lock_Status is present) THEN
```

VERIFY Lock\_Status = LOCKED  
CHECK (that the door control output is in a state that would cause the door to be locked)

4. IF (Lock\_Status is present) THEN  
    VERIFY Lock\_Status = UNLOCKED
5. CHECK (that the door control output is in a state that would cause the door to be unlocked)
6. WAIT (Door\_Pulse\_Time)
7. VERIFY Present\_Value = LOCK
8. IF (Lock\_Status is present) THEN  
    VERIFY Lock\_Status = LOCKED
9. CHECK (that the door control output is in a state that would cause the door to be locked)

-- Test EXTENDED\_PULSE\_UNLOCK

10. WRITE Present\_Value = EXTENDED\_PULSE\_UNLOCK
11. WAIT (**Internal Processing Fail Time**)
12. BEFORE Door\_Unlock\_Delay\_Time  
    IF (Lock\_Status is present) THEN  
        VERIFY Lock\_Status = LOCKED  
    CHECK (that the door control output is in a state that would cause the door to be locked)
13. IF (Lock\_Status is present) THEN  
    VERIFY Lock\_Status = UNLOCKED
14. CHECK (that the door control output is in a state that would cause the door to be unlocked)
15. WAIT (Door\_Extended\_Pulse\_Time)
16. VERIFY Present\_Value = LOCK
17. IF (Lock\_Status is present) THEN  
    VERIFY Lock\_Status = LOCKED
18. CHECK (that the door control output is in a state that would cause the door to be locked)

-- Test UNLOCK

19. WRITE Present\_Value = UNLOCK
20. WAIT (**Internal Processing Fail Time**)
21. IF (Lock\_Status is present) THEN  
    VERIFY Lock\_Status = UNLOCKED
22. CHECK (that the door control output is in a state that would cause the door to be locked)

### 7.3.2.X55.7 Masked\_Alarm\_Values Tests

Purpose: To verify that the Masked\_Alarm\_Values prevents an intrinsic alarm from occurring.

Test Concept: The Access Door is verified to be in an Out\_Of\_Service state and is not in an alarm state. Then a non-NORMAL enumeration value of BACnetDoorAlarmState X is written to the Door\_Alarm\_State, and the Access Door object transitions to an alarm state. X is written to the Masked\_Alarm\_Value, and Door\_Alarm\_State is checked to verify it returned to NORMAL. The sequence is repeated for all non-NORMAL enumeration values of BACnetDoorAlarmState.

Configuration Requirements: The Masked\_Alarm\_Values list shall be empty at the start of this test. Out\_Of\_Service shall be set to TRUE to allow writing to the Door\_Alarm\_State property. If Out\_Of\_Service is not writeable and cannot be set to TRUE by any other means, this test shall be skipped. The enumeration BACnetDoorAlarmState value X to be used in the test has to be present in either the Alarm\_Values or Fault\_Values property.

Test Steps:

1. VERIFY Status\_Flags = (FALSE ?, ?, TRUE)
2. VERIFY Door\_Alarm\_State = NORMAL
3. REPEAT X = (all valid values of the enumeration BACnetDoorAlarmState except NORMAL)  
    DO {  
        WRITE Door\_Alarm\_State = X  
        WAIT (**Internal Processing Fail Time**)  
        VERIFY Status\_Flags = (TRUE ?, ?, TRUE)



```

WRITE Masked_Alarm_Values= { X }
WAIT (Internal Processing Fail Time)
VERIFY Door_Alarm_State = NORMAL
VERIFY Status_Flags = (FALSE ?, ?, TRUE)
WRITE Masked_Alarm_Values= { }
WAIT (Internal Processing Fail Time)
}

```

### 7.3.2.X55.8 Door\_Open\_Too\_Long Test

Purpose: To verify that the DOOR\_OPEN\_TOO\_LONG condition is generated when the Access Door object is commanded to the LOCK state, but the physical door remains open beyond Door\_Open\_Too\_Long\_Time.

Test Concept: Setup the Access Door object to trigger alarm on DOOR\_OPEN\_TOO\_LONG state using Alarm\_Values and Masked\_Alarm\_Values. Next, set the physical door to the closed state to confirm that the Access Door object is in NORMAL state. Then, unlock the physical door and set the physical door to the open state. Finally, command the Access Door object to LOCK and verify that the Door\_Alarm\_State changes to DOOR\_OPEN\_TOO\_LONG after the specified Time\_Delay.

Configuration Requirements: This test shall be skipped if the IUT does not support intrinsic alarming. The IUT shall be configured such that it can determine and change the open/closed state of a door. All writes to the Present\_Value are at a priority higher than any internal algorithms writing to this property. The Door\_Alarm\_State shall have the value NORMAL at the start of the test. The Access Door object is configured with DOOR\_OPEN\_TOO\_LONG in the Alarm\_Values property and excluded from Masked\_Alarm\_Values property if present.

Test Steps:

1. MAKE (set the physical door to the closed state)
2. VERIFY Door\_Alarm\_State = NORMAL
3. WRITE Present\_Value = UNLOCK
4. MAKE (set the physical door to the open state)
5. WRITE Present\_Value = LOCK
6. WAIT (**Internal Processing Fail Time**)
7. WHILE (Door\_Open\_Too\_Long\_Time has not expired) DO {
  - VERIFY Door\_Alarm\_State = NORMAL
 }
 WAIT (Time\_Delay)
8. VERIFY Door\_Alarm\_State = DOOR\_OPEN\_TOO\_LONG

[Add new **Clause 7.3.2.X56**, p. 185]

[Reviewer Note: Clause 7.3.2 (Object Specific Tests) contains no tests for the Access Point object type.]

### 7.3.2.X56 Access Point Object Tests

The Access Point object type represents the external interface of the access control decision engine for a specific door. A credential is entered, the access rights of the credential are determined, and the access decision is determined based on the access rights. Testing this authentication and authorization functionality requires the support of other standard BACnet access control object types. The required and optional object types are shown in figure X1.

The access decision begins with a credential value being sent to the Access Point object for evaluation. Typically, the credential value is read at a Credential Data Input object which extracts the raw credential data from the physical reader, formats the data and then sends it to the corresponding Access Point object. If the Credential Data Input object type is not supported, then the vendor must provide an alternate method for the credential to be received by the Access Point.

When a credential value is received by the Access Point object it searches through the Access Credential objects to find the one with a matching credential value. For each credential value being tested a corresponding Access Credential must exist within the IUT. The only exception to this is when testing for an unknown credential (DENIED\_UNKNOWN\_CREDENTIAL).

To determine if access is granted or denied for a specific credential each Access Credential object must reference an Access Rights object which defines the appropriate access rights corresponding to the specific test being executed.

Some of the allowed and denied access tests require the Access Zone object. In this case an Access Point must be configured to be an entry access point to the access zone. These tests require that the Access Rights objects reference the Access Zone rather than the Access Point.

When the access decision is determined, the IUT shall provide a method to indicate the result. Typically, the decision is exposed through the Access Door object. When access is granted, the door is pulsed unlocked and when denied, the door remains locked. If the Access Door object is not used, then another method of showing the result shall be configured.

Unless specified otherwise in the specific test, the access point object (AP1) in the following tests shall have the following configuration:

- Authorization\_Mode shall have the value AUTHORIZE.
- Out\_Of\_Service shall be FALSE.
- Lockout shall be FALSE.
- Muster\_Point shall be FALSE.
- Authentication\_Status shall be READY.

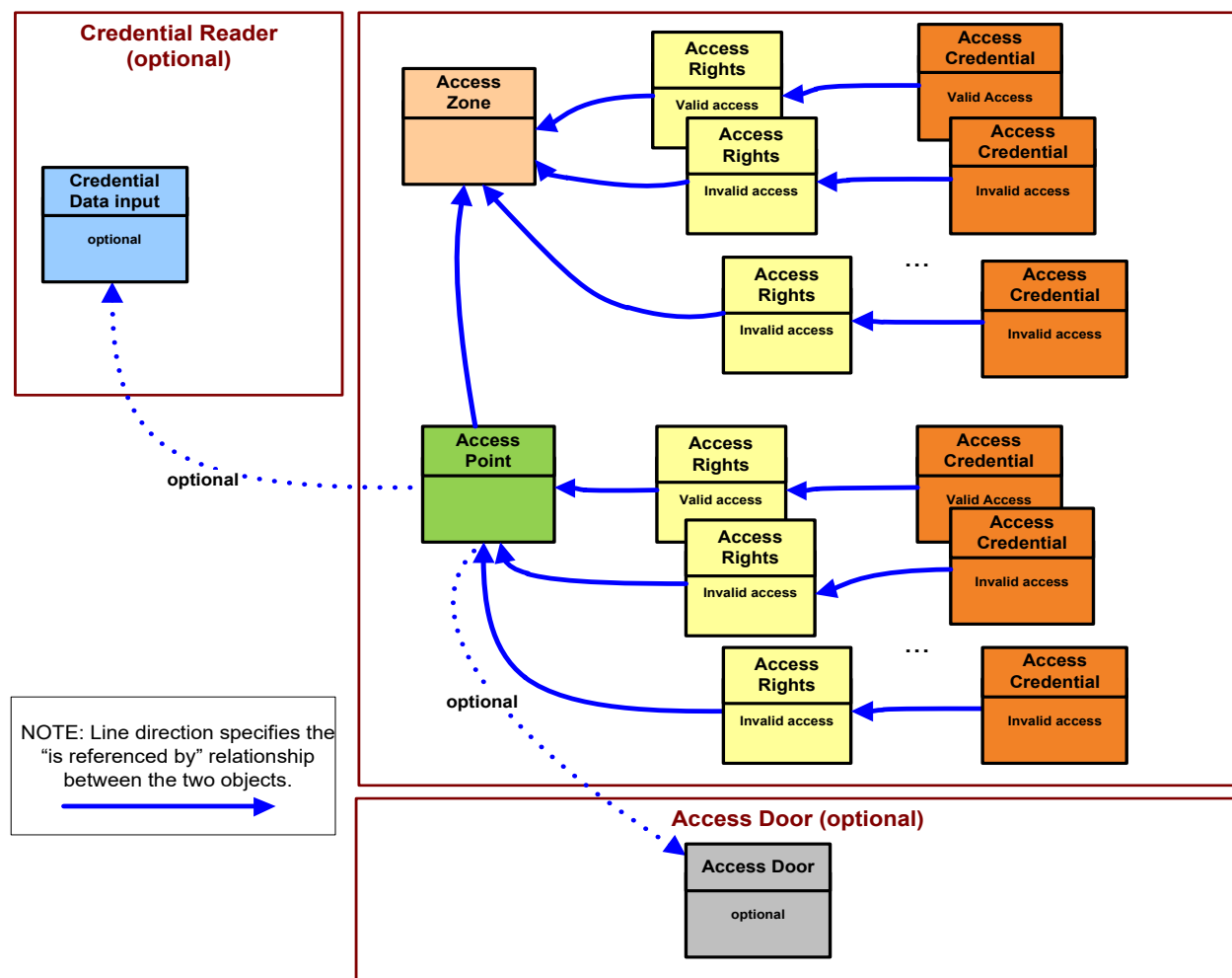


Figure X1: Objects and relationships used for testing Access Point objects

### 7.3.2.X56.1 Authentication\_Status and Access\_Event Test

Purpose: This test case verifies the authentication and authorization process are disabled when Out\_Of\_Service of the Access Point object is TRUE. It also verifies the interrelationship between the Out\_Of\_Service, Authentication\_Status, Access\_Event, and Access\_Event\_Time properties.

Test Concept: Write TRUE to the Out\_Of\_Service property and verify that the authorization and authentication functions are disabled. Write FALSE to the Out\_Of\_Service property and verify that they are enabled.

Configuration Requirements: See 7.3.2.X56

Test Steps:

1. VERIFY Authentication\_Status = READY
2. IF (Out\_Of\_Service is writable) THEN  
    WRITE Out\_Of\_Service = TRUE  
ELSE  
    MAKE (Out\_Of\_Service TRUE)
3. VERIFY Authentication\_Status = DISABLED
4. VERIFY Access\_Event = OUT\_OF\_SERVICE
5. VERIFY Access\_Event\_Time = (the time Out\_Of\_Service was set to TRUE)
6. IF (Out\_Of\_Service is writable) THEN  
    WRITE Out\_Of\_Service = FALSE  
ELSE  
    MAKE (Out\_Of\_Service FALSE)
7. VERIFY Authentication\_Status = READY
8. VERIFY Access\_Event = OUT\_OF\_SERVICE\_RELINQUISHED
9. VERIFY Access\_Event\_Time = (the time Out\_Of\_Service was set to FALSE)

### 7.3.2.X56.2 Allowed Access Test

Purpose: To verify that a valid credential that is allowed access to this access point at this time is granted access.

Test Concept: A valid credential, that has access to the access point being tested at the current time, is presented at the access point. It is then verified that access is allowed, and the appropriate access event is generated.

Configuration Requirements: See 7.3.2.X56. This test requires the following additional configuration: An active credential with valid access rights for the access point shall be represented by Access Credential object C1.

Test Steps:

1. READ EventTag = Access\_Event\_Tag
2. MAKE (present a valid credential at credential reader for this access point)
3. VERIFY Access\_Event = GRANTED
4. VERIFY Access\_Event\_Time = (the time that the credential was presented)
5. VERIFY Access\_Event\_Credential = C1
6. VERIFY Access\_Event\_Tag = EventTag + 1

### 7.3.2.X56.3 Denied Access Test

Purpose: To verify that a credential that is not allowed access to this access point at this time is denied access. There are a number of reasons why a credential may be denied access and this test tests the situations which must be supported by the access point.

Test Concept: To test that a credential, which is not allowed access to this access point, is presented at the access point with the result that access is allowed and the appropriate access event is generated.

#### Configuration Requirements:

See 7.3.2.X56. This test requires the following additional configuration:

- a) The vendor shall provide a set of credentials which correspond to Access Credential objects configured such that access to the access point shall be denied for the following reasons:
  - a. DENIED\_POINT\_NO\_ACCESS\_RIGHTS = C1
  - b. DENIED\_NO\_ACCESS\_RIGHTS = C2
  - c. DENIED\_ZONE\_NO\_ACCESS\_RIGHTS = C3
  - d. DENIED\_CREDENTIAL\_NOT\_YET\_ACTIVE = C4
  - e. DENIED\_CREDENTIAL\_EXPIRED = C5
  - f. DENIED\_CREDENTIAL\_MANUAL\_DISABLE = C6
  - g. DENIED\_INCORRECT\_AUTHENTICATION\_FACTOR = C7
  - h. DENIED\_OUT\_OF\_TIME\_RANGE = C8
  - i. DENIED\_THREAT\_LEVEL = C9
  - j. DENIED\_PASSBACK = C10
  - k. DENIED\_UNEXPECTED\_LOCATION\_USAGE = C11
  - l. DENIED\_MAX\_ATTEMPTS = C12
  - m. DENIED\_AUTHENTICATION\_FACTOR\_LOST = C13
  - n. DENIED\_AUTHENTICATION\_FACTOR\_STOLEN = C14
  - o. DENIED\_AUTHENTICATION\_FACTOR\_DAMAGED = C15
  - p. DENIED\_AUTHENTICATION\_FACTOR\_DESTROYED = C16
  - q. DENIED\_AUTHENTICATION\_FACTOR\_DISABLED = C17
  - r. DENIED\_AUTHENTICATION\_FACTOR\_ERROR = C18
  - s. DENIED\_CREDENTIAL\_UNASSIGNED = C19
  - t. DENIED\_CREDENTIAL\_NOT\_PROVISIONED = C20
  - u. DENIED\_CREDENTIAL\_LOCKOUT = C21
  - v. DENIED\_CREDENTIAL\_MAX\_DAYS = C22
  - w. DENIED\_CREDENTIAL\_MAX\_USES = C23
  - x. DENIED\_CREDENTIAL\_DISABLED = C24
  - y. DENIED\_LOCKOUT = C25

Notes to Tester: If the IUT does not support any of the above denial reasons, then the corresponding credentials are not required to be supplied.

#### Test Steps:

```

1. REPEAT C = (C1...C25) DO {
  READ EventTag = Access_Event_Tag
  MAKE (present the credential corresponding to C at the credential reader for this access point)
  VERIFY Access_Event = (denied reason corresponding to credential C)
  VERIFY Access_Event_Time = (the time that credential C was presented)
  VERIFY Access_Event_Credential = C
  VERIFY Access_Event_Tag = EventTag + 1
}
```

-- verify unknown credential event

```

2. READ EventTag = Access_Event_Tag
3. MAKE (present a credential which does not correspond to any configured Access Credential object at the
   credential reader for this access point)
4. VERIFY Access_Event = DENIED_UNKNOWN_CREDENTIAL
5. VERIFY Access_Event_Time = (the time that the credential was presented)
6. VERIFY (Access_Event_Credential = (4194303, ?, 4194303))
7. VERIFY Access_Event_Tag = EventTag + 1
```

### 7.3.2.X56.4 Authorization Mode Test

Purpose: To verify each authorization mode supported by this IUT.

Test Concept:

For each authorization mode supported by the IUT, a valid credential is presented at the access point to verify that the appropriate action is taken.

Configuration Requirements:

See 7.3.2.X56. This test requires the following additional configuration:

- a) An active credential with valid access rights for the access point shall be represented by Access Credential object C1.
- b) An active credential with no valid access rights shall be represented by Access Credential object C2.

Note: If the VERIFICATION\_REQUIRED or AUTHORIZATION\_DELAYED mode is supported, the vendor must provide a mechanism for external verification to be performed.

Test Steps:

-- verify GRANT\_ACTIVE mode

1. IF (GRANT\_ACTIVE is supported) THEN  
    READ EventTag = Access\_Event\_Tag  
    WRITE Authorization\_Mode = GRANT\_ACTIVE  
    MAKE (present credential C2 at credential reader for this access point)  
    VERIFY Access\_Event = GRANTED  
    VERIFY Access\_Event\_Time = (the time that credential C2 was presented)  
    VERIFY Access\_Event\_Credential = C2  
    VERIFY Access\_Event\_Tag = EventTag + 1

-- verify DENY\_ALL mode

2. IF (DENY\_ALL is supported) THEN  
    READ EventTag = Access\_Event\_Tag  
    WRITE Authorization\_Mode = DENY\_ALL  
    MAKE (present credential C1 at credential reader for this access point)  
    VERIFY Access\_Event = DENIED\_DENY\_ALL  
    VERIFY Access\_Event\_Time = (the time that credential C1 was presented)  
    VERIFY Access\_Event\_Credential = C1  
    VERIFY Access\_Event\_Tag = EventTag + 1

-- verify VERIFICATION\_REQUIRED mode (verification authorized)

3. IF (VERIFICATION\_REQUIRED is supported) THEN  
    READ EventTag = Access\_Event\_Tag  
    WRITE Authorization\_Mode = VERIFICATION\_REQUIRED  
    MAKE (present credential C1 at credential reader for this access point)  
    VERIFY Access\_Event = VERIFICATION\_REQUIRED  
    VERIFY Access\_Event\_Time = (the time that credential C1 was presented most recently)  
    VERIFY Access\_Event\_Credential = C1  
    VERIFY Authentication\_Status = WAITING\_FOR\_VERIFICATION  
    MAKE (external verification process grants access)  
    VERIFY Access\_Event = GRANTED  
    VERIFY Access\_Event\_Time = (the time that verification process granted access)  
    VERIFY Access\_Event\_Credential = C1  
    VERIFY Access\_Event\_Tag = EventTag + 1

-- verify VERIFICATION\_REQUIRED mode (verification denied)

- READ EventTag = Access\_Event\_Tag  
    WRITE Authorization\_Mode = VERIFICATION\_REQUIRED  
    MAKE (present credential C1 at credential reader for this access point)  
    VERIFY Access\_Event = VERIFICATION\_REQUIRED

```

VERIFY Access_Event_Time = (the time that credential C1 was presented)
VERIFY Access_Event_Credential = C1
VERIFY Authentication_Status = WAITING_FOR_VERIFICATION
MAKE (external verification process denies access)
VERIFY Access_Event = DENIED_VERIFICATION_FAILED
VERIFY Access_Event_Time = (the time that verification process denied access)
VERIFY Access_Event_Credential = C1
VERIFY Access_Event_Tag + 1

-- verify VERIFICATION_REQUIRED mode (verification timeout)
READ EventTag = Access_Event_Tag
WRITE Authorization_Mode = VERIFICATION_REQUIRED
MAKE (present credential C1 at credential reader for this access point)
VERIFY Access_Event = VERIFICATION_REQUIRED
VERIFY Access_Event_Time = (the time that credential C1 was presented)
VERIFY Access_Event_Credential = C1
VERIFY Authentication_Status = WAITING_FOR_VERIFICATION
MAKE (external verification process does not respond within verification time)
WAIT Verification_Time
VERIFY Access_Event = DENIED_VERIFICATION_TIMEOUT
VERIFY Access_Event_Time = (the time that verification process timed out)
VERIFY Access_Event_Credential = C1
VERIFY Access_Event_Tag = EventTag + 1

-- verify AUTHORIZATION_DELAYED mode (access granted)
4. IF (AUTHORIZATION_DELAYED is supported) THEN
    WRITE Authorization_Mode = AUTHORIZATION_DELAYED
    MAKE (present credential C1 at credential reader for this access point)
    VERIFY Access_Event = AUTHORIZATION_DELAYED
    VERIFY Access_Event_Time = (the time that credential C1 was presented)
    VERIFY Access_Event_Credential = C1
    MAKE (external verification process does not respond within verification time)
    WAIT Verification_Time
    VERIFY Access_Event = GRANTED
    VERIFY Access_Event_Time = (the time that verification process timed out)
    VERIFY Access_Event_Credential = C1

-- verify AUTHORIZATION_DELAYED mode (access denied)
WRITE Authorization_Mode = AUTHORIZATION_DELAYED
READ EventTag = Access_Event_Tag
MAKE (present credential C1 at credential reader for this access point)
VERIFY Access_Event = AUTHORIZATION_DELAYED
VERIFY Access_Event_Time = (the time that credential C1 was presented)
VERIFY Access_Event_Credential = C1
MAKE (external verification process denies access)
VERIFY Access_Event = DENIED_VERIFICATION_FAILED
VERIFY Access_Event_Time = (the time that verification process denied access)
VERIFY Access_Event_Credential = C1
VERIFY Access_Event_Tag = EventTag + 1

-- verify NONE mode
5. IF (NONE is supported) THEN
    WRITE Authorization_Mode = NONE
    WAIT Internal Processing Fail Time
    VERIFY Authentication_Status = DISABLED

```

### 7.3.2.X56.5 Access Rights Exemptions Test

Purpose: To verify the access rights exemption functionality.

Configuration Requirements:

See 7.3.2.X56. This test requires the following additional configuration:

- a) An active credential with no access rights shall be represented by Access Credential object C1.
- b) The Authorization\_Exemption list of C1 shall be empty.

Test Steps:

-- verify access is denied for the credential

1. MAKE (present credential C1 at credential reader for access point AP1)
2. VERIFY Access\_Event = DENIED\_NO\_ACCESS\_RIGHTS
3. VERIFY Access\_Event\_Time = (the time that the credential was presented)
4. VERIFY Access\_Event\_Credential = C1

-- verify access is granted for the credential when the master exemption set to TRUE

5. MAKE C1, Authorization\_Exemption = (ACCESS\_RIGHTS)
6. MAKE (present credential C1 at credential reader for access point AP1)
7. VERIFY Access\_Event = GRANTED
8. VERIFY Access\_Event\_Time = (the time that the credential was presented)
9. VERIFY Access\_Event\_Credential = C1

### 7.3.2.X56.6 Change Authentication Policy Test

Purpose: To verify that the Active\_Authentication\_Policy property of the Access Point object accepts valid authentication policy values and does not accept invalid ones. It also verifies that an error response is returned if the authentication policy is changed to a non-existent policy number.

Test Concept: The Active\_Authentication\_Policy is written with values of 1 to X, where X is the number of valid authentication policies to verify that the value is accepted. Then, it is written with a value larger than X to verify that the value is rejected with a VALUE\_OUT\_OF\_RANGE error. Finally, it is written with a value of 0 to verify that the value is also rejected with the same error.

Configuration Requirements:

See 7.3.2.X56. This test requires the following additional configuration:

- a) The IUT shall be configured with at least one active authentication policy.
- b) All authentication policies shall be valid policies.

Test Steps:

-- verify that the active authentication policy can set to all valid policies

1. READ Count = Number\_Of\_Authentication\_Policies
2. REPEAT X = (1 to Count) DO {  
    WRITE Active\_Authentication\_Policy = X  
    VERIFY Active\_Authentication\_Policy = X  
}

-- verify that writing an invalid authentication policy to active authentication policy results in a reject

3. TRANSMIT WriteProperty-Request,  
    'Object Identifier' = Access\_Point object,  
    'Property Identifier' = Active\_Authentication\_Policy,  
    'Property Value' = (any value larger than Count)
4. RECEIVE BACnet-Error-PDU,  
    'Error Class' = PROPERTY,  
    'Error Code' = VALUE\_OUT\_OF\_RANGE

-- verify that writing 0 to the authentication policy results in a reject

- 5 TRANSMIT WriteProperty-Request,  
     'Object Identifier' = Access\_Point object,  
     'Property Identifier' = Active\_Authentication\_Policy,  
     'Property Value' = 0
- 6 RECEIVE BACnet-Error-PDU,  
     'Error Class' = PROPERTY,  
     'Error Code' = VALUE\_OUT\_OF\_RANGE

### 7.3.2.X56.7 Lockout State Test

Purpose: To verify that access is denied for any credential when the access point is in the lockout state. To verify that using an invalid credential at the access point multiple times will cause the access point to go into a lockout state. To verify that the lockout will automatically relinquish after the specified time.

Test Concept: A credential which will result in denied access is repeatedly presented at the access point until the access point becomes locked out. When the access point becomes locked, valid credentials will also be denied access until the lockout relinquish time has expired.

Configuration Requirements:

See 7.3.2.X56. This test requires the following additional configuration:

- a) The Max\_Failed\_Attempts property, if present, has a value greater than 0.
- b) An active credential with valid access rights for the access point shall be represented by Access Credential object C1.
- c) An active credential with no valid access rights for the access point shall be represented by Access Credential object C2.
- d) The Failed\_Attempts\_Events list, if present, shall have at least one entry corresponding to the reason why C2 is denied access.
- e) The Lockout\_Relinquish\_Time has a value greater than 0.

Test Steps:

-- verify that valid credentials are denied when the Lockout property is TRUE

1. WRITE Lockout = TRUE
2. WAIT **Internal Processing Fail Time**
3. VERIFY Access\_Event = LOCKOUT\_OTHER
4. VERIFY Access\_Event\_Time = (the time that TRUE was written to the Lockout property)
5. VERIFY Access\_Event\_Credential = (4194303, ?, 4194303)
6. MAKE (present credential C1 at credential reader for this access point)
7. VERIFY Access\_Event = DENIED\_LOCKOUT
8. VERIFY Access\_Event\_Time = (the time that credential C1 was presented)
9. VERIFY Access\_Event\_Credential = C1

-- verify that using an invalid credential at the an access point multiple times will cause the access point to go into a lockout state

10. WRITE Lockout = FALSE
11. WAIT **Internal Processing Fail Time**
12. VERIFY Access\_Event = LOCKOUT\_RELINQUISH
13. VERIFY Access\_Event\_Time = (the time that FALSE was written to the Lockout property)
14. VERIFY Access\_Event\_Credential = (4194303, ?, 4194303)
15. IF (Failed\_Attempts and Max\_Failed\_Attempts are supported) THEN  
     REPEAT X= (1 to Max\_Failed\_Attempts + 1) DO {  
         READ FailedAttempts = Failed\_Attempts  
         MAKE (present credential C2 at credential reader for this access point)  
         VERIFY (Failed\_Attempts = FailedAttempts + 1)  
     }  
 16. VERIFY (Lockout = TRUE)
17. VERIFY (Access\_Event = LOCKOUT\_MAX\_ATTEMPTS)
18. VERIFY (Access\_Event\_Time = the time that Lockout was set to TRUE)
19. VERIFY (Access\_Event\_Credential = C2)



20. MAKE (present credential C1 at credential reader for this access point)
21. VERIFY (Access\_Event = DENIED\_LOCKOUT)
22. VERIFY (Access\_Event\_Time = the time that credential C1 was presented)
23. VERIFY (Access\_Event\_Credential = C1)

-- verify that the lockout will automatically relinquish after the specified time

24. WAIT Lockout\_Relinquish\_Time
25. VERIFY (Lockout = FALSE)
26. VERIFY (Access\_Event = LOCKOUT\_RELINQUISHED)
27. VERIFY (Access\_Event\_Time = the time that Lockout was set to FALSE)
28. VERIFY Access\_Event\_Credential = (4194303, ?, 4194303)
29. MAKE (present credential C1 at credential reader for this access point)
30. VERIFY (Access\_Event = GRANTED)
31. VERIFY (Access\_Event\_Time = the time that credential C1 was presented)
32. VERIFY (Access\_Event\_Credential = C1)

### 7.3.2.X56.8 Threat Level Test

Purpose: To verify Threat\_Level is used to Grant or Deny access based on the Threat\_Authority of the Access Credential.

Test Concept: Vary the Threat\_Level of the access point to be lower or equal than the Threat\_Authority of the credential to verify that access is granted at this access point. Change the Threat\_Level of the access point to the higher than the Threat\_Authority of the credential to verify that access is denied.

Configuration Requirements:

See 7.3.2.X56. This test requires the following additional configuration:

- a) An active credential with valid access rights for the access point shall be represented by Access Credential object C1. This credential shall have a Threat\_Authority of X, where  $0 < X < 100$ .

Test Steps:

-- verify that a credential with threat authority greater than threat level of the access point is granted access

1. WRITE Threat\_Level = (any value less than X)
2. MAKE (present credential C1 at credential reader for this access point)
3. VERIFY Access\_Event = GRANTED
4. VERIFY Access\_Event\_Time = (the time that credential C1 was presented)
5. VERIFY Access\_Event\_Credential = C1

-- verify that a credential with threat authority equal to the threat level of the access point is granted access

6. WRITE Threat\_Level = X
7. MAKE (present credential C1 at credential reader for this access point)
8. VERIFY Access\_Event = GRANTED
9. VERIFY Access\_Event\_Time = (the time that credential C1 was presented)
10. VERIFY Access\_Event\_Credential = C1

-- verify that a credential with threat authority less than the threat level of the access point is denied access

11. WRITE Threat\_Level = (any value greater than X)
12. MAKE (present credential C1 at credential reader for this access point)
13. VERIFY Access\_Event = DENIED\_THREAT\_LEVEL
14. VERIFY Access\_Event\_Time = (the time that credential C1 was presented)
15. VERIFY Access\_Event\_Credential = C1

### 7.3.2.X56.9 Denied Access Occupancy Upper Limit Test

Purpose: To verify occupancy counting, occupancy restrictions, and occupancy exemptions.

Test Concept: When occupancy counting is enabled and a valid credential is presented at the access point, then this test verifies that access is granted only if the occupancy limits are not violated. If the occupancy limits are violated, then access is denied.

If the credential has an occupancy exemption, then the credential will be granted access regardless of the occupancy count of the zone.

#### Configuration Requirements:

See 7.3.2.X56. This test requires the following additional configuration:

- a) The Access Point object is configured to be an entry access point to an access zone which is represented by Access Zone object Z1.
- b) The Occupancy\_Upper\_Limit property of Z1 shall have the value X, where  $X > 0$ .
- c) A number of active credentials all with valid access rights for the access point shall be represented by Access Credential objects  $C1 \dots C(X+1)$  which shall be configured such that each has valid access to the access point.
- d) The Occupancy\_Upper\_Limit\_Enforced property shall have a value of TRUE.
- e) The Occupancy\_Count\_Adjust property shall have a value of TRUE.
- f) The Occupancy\_Count property of Z1 shall have the value 0.
- g) The Occupancy\_Count\_Enable property of Z1 shall have a value of TRUE.
- h) The Authorization\_Exemptions property, if it exists, of  $C(X+1)$  shall be empty.

#### Test Steps:

-- verify that a credential with valid access is granted access while the occupancy count of the zone is less than the Occupancy\_Upper\_Limit property in the zone

1. REPEAT  $C = (C1 \dots CX)$  DO {
  - READ Count = Z1, Occupancy\_Count
  - MAKE (present credential C at credential reader for this access point)
  - VERIFY Z1, Occupancy\_Count = (Count + 1)
  - VERIFY Access\_Event = GRANTED
  - VERIFY Access\_Event\_Time = (the time that credential C was presented)
  - VERIFY Access\_Event\_Credential = C

-- verify that a credential with valid access is denied access when the occupancy count of the zone becomes greater than the Occupancy upper limit

2. READ Count = Z1, Occupancy\_Count
3. VERIFY Z1, Occupancy\_Upper\_Limit = Count
4. MAKE (present credential  $C(X+1)$  at credential reader for this access point)
5. VERIFY Access\_Event = DENIED\_UPPER\_OCCUPANCY\_LIMIT
6. VERIFY Access\_Event\_Time = (the time that credential  $C(X+1)$  was presented)
7. VERIFY Access\_Event\_Credential =  $C(X+1)$
8. VERIFY Z1, Occupancy\_Count = Count

-- verify that a credential with valid access and an occupancy exemption is granted access when the occupancy count of the zone becomes greater than the Occupancy upper limit

9. IF (the Authorization\_Exemption property is not supported or the OCCUPANCY\_CHECK exemption is not supported) THEN {
  10. READ Count = Z1, Occupancy\_Count
  11. VERIFY Z1, Occupancy\_Upper\_Limit = Count
  12. WRITE  $C(X+1)$ , Authorization\_Exemptions = (OCCUPANCY\_CHECK)
  13. MAKE (present credential  $C(X+1)$  at credential reader for this access point)
  14. VERIFY Access\_Event = GRANTED
  15. VERIFY Access\_Event\_Time = (the time that credential  $C(X+1)$  was presented)
  16. VERIFY Access\_Event\_Credential =  $C(X+1)$
  17. VERIFY Z1, Occupancy\_Count = Count + 1

#### 7.3.2.X56.10 Denied Access Disabled Credential Test

Purpose: To test that a credential is denied access at an access point when the credential is disabled even when it has valid access rights.

Test Concept: A disabled credential is presented at an access point and access is denied. For testing purposes, the credential shall be disabled by setting the `Expiration_Time` to a time in the past.

#### Configuration Requirements:

See 7.3.2.X56.1. This test requires the following additional configuration:

- a) An access credential with valid access to AP1 shall be represented by Access Credential C1.
- b) The `Credential_Status` property shall have the value `ACTIVE`.
- c) The `Reason_For_Disable` property shall be empty.

#### Test Steps:

– test granted access when credential is active

1. VERIFY `Reason_For_Disable` = ( )
2. VERIFY `Credential_Status` = `ACTIVE`
3. MAKE (present credential C1 at credential reader for access point AP1)
4. VERIFY AP1, `Access_Event` = `GRANTED`
5. VERIFY AP1, `Access_Event_Time` = (the time that the credential C1 was presented)
6. VERIFY AP1, `Access_Event_Credential` = C1

– test denied access when credential is inactive

7. MAKE (`Expiration_Time` = time < current time)
8. VERIFY `Reason_For_Disable` = (`DISABLED_EXPIRED`)
9. VERIFY `Credential_Status` = `INACTIVE`
10. MAKE (present credential C1 at credential reader for access point AP1)
11. VERIFY AP1, `Access_Event` = `DENIED_CREDENTIAL_EXPIRED`
12. VERIFY AP1, `Access_Event_Time` = (the time that the credential C1 was presented)
13. VERIFY AP1, `Access_Event_Credential` = C1

[Add new **Clause 7.3.2.X57**, p. 185]

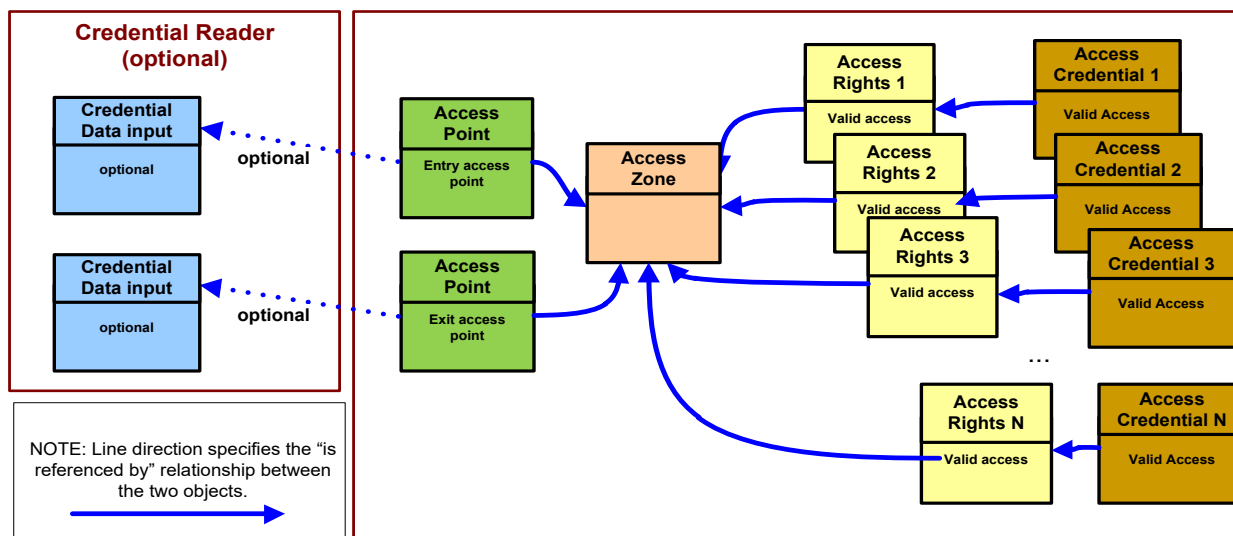
[Reviewer Note: Clause 7.3.2 (Object Specific Tests) contains no tests for the Access Zone object type.]

### 7.3.2.X57 Access Zone Object Tests

Many of the tests for the Access Zone object require interactions from other BACnet access control objects as a result of a valid credential being processed. The required and optional BACnet object types are shown in figure X2.

The Access Zone is defined by list of Access Points that act as entry and exit points of the zone. The configuration requires at least one Access Point object which is configured to be an entry access point and at least one Access Point that is configured to be an exit access point. Each Access Point object which is part of the Access Zone must have an associated Credential Data Input or a proprietary method to input credential values to the Access Point.

Properties in the Access Zone are written by the corresponding Access Point object when a valid credential is processed. The vendor must configure the IUT to have a sufficient number of valid credentials for the test being executed. Each credential must have an associated Access Credential object. Each Access Credential must have an associated Access Rights object that provides valid access to the zone.



**Figure X2:** Objects used for testing the Access Zone object

### 7.3.2.X57.1 Occupancy State Test

Purpose: This test verifies that the occupancy state reflects the occupancy conditions of the zone.

Test Concept: Adjust the occupancy count of the zone to levels above, at, and below the `Occupancy_Upper_Limit` and `Occupancy_Lower_Limit` and verify that the `Occupancy_State` has the appropriate value.

Configuration Requirements:

See 7.3.2.X57. This test requires the following additional configuration:

- The Access Zone shall not be out of service.
- `Occupancy_Count_Enable` shall have a value of TRUE
- The `Occupancy_Upper_Limit` shall have a value X where  $X > 0$ .
- If the property is supported, the `Occupancy_Lower_Limit` shall have a value Y where  $0 < Y < X$ .

Test Steps:

-- test normal case where occupancy count is between the upper and lower limit

- WRITE `Adjust_Value` = 0
- VERIFY `Occupancy_Count` = 0
- WRITE `Adjust_Value` = X-1
- VERIFY `Occupancy_Count` = (X-1)
- VERIFY `Occupancy_State` = NORMAL

-- verify the case where occupancy count is at and above the upper limit

- WRITE `Adjust_Value` = 1
- VERIFY `Occupancy_Count` = X
- VERIFY `Occupancy_State` = AT\_UPPER\_LIMIT
- WRITE `Adjust_Value` = 1
- VERIFY `Occupancy_Count` = X+1
- VERIFY `Occupancy_State` = ABOVE\_UPPER\_LIMIT

-- verify the case where occupancy count is at and above the lower limit

- WRITE `Adjust_Value` = 0
- VERIFY `Occupancy_Count` = 0
- VERIFY `Occupancy_State` = BELOW\_LOWER\_LIMIT
- WRITE `Adjust_Value` = Y
- VERIFY `Occupancy_Count` = Y
- VERIFY `Occupancy_State` = AT\_LOWER\_LIMIT

18. WRITE Adjust\_Value = 1
19. VERIFY Occupancy\_Count = (Y+1)
20. VERIFY Occupancy\_State = NORMAL

-- verify occupancy state when occupancy counting is disabled

21. WRITE Occupancy\_Count\_Enable = FALSE
22. VERIFY Occupancy\_State = DISABLED

### 7.3.2.X57.2 Occupancy Counting Test

Purpose: To verify the occupancy counting functionality.

Test Concept:

Present a credential at the entry and exit access points and test that the occupancy count is properly changed.

Configuration Requirements:

See 7.3.2.X57. This test requires the following additional configuration:

- a) The Access Zone shall not be out of service.
- b) Occupancy\_Count\_Enable shall have a value of TRUE
- c) An access point which is an entry point to this zone shall be represented by Access Point object AP1. The Occupancy\_Count\_Adjust property for this object shall have a value of TRUE.
- d) An access point which is an exit point to this zone shall be represented by Access Point object AP2. The Occupancy\_Count\_Adjust property for this object shall have a value of TRUE.
- e) Two active credentials with valid access rights for the access points AP1 and AP2 shall be represented by Access Credential objects C1 and C2.

Test Steps:

-- verify that presenting a credential at the entry access point increases the occupancy count

1. WRITE Adjust\_Value = 0
2. VERIFY Occupancy\_Count = 0
3. MAKE (present credential C1 at credential reader for access point AP1)
4. VERIFY Occupancy\_Count = 1
5. MAKE (present credential C2 at credential reader for access point AP1)
6. VERIFY Occupancy\_Count = 2

-- verify that presenting a credential at the exit access point decreases the occupancy count

7. MAKE (present credential C1 at credential reader for access point AP2)
8. VERIFY Occupancy\_Count = 1
9. MAKE (present credential C2 at credential reader for access point AP2)
10. VERIFY Occupancy\_Count = 0

### 7.3.2.X57.3 Keeping Track of Credentials Test

Purpose: To verify that the zone can keep track of the current credentials in the zone.

Test Concept:

Present a valid credential at an entry access point to this access zone. The object reference of the credential should be in the credentials in zone list. When the credential is presented to an exit access point for this zone the reference to the credential in the credential in zone list should be removed.

Configuration Requirements:

See 7.3.2.X57. This test requires the following additional configuration:

- a) The Access Zone shall not be out of service.
- b) The Credentials\_In\_Zone property shall be an empty list.
- c) An access point which is an entry point to this zone shall be represented by Access Point object AP1.
- d) An access point which is an exit point to this zone shall be represented by Access Point object AP2.

- e) Two active credential with valid access rights for the access points AP1 and AP2 shall be represented by Access Credential object C1 and C2.

#### Test Steps:

- verify that presenting a credential at an entry access point puts it in the Credentials\_In\_Zone list
1. VERIFY Credentials\_In\_Zone = ( )
  2. MAKE (present credential C1 at credential reader for access point AP1)
  3. VERIFY Credentials\_In\_Zone = (C1)
  4. VERIFY Last\_Credential\_Added = C1
  5. VERIFY Last\_Credential\_Added\_Time = (the time that credential C1 was presented at AP1)
  6. MAKE (present credential C2 at credential reader for access point AP1)
  7. VERIFY Credentials\_In\_Zone = (C1,C2)
  8. VERIFY Last\_Credential\_Added = C2
  9. VERIFY Last\_Credential\_Added\_Time = (the time that credential C2 was presented at AP1)
- verify that presenting a credential at an exit access point removes it from the Credentials\_In\_Zone list
10. MAKE (present credential C1 at credential reader for access point AP2)
  11. VERIFY Credentials\_In\_Zone = (C2)
  12. VERIFY Last\_Credential\_Removed = C1
  13. VERIFY Last\_Credential\_Removed\_Time = the time that credential C1 was presented at AP2
  14. MAKE (present credential C2 at credential reader for access point AP2)
  15. VERIFY Credentials\_In\_Zone = ( )
  16. VERIFY Last\_Credential\_Removed = C2
  17. VERIFY Last\_Credential\_Removed\_Time = the time that credential C2 was presented at AP2

### 7.3.2.X57.4 Passback Mode Test

Purpose: To verify the passback functionality and passback exemption.

Test Concept: A valid credential is presented at the entry access point to this access zone. When the credential is presented a second time, a passback notification should be generated, and if the passback mode is set to hard passback, then access to the zone should be denied. If the credential has a passback exemption, then access will never be denied due to a passback violation.

#### Configuration Requirements:

See 7.3.2.X57. This test requires the following additional configuration:

- a) The Access Zone shall not be out of service.
- b) An access point which is an entry point to this zone shall be represented by Access Point object AP1. The Authorization\_Mode property of AP1 shall have the value Authorize.
- c) An access point which is an exit point to this zone shall be represented by Access Point object AP2. The Authorization\_Mode property of AP2 shall have the value Authorize.
- d) An active credential with valid access rights for the access point AP1 and AP2 shall be represented by Access Credential object C1.
- e) The C1.Authorization\_Exemptions list shall be empty.

#### Test Steps:

- verify soft passback mode
1. MAKE (Passback\_Mode = SOFT\_PASSBACK)
  2. READ EventTag = AP1,Access\_Event\_Tag
  3. MAKE (present credential C1 at credential reader for access point AP1)
  4. VERIFY AP1, Access\_Event = GRANTED
  5. VERIFY AP1,Access\_Event\_Time = (the time that credential C1 was presented)
  6. VERIFY AP1,Access\_Event\_Credential = C1
  7. VERIFY AP1,Access\_Event\_Tag = (EventTag + 1)
  8. MAKE (present credential C1 at credential reader for access point AP1)
  9. VERIFY AP1,Access\_Event = GRANTED
  10. VERIFY (AP1.Access\_Event\_Time = (the time that credential C1 was presented most recently)

```

11. VERIFY AP1.Access_Event_Credential = C1
12. VERIFY AP1,Access_Event_Tag = (EventTag +2)
13. MAKE (present credential C1 at credential reader for access point AP2)

-- verify hard passback mode
14. MAKE (Passback_Mode = HARD_PASSBACK)
15. READ EventTag = AP1,Access_Event_Tag
16. MAKE (present credential C1 at credential reader for access point AP1)
17. VERIFY AP1.Access_Event = GRANTED
18. VERIFY AP1.Access_Event_Time = (the time that credential C1 was presented most recently)
19. VERIFY AP1.Access_Event_Credential = C1
20. VERIFY AP1,Access_Event_Tag = EventTag + 1
21. MAKE (present credential C1 at credential reader for access point AP1)
22. VERIFY AP1.Access_Event = DENIED_PASSBACK
23. VERIFY AP1.Access_Event_Time = (the time that credential C1 was presented most recently)
24. VERIFY AP1.Access_Event_Credential = C1
25. VERIFY AP1,Access_Event_Tag = (EventTag + 2)

-- verify passback timeout
26. WAIT (Passback_Timeout)
27. MAKE (present credential C1 at credential reader for access point AP1)
28. VERIFY AP1.Access_Event = GRANTED
29. VERIFY (AP1.Access_Event_Time = the time that credential C1 was presented)
30. VERIFY (AP1.Access_Event_Credential = C1)

-- verify hard passback off
31. MAKE (Passback_Mode = PASSBACK_OFF)
32. READ EventTag = AP1,Access_Event_Tag
33. MAKE (present credential C1 at credential reader for access point AP1)
34. VERIFY AP1.Access_Event = GRANTED
35. VERIFY AP1.Access_Event_Time = (the time that credential C1 was presented most recently)
36. VERIFY AP1.Access_Event_Credential = C1
37. VERIFY AP1,Access_Event_Tag = EventTag + 1
38. MAKE (present credential C1 at credential reader for access point AP1)
39. VERIFY AP1.Access_Event = GRANTED
40. VERIFY AP1.Access_Event_Time = (the time that credential C1 was presented most recently)
41. VERIFY (AP1.Access_Event_Credential = C1)
42. VERIFY AP1,Access_Event_Tag = (EventTag + 2)

-- verify passback exemption
43. MAKE (Passback_Mode = HARD_PASSBACK)
44. MAKE (C1, Authorization_Exemption = (PASSBACK))
45. READ EventTag = AP1,Access_Event_Tag
46. MAKE (present credential C1 at credential reader for access point AP1)
47. VERIFY AP1.Access_Event = GRANTED
48. VERIFY AP1.Access_Event_Time = (the time that credential C1 was presented most recently)
49. VERIFY AP1.Access_Event_Credential = C1
50. VERIFY AP1,Access_Event_Tag = EventTag + 1
51. MAKE (present credential C1 at credential reader for access point AP1)
52. VERIFY AP1.Access_Event = GRANTED
53. VERIFY AP1.Access_Event_Time = (the time that credential C1 was presented most recently)
54. VERIFY AP1.Access_Event_Credential = C1
55. VERIFY AP1,Access_Event_Tag = (EventTag + 2)

```

[Add new **Clause 7.3.2.X59**, p. 185]

[Reviewer Note: Clause 7.3.2 (Object Specific Tests) contains no tests for the Access Rights object type.]

### 7.3.2.X59 Access Rights Object Tests

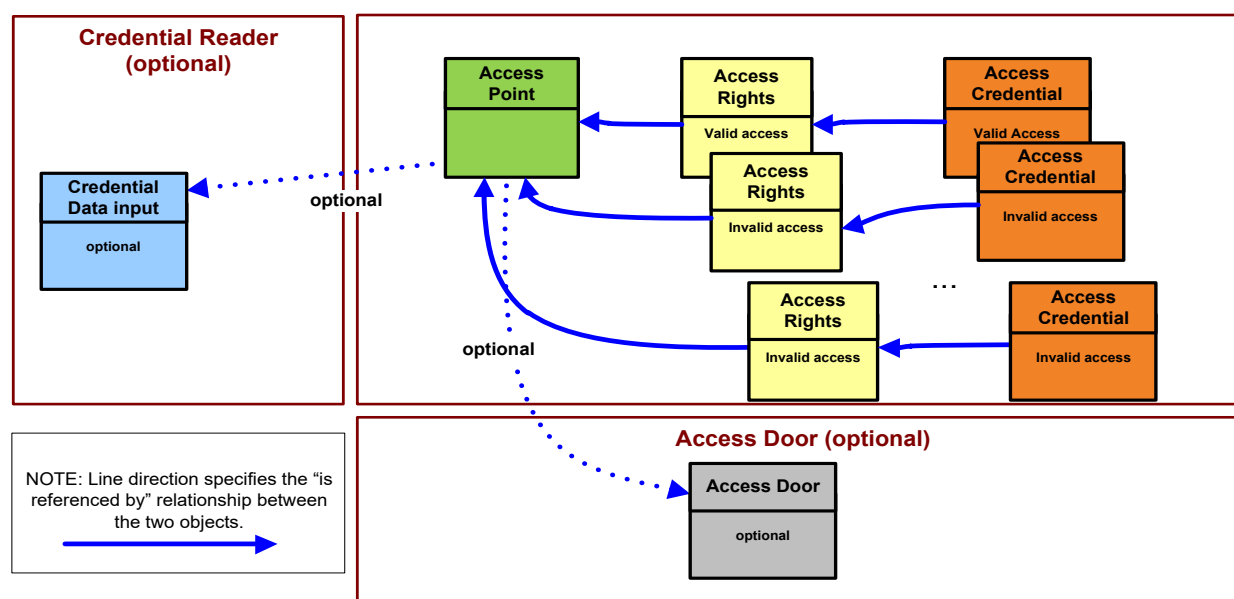
Many of the tests for the Access Rights object require interactions from other BACnet access control objects as a result of a credential being processed. The required and optional BACnet object types are shown in Figure d.

The Access Rights object specifies both positive and negative access rights. This list of access rights is used by the Access Point object to determine the access decision. To test the access rights, the vendor must configure the IUT to have at least one Access Point object which is referenced in the Access Rights objects used for this test.

Each Access Point object used in the test must have an associated Credential Data Input or a proprietary method to input credential values to the Access Point.

The vendor must configure the IUT such that for each Access Rights object there is at least one corresponding Access Credential object that references the Access Rights object.

When the access decision is determined, the IUT shall provide a method to show the result. Typically, the decision is exposed through the Access Door object. When access is granted, the door is pulsed unlocked and when denied the door remains locked. If the Access Door object is not used, then another method of showing the result shall be configured.



**Figure d:** Objects used for testing the Access Rights object

#### 7.3.2.X59.1 Enable Test

Purpose: This test verifies that the access rights object does not allow access when the Enable property is FALSE.

Test Concept: Present a valid credential at the access point. Since the access rights object that provides the access rights to the access point is not enabled, access shall be denied.

Configuration Requirements: See 7.3.2.X59. This test requires the following additional configuration:

- An access point shall be represented by Access Point object AP1.
- An access rights object which specifies valid access rights to AP1 shall be represented by Access Rights object AR1.
- AR1 shall have the Enable property set to TRUE
- An active credential with access rights specified by AR1 shall be represented by Access Credential object C1.

Test Steps:

-- verify access granted with this access rights object when enable property is TRUE

- MAKE (present credential C1 at credential reader for access point AP1)
- VERIFY AP1,Access\_Event = GRANTED



3. VERIFY AP1,Access\_Event\_Time = (the time that the credential was presented)
4. VERIFY AP1,Access\_Event\_Credential = C1
- verify access denied with this access rights object when Enable property is FALSE
5. WRITE AR1, Enable = FALSE
6. MAKE (present credential C1 at credential reader for access point AP1)
7. VERIFY AP1,Access\_Event = DENIED\_NO\_ACCESS\_RIGHTS
8. VERIFY AP1,Access\_Event\_Time = (the time that the credential was presented)
9. VERIFY AP1,Access\_Event\_Credential = C1

### 7.3.2.X59.2 Negative Rules Test

Purpose: To verify that negative access rules explicitly disable access to an access point.

Test Concept: Present a credential at the access point. The access point at which the credential is trying to get entry is in the negative rules list, and that rule is valid at the current time. Access to the access point is denied.

Configuration Requirements: See 7.3.2.X59. This test requires the following additional configuration:

- a) An access point shall be represented by Access Point object AP1.
- b) An access rights object which specifies negative access rights to AP1 shall be represented by Access Rights object AR1.
- c) An active credential with access rights specified by AR1 shall be represented by Access Credential object C1.

Test Steps:

1. MAKE (present credential C1 at credential reader for access point AP1)
2. VERIFY AP1,Access\_Event = DENIED\_POINT\_NO\_ACCESS\_RIGHTS
3. VERIFY AP1,Access\_Event\_Time = (the time that the credential was presented)
4. VERIFY AP1,Access\_Event\_Credential = C1

### 7.3.2.X59.3 Positive Access Rules Test

Purpose: To verify that the positive access rules explicitly enable access to an access point.

Test Concept: Present a credential at the access point. The access point at which the credential is trying to get entry is in the positive rules list, and that rule is valid at the current time. Access to the access point is granted.

Configuration Requirements: See 7.3.2.X59. This test requires the following additional configuration:

- a) An access point shall be represented by Access Point object AP1.
- b) An access rights object which specifies positive access rights to AP1 shall be represented by Access Rights object AR1.
- c) An active credential with access rights specified by AR1 shall be represented by Access Credential object C1.

Test Steps:

1. MAKE (present credential C1 at credential reader for access point AP1)
2. VERIFY AP1,Access\_Event = GRANTED
3. VERIFY AP1,Access\_Event\_Time = (the time that the credential was presented)
4. VERIFY AP1,Access\_Event\_Credential = C1

### 7.3.2.X59.4 Accompaniment Test

Purpose: To verify that the accompaniment functionality works properly.

Test Concept: Present a credential which needs accompaniment at the access point. Wait for the accompaniment time, as specified in the access point. When this times out, the credential should be denied entry to the access point. Present the first credential again at the access point. Present a second credential at the access point which has the rights to accompany the first credential. Access should be granted to the first credential.

Configuration Requirements: See 7.3.2.X59. This test requires the following additional configuration:

- a) An access point shall be represented by Access Point object AP1. If the Accompaniment\_Time property is supported, it shall be set to a value  $> 0$ .
- b) An access rights object which specifies positive access rights to AP1 shall be represented by Access Rights object AR1.
- c) An active credential with access rights specified by AR1 shall be represented by Access Credential object C1 that requires accompaniment.
- d) An active credential which has access rights which allow it to accompany a credential with access rights specified by AR1 through AP1, shall be represented by Access Credential object C2.
- e) An active credential which has valid access rights to AP1 but which does not meet the accompaniment requires of AR1, shall be represented by Access Credential object C3.

Test Steps:

-- valid access through the access point

1. READ Tag = Access\_Event\_Tag
2. MAKE (present credential C1 at credential reader for access point AP1)
3. MAKE (present credential C2 at credential reader for access point AP1)
4. VERIFY AP1,Access\_Event = GRANTED
5. VERIFY AP1,Access\_Event\_Time = (the time that the credential C1 was presented)
6. VERIFY AP1,Access\_Event\_Credential = C1
7. VERIFY AP1,Access\_Event\_Tag = (Tag + 1)

-- no accompaniment presented

8. MAKE (present credential C1 at credential reader for access point AP1)
9. WAIT AP1,Accompaniment\_Time
10. VERIFY AP1,Access\_Event = DENIED\_NO\_ACCOMPANIMENT
11. VERIFY AP1,Access\_Event\_Time = (the time that the credential C1 was presented)
12. VERIFY AP1,Access\_Event\_Credential = C1
13. VERIFY Access\_Event\_Tag = (Tag + 2)

-- Invalid accompaniment

14. MAKE (present credential C1 at credential reader for access point AP1)
15. MAKE (present credential C3 at credential reader for access point AP1)
16. VERIFY AP1,Access\_Event = DENIED\_INCORRECT\_ACCOMPANIMENT
17. VERIFY AP1,Access\_Event\_Time = (the time that the credential C1 was presented)
18. VERIFY AP1,Access\_Event\_Credential = C1
19. VERIFY AP1,Access\_Event\_Tag = (Tag + 3)

[Add new **Clause 7.3.2.X60**, p. 185]

[Reviewer Note: Clause 7.3.2 (Object Specific Tests) contains no tests for the Access Credential object type.]

### 7.3.2.X60 Access Credential Object Tests

Many of the tests for the Access Credential object require interactions from other BACnet access control objects as a result of a credential being processed. The required and optional BACnet object types are shown in Figure e.

The Access Credential defines a list of credential values (authentication factors) that are used to identify the credential. The credential values are used by the Access Point object in determining the access control decision. To test the Access Credential object, the vendor must configure the IUT to have at least one Access Point object. Each Access Credential used in this test must reference an Access Rights object which references the Access Point object

Each Access Point object used in the test must have an associated Credential Data Input or a proprietary method to input credential values to the Access Point.

The vendor must configure the IUT such that for each Access Rights object there is at least one corresponding Access Credential object that references the Access Rights object.

When the access decision is determined, the IUT shall provide a method to show the result. Typically, the decision is exposed through the Access Door object. When access is granted, the door is pulsed unlocked and when denied the door remains locked. If the Access Door object is not used, then another method of showing the result shall be configured.

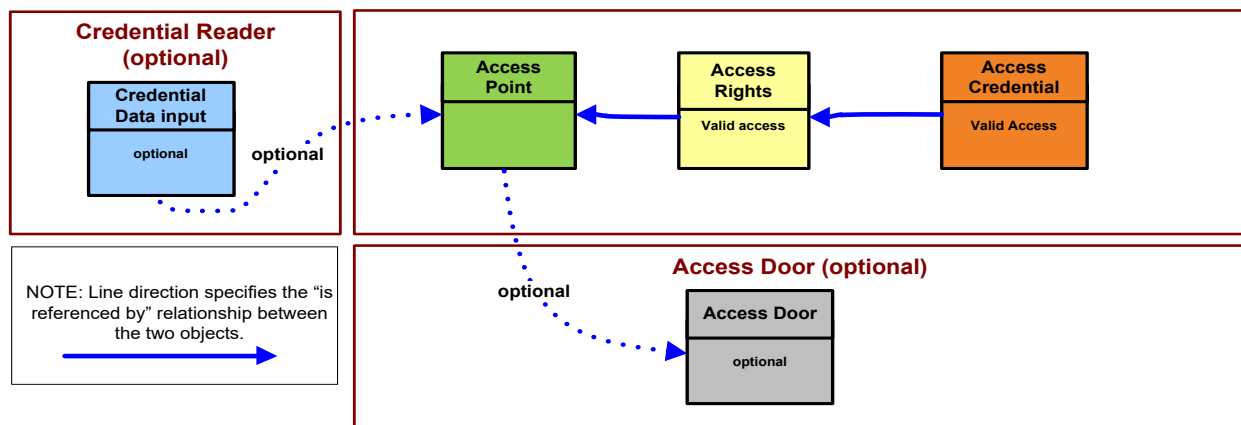


Figure e: BACnet Objects used for testing the Access Credential object

### 7.3.2.X60.1 Credential Status, Credential Disable and Reason for Disable Test

Purpose: To verify the ability to disable the credential and set the associated reason and to enable the credential.

Test Concept: The credential status is set to INACTIVE, and the corresponding reason or reasons are written to the Reason\_For\_Disable list.

Configuration Requirements: See 7.3.2.X60. This test requires the following additional configuration:

- An access credential shall be represented by Access Credential C1.
- The Credential\_Status property shall have the value ACTIVE.
- The Reason\_For\_Disable property shall be empty.

Notes to Tester: This test only verifies the most common disable reasons (DISABLED\_MANUAL, DISABLED\_NOT\_YET\_ACTIVE, DISABLED\_EXPIRED).

Test Steps:

- ```
-- test DISABLED_MANUAL
```
- VERIFY Credential\_Status = ACTIVE
  - VERIFY Reason\_For\_Disable = ( )
  - MAKE (add DISABLED\_MANUAL to Reason\_For\_Disable property)
  - VERIFY Reason\_For\_Disable = (DISABLED\_MANUAL)
  - VERIFY Credential\_Status = INACTIVE
  - MAKE (remove DISABLED\_MANUAL from Reason\_For\_Disable property)
  - VERIFY Reason\_For\_Disable = ( )
  - VERIFY Credential\_Status = ACTIVE

```
-- test DISABLED_NOT_YET_ACTIVE
9.  VERIFY Credential_Status = ACTIVE
10. VERIFY Reason_For_Disable = ( )
11. MAKE (set Activation_Time = time > current time)
12. VERIFY Reason_For_Disable = (DISABLED_NOT_YET_ACTIVE)
13. VERIFY Credential_Status = INACTIVE
14. MAKE (set Activation_Time = time < current time)
15. VERIFY Reason_For_Disable = ( )
16. VERIFY Credential_Status = ACTIVE

-- test DISABLED_EXPIRED
17. VERIFY Credential_Status = ACTIVE
18. VERIFY Reason_For_Disable = ( )
19. MAKE (set Expiration_Time = time < current time)
20. VERIFY Reason_For_Disable = (DISABLED_EXPIRED)
21. VERIFY Credential_Status = INACTIVE
22. MAKE (set Expiration_Time = time > current time)
23. VERIFY Reason_For_Disable = ( )
24. VERIFY Credential_Status = ACTIVE
```

### 7.3.2.X60.2 Activation Time and Expiration Time Test

Purpose: To test the activation time and expiration time functionality of this object.

Test Concept: The Activation\_Time of the credential is set to a time in the future and the credential should be disabled. The Expiration\_Time is set to a time in the past and the credential should be disabled.

Configuration Requirements: See 7.3.2.X60. This test requires the following additional configuration:

- a) The Credential\_Status property shall have the value ACTIVE.
- b) The Reason\_For\_Disable property shall be empty.
- c) The Activation\_Time shall have an initial value of 0xFF
- d) The Expiration\_Time shall have an initial value of 0xFF.

Test Steps:

```
-- test activation time
1.  VERIFY Credential_Status = ACTIVE
2.  VERIFY Reason_For_Disable = ( )
3.  MAKE (set Activation_Time = time > current time)
4.  VERIFY Credential_Status = INACTIVE
5.  VERIFY Reason_For_Disable = (DISABLED_NOT_YET_ACTIVE)
6.  MAKE (set Activation_Time = time < current time)
7.  VERIFY Credential_Status = ACTIVE
8.  VERIFY Reason_For_Disable = ( )

-- test expiration time
9.  VERIFY Credential_Status = ACTIVE
10. VERIFY Reason_For_Disable = ( )
11. MAKE (Expiration_Time = time < current time)
12. VERIFY Credential_Status = INACTIVE
13. VERIFY Reason_For_Disable = (DISABLED_EXPIRED)
14. MAKE (Expiration_Time = time > current time)
15. VERIFY Credential_Status = ACTIVE
16. VERIFY Reason_For_Disable = ( )
```

### 7.3.2.X60.3 Disabled Access Rights Test

Purpose: To verify the enable field disables an access right for this credential when set to FALSE.

Configuration Requirements: See 7.3.2.X60. This test requires the following additional configuration:

- a) An access point shall be represented by Access Point object AP1.
- b) An access rights object which specifies valid access rights to AP1 shall be represented by Access Rights object AR1.
- c) An active credential where Assigned\_Access\_Rights[1].Assigned-Access-Rights = AR1 shall be represented by Access Credential object C1. Assigned\_Access\_Rights[1].Enable shall be TRUE.

Test Steps:

1. MAKE (present credential C1 at credential reader for access point AP1)
2. VERIFY AP1,Access\_Event = GRANTED
3. VERIFY AP1,Access\_Event\_Time = (the time that the credential was presented)
4. VERIFY AP1,Access\_Event\_Credential = C1
5. WRITE Assigned\_Access\_Rights[1].Enable = FALSE
6. MAKE (present credential C1 at credential reader for access point AP1)
7. VERIFY AP1,Access\_Event = DENIED\_NO\_ACCESS\_RIGHTS
8. VERIFY AP1,Access\_Event\_Time = (the time that the credential was presented)
9. VERIFY AP1,Access\_Event\_Credential = C1

### 7.3.2.X60.4 Days Remaining and Uses Remaining Test

Purpose: To test the days remaining and uses remaining functionality of this object.

Test Concept: Set the Days\_Remaining property to 0. The credential will become inactive and the corresponding reason for disable put in the Reason\_For\_Disable property.

Set the Uses\_Remaining to 0. The credential will become inactive and the corresponding reason for disable will be put in the Reason\_For\_Disable property.

Configuration Requirements: See 7.3.2.X60. This test requires the following additional configuration:

- a) The Credential\_Status property shall have the value ACTIVE.
- b) The Reason\_For\_Disable property shall be empty.
- c) Days\_Remaining shall have a value of -1 or >0.
- d) Uses\_Remaining shall have a value of -1 or >0.

Test Steps:

-- test day remaining

1. VERIFY Credential\_Status = ACTIVE
2. VERIFY Reason\_For\_Disable = ( )
3. MAKE (Days\_Remaining = 0 )
4. VERIFY Credential\_Status = INACTIVE
5. VERIFY Reason\_For\_Disable = (DISABLED\_MAX\_DAYS)
6. MAKE (Days\_Remaining = -1 OR Days\_Remaining > 0)
7. VERIFY Credential\_Status = ACTIVE
8. VERIFY Reason\_For\_Disable = ( )

-- test uses remaining

9. VERIFY Credential\_Status = ACTIVE
10. VERIFY Reason\_For\_Disable = ( )
11. MAKE ( Uses\_Remaining = 0 )
12. VERIFY Credential\_Status = INACTIVE
13. VERIFY Reason\_For\_Disable = (DISABLED\_MAX\_USES)
14. MAKE (Uses\_Remaining = -1 OR Uses\_Remaining > 0)
15. VERIFY Credential\_Status = ACTIVE
16. VERIFY Reason\_For\_Disable = ( )

### 7.3.2.X60.5 Absentee Limit Test

Purpose: To verify the absentee limit functionality of this object.

Test Concept: Set the Absentee\_Limit property some value  $\geq 0$ . Use the credential to access an access point. Change the current date to one that is greater than (current date + Absentee\_Limit) and attempt to gain access to an access point. Access should be denied because the credential should be disabled due to inactivity.

Configuration Requirements: See 7.3.2.X60. This test requires the following additional configuration:

- a) Absentee\_Limit  $\geq 0$
- b) The Credential\_Status property shall have the value ACTIVE.
- c) The Reason\_For\_Disable property shall be empty.
- d) Days\_Remaining shall have a value  $> 0$ .
- e) Last\_Use\_Time shall be set to a valid date and time.

Test Steps:

1. VERIFY Credential\_Status = ACTIVE
2. VERIFY Reason\_For\_Disable = ( )
3. TRANSMIT UTCTimeSynchronization-Request, 'Time' = (any day and time greater than Absentee\_Limit days)
4. VERIFY Credential\_Status = INACTIVE
5. VERIFY Reason\_For\_Disable = (DISABLED\_INACTIVITY)
- clear the condition
6. MAKE (set Absentee\_Limit > number of days since Last\_Use\_Time)
7. VERIFY Credential\_Status = ACTIVE
8. VERIFY Reason\_For\_Disable = ( )

### 7.3.2.X60.6 Last Access Point, Last Use Time and Last Access Event Test

Purpose: To verify that the Last Access Point, Last Access Event, and Last Use Time properties are updated when this credential is used at an access point.

Configuration Requirements: See 7.3.2.X60. This test requires the following additional configuration:

- a) An access point shall be represented by Access Point object AP1.
- b) An access rights object which specifies valid access rights to AP1 shall be represented by Access Rights object AR1.
- c) An active credential with access rights specified by AR1 shall be represented by Access Credential object C1.

Test Steps:

1. MAKE (present credential C1 at credential reader for access point AP1)
2. VERIFY AP1,Access\_Event = GRANTED
3. VERIFY AP1,Access\_Event\_Time = (the time that the credential was presented)
4. VERIFY AP1,Access\_Event\_Credential = C1
5. VERIFY Last\_Access\_Point = AP1
6. VERIFY Last\_Use\_Time = AP1,Access\_Event\_Time
7. VERIFY Last\_Access\_Event = GRANTED

### 7.3.2.X60.7 Extended Time Enable Test

Purpose: To verify that a credential used at an access point with the extended flag set to TRUE results in the corresponding door to pulse open for a Pulse\_Extended period of time as defined in the door object.

Configuration Requirements: See 7.3.2.X60. This test requires the following additional configuration:

- a) An access door shall be represented by Access Door object AD1 and Door\_Delay\_Time shall be 0.
- b) An access point, which controls AD1, shall be represented by Access Point object AP1.
- c) An access rights object which specifies valid access rights to AP1 shall be represented by Access Rights object AR1.
- d) An active credential with access rights specified by AR1 shall be represented by Access Credential object C1.
- e) The Extended\_Time\_Enable property shall be FALSE.

Test Steps:

-- verify that PULSE\_UNLOCK is written when the extended time enable is FALSE

1. MAKE (present credential C1 at credential reader for access point AP1)
2. VERIFY AP1,Access\_Event = GRANTED
3. VERIFY AP1,Access\_Event\_Time = (the time that the credential was presented)
4. VERIFY AP1,Access\_Event\_Credential = C1
5. BEFORE Door\_Pulse\_Time VERIFY AD1, Present\_Value = PULSE\_UNLOCK

-- verify that EXTENDED\_PULSE\_UNLOCK is written when the extended time enable is TRUE

6. WRITE Extended\_Time\_Enable = TRUE
7. MAKE (present credential C1 at credential reader for access point AP1)
8. VERIFY AP1,Access\_Event = GRANTED
9. VERIFY AP1,Access\_Event\_Time = (the time that the credential was presented)
10. VERIFY AP1,Access\_Event\_Credential = C1
11. BEFORE Door\_Pulse\_Time VERIFY AD1, Present\_Value = EXTENDED\_PULSE\_UNLOCK

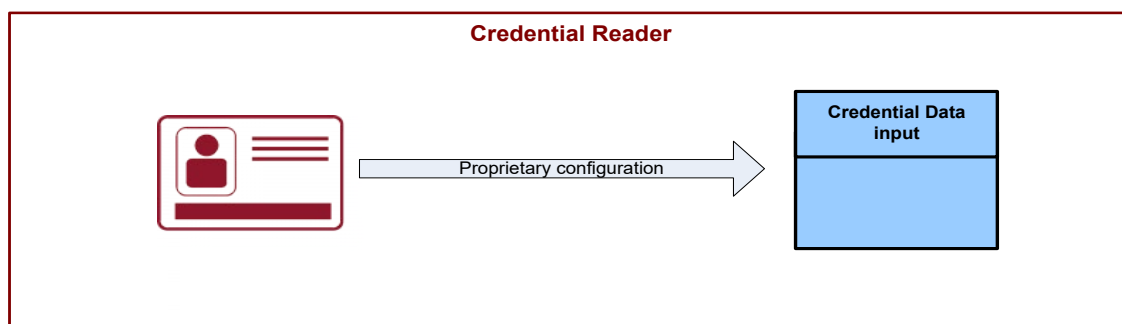
[Add new **Clause 7.3.2.X61**, p. 185]

[Reviewer Note: Clause 7.3.2 (Object Specific Tests) contains no tests for the Credential Data object type.]

### 7.3.2.X61 Credential Data Input Object Tests

The Credential Data Input object type represents a device or process that reads an authentication factor from a physical device such as a card reader, keypad, or biometric device. There are countless variations and authentication formats supported for these devices. As such, there is not a standard format or device configuration that can be mandated for these tests.

The vendor must configure the IUT such that the Credential Data Input device can read an authentication factor from the corresponding physical device, including setting the Supported\_Formats[1] property to the correct authentication factor format (Figure f). This configuration is considered to be a local matter and will not be tested.



**Figure f: Credential Data Input configuration**

#### 7.3.2.X61.1 Return from Out Of Service Undefined Test

Purpose: To verify that the Present\_Value Format-Type becomes undefined when out of service is set to false.

Configuration Requirements: See 7.3.2.X61. This test requires the following additional configuration:

- a) The Out\_Of\_Service property shall be TRUE.

#### Test Steps:

1. WRITE Out\_Of\_Service = FALSE
2. VERIFY Present\_Value. Format-Type = UNDEFINED
3. VERIFY Present\_Value. Format-Class = 0

### 7.3.2.X61.2 Read Valid Authentication Factor Test

Purpose: To verify that Present\_Value is set to the proper value when an authentication factor with a recognized format is read at the corresponding physical device.

Configuration Requirements: See 7.3.2.X61. This test requires the following additional configuration:

- a) The Out\_Of\_Service property shall be FALSE.
- b) Two authentication factors, AF1 and AF2, shall be provided which can be read by the physical device which have the format specified in Supported\_Formats[1].

#### Test Steps:

##### -- test AF1

1. MAKE (present AF1 at the credential reader)
2. VERIFY Present\_Value. Format-Type = Supported\_Formats[1]. Format-Type
3. VERIFY Present\_Value. Format-Class = Supported\_Formats[1]. Format-Class
4. VERIFY Present\_Value. Value = the authentication format value of AF1

##### -- test AF2

5. MAKE (present AF2 at the credential reader)
6. VERIFY Present\_Value. Format-Type = Supported\_Formats[1]. Format-Type
7. VERIFY Present\_Value. Format-Class = Supported\_Formats[1]. Format-Class
8. VERIFY Present\_Value. Value = the authentication format value of AF2

[Add new **Clause 7.3.2.X62**, p. 185]

[Reviewer Note: Clause 7.3.2 (Object Specific Tests) contains no tests for the Network Port object type.]

### 7.3.2.X62 Network Port Object Tests

Configuration Requirements: In addition to the requirements listed for each test, the Network Port object which is being tested shall be configured and operating and have no changes pending, unless required by the specific test's specification.

#### 7.3.2.X62.1 Network Port Configuration Tests

These tests verify that Network Port objects reflect the configuration in use, and for writable ones, change the network configuration.

##### 7.3.2.X62.1.1 Configure Network Through Network Port Object Test

Purpose: This test verifies that Network Port properties control aspects of the network configuration as expected.

Test Concept: Given the complexity of the Network Port object, and the impact changes to the Network Port has on the test network, this test is provided to allow testing of the Network Port functionality as the network is reconfigured for other tests. The Network Port object is modified to meet the conditions of the new test network setup. The changes are activated, the TD



is reconfigured to match, and communication with the IUT is re-verified. The configuration of the network is expected to be tested in more detail as the other datalink tests are applied.

**Test Configuration:** The test network is configured such that the TD and IUT can communicate, but the configuration does not match the target network configuration. P1 through PN are Network Port properties that need to be written in order to transition the network from the current setup to the target network setup. This set of properties shall be selected from the set of the properties that are writable in the IUT.

**Test Steps:**

1. REPEAT P = P1 ... PN {  
    WRITE P = (NV: the value required for the target network setup)  
    VERIFY P = NV  
}
2. VERIFY Changes\_Pending = TRUE
3. REPEAT P = P1 ... PN {  
    CHECK (the new value for P is not in use by the network port, unless the new value is the same as the old value)  
}
4. TRANSMIT ReinitializeDevice-Request  
    'Reinitialized State of Device' = WARMSTART | ACTIVATE\_CHANGES  
    'Password' = (any valid password)
5. RECEIVE BACnet-SimpleACK-PDU
6. MAKE(change the TD network setup and the network setup of all other devices on the network to match the target network setup)
7. WAIT Activate Changes Fail Time
8. VERIFY Changes\_Pending = FALSE

### 7.3.2.X62.1.2 Verify Network Configuration Through Network Port Object Test

**Purpose:** This test verifies that Network Port properties correctly reflect aspects of the network configuration as expected.

**Test Concept:** Given the complexity of the Network Port object, and the impact changes to the Network Port has on the test network, this test is provided to allow testing of the Network Port functionality as the network is reconfigured for other tests. The IUT's network configuration is modified to meet the conditions of the new test network setup. The TD is reconfigured to match, and communication with the IUT is re-verified. The Network Port object is then checked to ensure it reflects the new network setup.

**Test Steps:**

1. MAKE(configure the network, including reconfiguring the TD, IUT, and other devices on the network)
2. CHECK(that the value of each of the present Network Port properties which applies to the associated data link reflects the current network setup)

### 7.3.2.X62.1.3 Network Port Non-Volatility Properties Test

**Purpose:** This test verifies that after Network Port properties are changed and activated, the revised value is maintained through a power failure and device restart.

**Test Concept:** Write one or more properties, P1 ... PN, of a Network Port object which are required for proper operation of the network port. If any of the properties utilize the pending changes functionality, activate the changes. Restart the IUT device by temporarily removing power. When the device has resumed operation after that restart, verify that the new values for the properties were maintained across the reset and are in use by the port.

**Test Steps:**

1. REPEAT P = P1 ... PN {

```

WRITE P = (a new value different from the property's current value)
}
2. IF any of the properties utilize the pending change functionality THEN
  VERIFY Changes_Pending = TRUE
  TRANSMIT ReinitializeDevice-Request
    'Reinitialized State of Device' = WARMSTART | ACTIVATE_CHANGES
    'Password' = (any valid password)
  RECEIVE BACnet-SimpleACK-PDU
  MAKE(reconfigure the TD and other devices on the network to the new network settings)
  WAIT Activate Changes Fail Time
ELSE
  VERIFY Changes_Pending = FALSE
3. REPEAT P = P1 ... PN {
  VERIFY P = (the new value for the property)
}
4. MAKE (the IUT power cycle to reinitialize)
5. REPEAT P = P1 ... PN {
  VERIFY P = (the new value for the property)
  CHECK (that the value for P is in use by the network port)
}

```

### 7.3.2.X62.1.4 Network Port Configuration Conflict Test

Purpose: To verify that either multiple clients can write to a Network Port object at the same time, or the CONFIGURATION\_IN\_PROGRESS error is reported.

Test Concept: The TD simulates 2 devices (TD and TD2), attempting to write to properties in a Network Port object, O1. The IUT must either accept the second write, or the IUT returns an error with an Error Class of DEVICE and an error code of CONFIGURATION\_IN\_PROGRESS. Finally, the Network Port's changes are activated and then verified.

Test Steps:

1. TRANSMIT WriteProperty-Request,  
SOURCE = TD,  
    'Object Identifier' = O1,  
    'Property Identifier' = (P1: a writable property which utilizes the pending changes  
        functionality),  
    'Property Value' = (any valid value),
2. RECEIVE BACnet-SimpleACK-PDU
3. TRANSMIT WriteProperty-Request,  
SOURCE = TD2,  
    'Object Identifier' = O1,  
    'Property Identifier' = (P2: a writable property which utilizes the pending changes  
        functionality, and is different than P1, if possible),  
    'Property Value' = (any valid value),
4. RECEIVE BACnet-SimpleACK-PDU  
    DESTINATION = IUT  
| BACnet-Error-PDU  
    'Error Class' = DEVICE,  
    'Error Code' = CONFIGURATION\_IN\_PROGRESS
5. TRANSMIT ReinitializeDevice-Request,  
    'Reinitialized State of Device' = ACTIVATE\_CHANGES,  
    'Password' = (any valid password)
6. RECEIVE BACnet-SimpleACK-PDU
7. MAKE(reconfigure the TD and other devices on the network to the new network settings)
8. WAIT Activate Changes Fail Time
9. IF P1 is a different property than P2 THEN  
    CHECK(P1's value is in use by the network port)
10. CHECK(P2's value is in use by the network port)

### 7.3.2.X62.2 Network-Number-Is Updates Network\_Number\_Quality Test

Purpose: To verify that Network\_Number\_Quality is updated when the IUT learns its Network\_Number from Network-Number-Is.

Test Concept: Write 0 to Network\_Number to set Network\_Number\_Quality to UNKNOWN. Send a Network-Number-Is message to the IUT indicating that the Network\_Number is learned and verify that Network\_Number\_Quality changes to LEARNED. Send a Network-Number-Is message to the IUT indicating that the Network\_Number is configured and verify that Network\_Number\_Quality changes to LEARNED\_CONFIGURED. Write 0 to Network\_Number and verify that Network\_Number\_Quality changes to UNKNOWN.

Test Configuration: Select a Network Port object, O1, which is enabled and has a writable Network\_Number. Connect the TD to the network associated with Network Port O1. This test shall be skipped if the TD cannot be directly connected to the IUT's network.

Test Steps:

- ```
-- set network number quality to UNKNOWN
1.  WRITE Network_Number = 0
2.    TRANSMIT ReinitializeDevice-Request,
      'Reinitialized State of Device' =  ACTIVATE_CHANGES,
      'Password' = (any valid password)
3.  RECEIVE BACnet-SimpleACK-PDU
4.  WAIT Activate Changes Fail Time
5.  VERIFY Network_Number_Quality = UNKNOWN

-- make IUT learn the network number
6.  TRANSMIT Network-Number-Is
      DESTINATION = LOCAL_BROADCAST | IUT,
      'Network Number' = (N1: any valid value)
      'Flag' = 0          -- learned
7.  VERIFY Network_Number_Quality = LEARNED
8.  VERIFY Network_Number = N1

-- make IUT learn the network number from a configure device
9.  TRANSMIT Network-Number-Is
      DESTINATION = LOCAL_BROADCAST | IUT,
      'Network Number' = (N2: any valid value)
      'Flag' = 1          -- configured
10. VERIFY Network_Number_Quality = LEARNED_CONFIGURED
11. VERIFY Network_Number = N2

-- configure the IUT's network number
12. WRITE Network_Number = (N3: any valid value other than 0)
13.  TRANSMIT ReinitializeDevice-Request,
      'Reinitialized State of Device' =  ACTIVATE_CHANGES,
      'Password' = (any valid password)
14. RECEIVE BACnet-SimpleACK-PDU
15. WAIT Activate Changes Fail Time
16. VERIFY Network_Number_Quality = CONFIGURED

17. TRANSMIT Network-Number-Is
      DESTINATION = LOCAL_BROADCAST | IUT,
      'Network Number' = (N4: any valid value)
      'Flag' = 1          -- configured
18. VERIFY Network_Number_Quality = CONFIGURED
19. VERIFY Network_Number = N3
```

```
-- revert network number quality to UNKNOWN
20. WRITE Network_Number = 0
21.   TRANSMIT ReinitializeDevice-Request,
      'Reinitialized State of Device' =   ACTIVATE_CHANGES,
      'Password' =                       (any valid password)
22. RECEIVE BACnet-SimpleACK-PDU
23. WAIT Activate Changes Fail Time
24. VERIFY Network_Number_Quality = UNKNOWN
```

### 7.3.2.X62.3 Network Port Command Tests

#### 7.3.2.X62.3.1 IDLE Command Rejected

Purpose: To verify that the Command property does not accept write of IDLE.

Test Concept: Write IDLE to the command property and verify that an error-class of PROPERTY with an error-code of VALUE\_OUT\_OF\_RANGE is returned.

Test Configuration: Execute the test against a Network Port object with a writable Command property. This test shall be skipped if the IUT does not support the Command property. The Network Port object shall have no pending changes.

Test Steps:

```
1.   TRANSMIT WriteProperty-Request,
      'Object Identifier' =   (a Network Port object),
      'Property Identifier' = Command,
      'Property Value' =     IDLE
2.   RECEIVE BACnet-Error-PDU
      'Error Class' =   PROPERTY,
      'Error Code' =   VALUE_OUT_OF_RANGE
```

#### 7.3.2.X62.3.2 DISCARD\_CHANGES Command Test

Purpose: To verify that the Network Port discards pending changes when the Command DISCARD\_CHANGES is received.

Test Concept: Write values to one or more properties, P1 .. Px, which utilize the pending changes functionality. Write DISCARD\_CHANGES to the Command property and verify that the properties have reverted to their previous values.

Test Configuration: Execute the test on a Network Port object which supports the DISCARD\_CHANGES command. This test shall be skipped if the IUT does not support the DISCARD\_CHANGES command.

Test Steps:

```
-- save initial values of the properties and change each one to a new value
1.   REPEAT I = (in the range 1 through the number of properties being written) {
      V[I] = READ P[I]
      WRITE P[I] = (a value different than V[I], if possible)
    }

-- discard the changes
2.   WRITE Command = DISCARD_CHANGES
3.   WAIT Activate Changes Fail Time

-- verify that no changes are pending any more
4.   VERIFY Changes_Pending = FALSE
```

## 5. VERIFY Command = IDLE

-- verify that the properties have reverted in value, and that the old value remains in use by the port

```
6. REPEAT I = (in the range 1 through the number of properties being written) {
    VERIFY P[I] = V[I]
    CHECK(the value V[I] is in use by the network port)
}
```

-- command the device to activate any changes which should have no effect

```
7. TRANSMIT ReinitializeDevice-Request
    'Reinitialized State of Device' = WARMSTART | ACTIVATE_CHANGES
    'Password' = (any valid password)
8. RECEIVE BACnet-SimpleACK-PDU
9. MAKE(reconfigure the TD and other devices on the network to the new network settings)
10. WAIT Activate Changes Fail Time
11. VERIFY Command = IDLE
```

-- verify that the properties retain their original values, and that that value remains in use by the port

```
12. REPEAT I = (in the range 1 through the number of properties being written) {
    VERIFY P[I] = V[I]
    CHECK(the value V[I] is in use by the network port)
}
```

## 7.3.2.X62.3.3 RENEW\_FD\_REGISTRATION Command Tests

### 7.3.2.X62.3.3.1 RENEW\_FD\_REGISTRATION Command Test

Purpose: To verify that the Network Port attempts to renew its Foreign Device registration when commanded to do so.

Test Concept: Starting with a Network Port object which is already registered with a BBMD as a Foreign Device, command the Network Port to RENEW\_FD\_REGISTRATION but do have the TD respond. Verify that the Network Port attempts to renew its Foreign Device registration. Wait until the Network Port has completed its attempt and verify that the Reliability has been set to RENEW\_FD\_REGISTRATION\_FAILURE. Command the Network Port to RENEW\_FD\_REGISTRATION and have the TD respond. Verify that the attempt succeeds and that Reliability is reset to NO\_FAULT\_DETECTED. Command the Network Port to RENEW\_FD\_REGISTRATION and have the TD respond. Verify that the attempt succeeds.

Test Configuration: Configure a Network Port for BACnet/IP or BACnet/IPv6 in FOREIGN mode. Allow the IUT to complete its registration with the TD acting as the BBMD before continuing. If the IUT does not support registering as a Foreign Device, or the IUT does not support the RENEW\_FD\_REGISTRATION command, then this test shall be skipped. The Network Port object shall have no pending changes.

Test Steps:

```
-- make sure our initial conditions are good
1. VERIFY Changes_Pending = FALSE
2. VERIFY Reliability = NO_FAULT_DETECTED
3. VERIFY BACnet_IP_Mode = FOREIGN

-- request the renewal, and wait for it to timeout
4. WRITE Command = RENEW_FD_REGISTRATION
5. BEFORE Internal Processing Fail Time
    RECEIVE Register-Foreign-Device
    'Time-to-Live' = FD_Subscription_Lifetime

6. WAIT Foreign Device Registration Fail Time
7. VERIFY Reliability = RENEW_FD_REGISTRATION_FAILURE
8. VERIFY Command = IDLE
```

```
-- re-request the renewal, and allow it to succeed
9.  WRITE Command = RENEW_FD_REGISTRATION
10. BEFORE Internal Processing Fail Time
    RECEIVE Register-Foreign-Device
    'Time-to-Live' = FD_Subscription_Lifetime
11. TRANSMIT BVLC-Result,
    'Result Code' = Successful completion
12. VERIFY Reliability = NO_FAULT_DETECTED
13. VERIFY Command = IDLE

14. WAIT (a random amount of time significantly less than FD_Subscription_Lifetime)

-- re-request the renewal, and allow it to succeed
15. WRITE Command = RENEW_FD_REGISTRATION
16. BEFORE Internal Processing Fail Time
    RECEIVE Register-Foreign-Device
    'Time-to-Live' = FD_Subscription_Lifetime
17. TRANSMIT BVLC-Result,
    'Result Code' = Successful completion
18. VERIFY Reliability = NO_FAULT_DETECTED
19. VERIFY Command = IDLE
```

### 7.3.2.X62.3.3.2 RENEW\_FD\_REGISTRATION Command Failure Test

Purpose: To verify that Network Port object respond to RENEW\_FD\_REGISTRATION commands when the command is not supported / enabled.

Test Concept: Attempt to command a Network Port which is either not an BACnet/IP nor BACnet/IPv6 port or which is not in FOREIGN mode, to renew its FD subscription. Verify that the attempt fails with an Error Class of PROPERTY and an error code of VALUE\_OUT\_OF\_RANGE.

Test Configuration: Select a Network Port which is not in FOREIGN mode. If the IUT does not support the Command property, then this test shall be skipped.

Test Steps:

```
-- make sure our initial conditions are good
1.  IF Network_Type is IPV4 or IPV6 THEN
2.      VERIFY BACnet_IP_Mode <> FOREIGN

3.  TRANSMIT WriteProperty-Request,
    'Object Identifier' = (the Network Port object),
    'Property Identifier' = Command,
    'Property Value' = RENEW_FD_SUBSCRIPTION,
4.  RECEIVE BACnet-Error-PDU
    'Error Class' = PROPERTY,
    'Error Code' = VALUE_OUT_OF_RANGE
5.  VERIFY Command = IDLE
```

### 7.3.2.X62.3.4 RESTART\_SLAVE\_DISCOVERY Command Tests

#### 7.3.2.X62.3.4.1 RESTART\_SLAVE\_DISCOVERY Command Test

Purpose: To verify that the Network Port restarts the slave discovery process when commanded to.

Test Concept: Starting with a Network Port object which is configured as an MS/TP Slave Proxy, command the Network Port to RESTART\_SLAVE\_DISCOVERY. Verify that the IUT restarts slave discovery.

Test Configuration Configure an MSTP Network Port object to act as an MS/TP Slave Proxy with Auto\_Slave\_Discovery set to TRUE. Configure the TD to act as an MS/TP slave. Delay the start of the test until after the IUT has completed its initial slave confirmation. If the IUT does not support automatic slave discovery, or the IUT does not support the RESTART\_SLAVE\_DISCOVERY command, then this test shall be skipped.

Notes to Tester: The IUT may interrogate the slave addresses in any order. The IUT is allowed to generate any other traffic during the test, including, and is not limited to reading property values from the devices it finds.

Test Steps:

- make sure our initial conditions are good
- 1. VERIFY Network\_Type = MSTP
- 2. VERIFY Slave\_Proxy\_Enable = TRUE
- request the renewal, and wait for it to timeout
- 3. WRITE Command = RESTART\_SLAVE\_DISCOVERY
- 4. BEFORE Slave Proxy Confirm Interval
  - REPEAT addr=(all MS/TP addresses excluding the IUT's MAC address) DO {
  - RECEIVE DESTINATION=addr, SRC=IUT
  - ReadProperty-Request,
  - 'Object Identifier' = (DEVICE,4194303),
  - 'Property Identifier' =Protocol\_Services\_Supported
  - }
- 5. VERIFY Command = IDLE

### 7.3.2.X62.3.4.2 RESTART\_SLAVE\_DISCOVERY Command Failure Test

Purpose: To verify that Network Port object respond to RESTART\_SLAVE\_DISCOVERY commands when the command is not supported / enabled.

Test Concept: Attempt to command a Network Port which is not acting as a slave proxy to RESTART\_SLAVE\_DISCOVERY. Verify that the attempt fails with an Error Class of PROPERTY and an error code of VALUE\_OUT\_OF\_RANGE.

Test Configuration: Select a Network Port which is not configured to be a slave proxy. If the IUT supports slave proxy functionality, this test shall be skipped as the standard does not specify how the IUT should respond when slave proxy is supported but not enabled. If the IUT does not support the Command property, then this test shall be skipped.

Test Steps:

- 1. TRANSMIT WriteProperty-Request,
  - 'Object Identifier' = (the Network Port object),
  - 'Property Identifier' = Command,
  - 'Property Value' = RESTART\_SLAVE\_DISCOVERY
- 2. IF Network\_Type is MSTP THEN
  - RECEIVE BACnet-Error-PDU
    - 'Error Class' = PROPERTY,
    - 'Error Code' = OPTIONAL\_FUNCTIONALITY\_SUPPORTED
  - ELSE
    - RECEIVE BACnet-Error-PDU
      - 'Error Class' = PROPERTY,
      - 'Error Code' = VALUE\_OUT\_OF\_RANGE
- 3. VERIFY Command = IDLE

### 7.3.2.X62.3.5 RENEW\_DHCP Command Tests

#### 7.3.2.X62.3.5.1 RENEW\_DHCP Command Test

Purpose: To verify that the Network Port attempts to renew its addressing information when commanded to.

Test Concept: Starting with a Network Port object which is configured to use DHCP and which supports the RENEW\_DHCP command, command the port to RENEW\_DHCP. Verify that the IUT requests a renewal of addressing information.

Test Configuration: Select a Network Port which is an IP or IPv6 port setup for auto-addressing. If the IUT does not support the Command property, then this test shall be skipped.

Test Steps:

```
-- make sure our initial conditions are good
1. IF Network_Type = IPV4 THEN
    VERIFY IP_DHCP_Enable = TRUE
2. If Network_Type = IPV6 THEN
    VERIFY IPv6_Auto_Addressing_Enable = TRUE

-- request the renewal, and wait for it to timeout
3. WRITE Command = RENEW_DHCP
4. CHECK(that the IUT requested a renewal of its addressing information)
5. VERIFY Command = IDLE
```

#### 7.3.2.X62.3.5.2 RENEW\_DHCP Command Failure Test

Purpose: To verify that Network Port object responds to RENEW\_DHCP commands when the command is not supported / enabled.

Test Concept: Attempt to command a Network Port which is either not an BACnet/IP nor BACnet/IPv6 port or which is not in configured for auto-addressing. Verify that the attempt fails with an Error Class of PROPERTY and an error code of VALUE\_OUT\_OF\_RANGE.

Test Configuration: Select a Network Port which is not an IP or IPv6 port setup for auto-addressing. If the IUT does not support the Command property, then this test shall be skipped.

Test Steps:

```
-- make sure our initial conditions are good
1. IF Network_Type is IPV4 and IP_DHCP_Enable is present THEN
    VERIFY IP_DHCP_ENABLE = FALSE
2. IF Network_Type is IPV6 and IPV6_Auto_Addressing_Enabled is present THEN
    VERIFY IPV6_Auto_Addressing_Enabled = FALSE

3. TRANSMIT WriteProperty-Request,
    'Object Identifier' = (the Network Port object),
    'Property Identifier' = Command,
    'Property Value' = RENEW_DHCP

4. IF Network_Type is IPV4 or IPV6 THEN
    RECEIVE BACnet-Error-PDU
        'Error Class' = PROPERTY,
        'Error Code' = OPTIONAL_FUNCTIONALITY_NOT_SUPPORTED
    ELSE
    RECEIVE BACnet-Error-PDU
        'Error Class' = PROPERTY,
        'Error Code' = VALUE_OUT_OF_RANGE

5. VERIFY Command = IDLE
```



### 7.3.2.X62.3.6 RESTART\_AUTO-NEGOTIATION Command Tests

#### 7.3.2.X62.3.6.1 RESTART\_AUTO-NEGOTIATION Command Test

Purpose: To verify that the Network Port attempts to renegotiate its link speed when commanded to.

Test Concept: Starting with a Network Port object which is configured to auto-negotiate its link speed and which supports the RESTART\_AUTONEGOTIATION command, is commanded to restart auto-negotiation. The link speed is changed, and it is verified that the IUT performs link speed negotiation and is able to communicate with the new speed.

Test Configuration: The TD and IUT are connected on a network for which the IUT performs link speed auto-negotiation. The Network Port object for the port is configured to perform auto-negotiation, If the IUT does not support the RESTART\_AUTONEGOTIATION command, then this test shall be skipped.

Test Steps:

- ```
-- make sure our initial conditions are good
1.  VERIFY Link_Speed_Autonegotiate = TRUE

-- request the renewal, and wait for it to timeout
3.  WRITE Command = RESTART_AUTONEGOTIATION
4.  MAKE(change the link speed for the network or link)
5.  WAIT Auto Negotiation Fail Time
6.  CHECK(check any external indications that the new link speed was detected)
7.  VERIFY Command = IDLE          -- the act of validating the command
                                   -- property is sufficient to validate that the command worked
```

#### 7.3.2.X62.3.6.2 RESTART\_AUTONEGOTIATION Command Failure Test

Purpose: To verify that Network Port objects respond to the RESTART\_AUTO-NEGOTIATION command with the correct error codes when the command is not supported / enabled.

Test Concept: Starting with a Network Port object which is not configured to auto-negotiate its link speed or which does not support the RESTART\_AUTO-NEGOTIATION, command it to restart auto-negotiation. Verify that the correct error code is returned.

Test Configuration: If the network port support auto-negotiation, disable it. If the IUT does not support the Command property, or all Network Port object support auto-negotiation and it cannot be disabled, then this test shall be skipped.

Test Steps:

- ```
-- make sure our initial conditions are good
1.  VERIFY Link_Speed_Auto-negotiate = TRUE

-- request the renewal, and wait for it to timeout
2.  TRANSMIT WriteProperty-Request,
    'Object Identifier' = (the Network Port object),
    'Property Identifier' = Command,
    'Property Value' = RESTART_AUTO-NEGOTIATION

3.  IF the port does not support auto-negotiation THEN
    RECEIVE BACnet-Error-PDU
        'Error Class' = PROPERTY,
        'Error Code' = OPTIONAL_FUNCTIONALITY_NOT_SUPPORTED
    ELSE
```

```
RECEIVE BACnet-Error-PDU
    'Error Class' =      PROPERTY,
    'Error Code' =      VALUE_OUT_OF_RANGE
```

### 7.3.2.X62.3.7 DISCONNECT Command Tests

#### 7.3.2.X62.3.7.1 DISCONNECT Command Test

Purpose: To verify that the Network Port attempts to disconnect its link speed when commanded to.

Test Concept: Starting with a Network Port object which supports the DISCONNECT command. The port is commanded to disconnect. The disconnection of the link is verified.

Test Configuration: The TD and IUT are connected on a network which supports disconnection. If the IUT does not support the DISCONNECT command, then this test shall be skipped.

Test Steps:

-- make sure our initial conditions are good

1. VERIFY Network\_Type = (a network type that supports disconnection, such as PTP)
2. WRITE Command = DISCONNECT
3. CHECK(that the link was disconnected)

#### 7.3.2.X62.3.7.2 DISCONNECT Command Failure Test

Purpose: To verify that Network Port objects respond to the DISCONNECT command with the correct error codes when the command is not supported / enabled.

Test Concept: With a Network Port object for a network which does not support disconnection, command it to disconnect. Verify that the correct error code is returned.

Test Configuration: If the IUT does not support the Command property, or all Network Port object support disconnection, then this test shall be skipped.

Test Steps:

1. TRANSMIT WriteProperty-Request,  
    'Object Identifier' = (the Network Port object),  
    'Property Identifier' = Command,  
    'Property Value' = DISCONNECT
2. RECEIVE BACnet-Error-PDU  
    'Error Class' = PROPERTY,  
    'Error Code' = VALUE\_OUT\_OF\_RANGE

### 7.3.2.X62.3.8 RESTART\_PORT Command Tests

#### 7.3.2.X62.3.8.1 RESTART\_PORT Command Test

Purpose: To verify that the Network Port attempts to restart its port when commanded to.

Test Concept: With a Network Port object which supports the RESTART\_PORT command, command the port to restart. The restart of the port is verified.

Test Configuration: If the IUT does not support the RESTART\_PORT command, then this test shall be skipped.

Test Steps:

- make sure our initial conditions are good
- 1. WRITE Command = RESTART\_PORT
- 2. CHECK(that the port was restarted)

### 7.3.2.X62.3.8.2 RESTART\_PORT Command Failure Test

Purpose: To verify that Network Port objects respond to the RESTART\_PORT command with the correct error codes when the command is not supported.

Test Concept: With a Network Port object which does not support the RESTART\_PORT command, command the port to restart. Verify that the correct error code is returned.

Test Configuration: If the IUT does not support the Command property, or all Network Port object support disconnection, then this test shall be skipped.

Test Steps:

1. TRANSMIT WriteProperty-Request,
  - 'Object Identifier' = (the Network Port object),
  - 'Property Identifier' = Command,
  - 'Property Value' = RESTART\_PORT
2. RECEIVE BACnet-Error-PDU
  - 'Error Class' = PROPERTY,
  - 'Error Code' = OPTIONAL\_FUNCTIONALITY\_NOT\_SUPPORTED

### 7.3.2.X62.3.9 No Commands if Changes\_Pending Test

Purpose: To verify that the Network Port disallows commands, except DISCARD\_CHANGES, when Changes\_Pending.

Test Concept: using Network Port object NP, write values to one or more properties, P1 .. Px, which utilize the pending changes functionality. Write each of the other commands and verify they are rejected.

Test Configuration: Execute the test on a Network Port object which supports the Command property.

Test Steps:

- write some properties
- 1. REPEAT P = (P1 .. Px) {
  - WRITE NP, P = (any valid value)
- }
- verify that changes are pending
- 2. VERIFY Changes\_Pending = TRUE
- write each supported Command value, except DISCARD\_CHANGES
- 3. REPEAT CMD = (all non-IDLE valid values that NP supports except DISCARD\_CHANGES) {
  - TRANSMIT WriteProperty-Request
    - 'Object Identifier' = NP
    - 'Property' = Command,
    - 'Property Value' = CMD
  - RECEIVE BACnet-Error-PDU
    - 'Error Class' = PROPERTY,
    - 'Error Code' = INVALID\_VALUE\_IN\_THIS\_STATE
- }

```
-- revert the Network Port object
4. IF the IUT supports DISCARD_CHANGES THEN {
    WRITE Command = DISCARD_CHANGES
  } ELSE {
    MAKE(the IUT discard its changes)
  }
```

### 7.3.2.X62.4 Hierarchical Network Port Tests

#### 7.3.2.X62.4.1 Valid Hierarchy Test

Purpose: To verify that the set of network port objects in the IUT are organized in a valid hierarchy.

Test Concept: Visit each Network Port object which represents a configured application layer port. Ensure that the top Network Port object has a Protocol\_Level of BACNET\_APPLICATION or NON\_BACNET\_APPLICATION. Visit each Network Port object in the hierarchy ensuring that the Protocol\_Level properties are valid.

Test Steps:

```
1. REPEAT NP = (object id of each Network Port object which has a Protocol_Level of
    BACNET_APPLICATION or NON_BACNET_APPLICATION) {
2.   REPEAT NPx = (object id of each Network Port object in NP's hierarchy) {
     PL = READ (Network Port, NPx), Protocol_Level
     IF PL is BACNET_APPLICATION or NON_BACNET_APPLICATION THEN
       ERROR Invalid Protocol_Level in child Network Port object
     IF PL is PHYSICAL THEN
       VERIFY (Network Port, NPx), Reference_Port = 4194303
     }
  }
```

#### 7.3.2.X62.4.2 Properties in Referenced Network Port Reflected in Top Network Port Object

Purpose: To verify that properties in referenced Network Port objects are reflected in the top Network Port object.

Test Concept: Visit each Network Port object which represents a configured BACnet application layer port. Visit each Network Port object in the hierarchy ensuring that the properties in the referenced Network Port object exist and have the same value in the top Network Port object.

Test Steps:

```
1. REPEAT NP = (object id of each Network Port object which has a Protocol_Level of
    BACNET_APPLICATION) {
    -- verify that the required properties exist for this Network Port object based
    -- on its Network_Type
2.   REPEAT P = (each required property for NP's Network_Type, see Table 12-72) {
     VERIFY (Network Port, NP), P = (any valid value)
   }
3.   REPEAT NPx = (object id of each Network Port object in NP's hierarchy) {
     -- verify that the expected properties exist in the Network Port object based
     -- on its Network_Type and Protocol_Level. In addition, verify that the property
     -- value is inherited into NP (unless already inherited from a different Network Port)
     REPEAT P = (each expected property in NPx based on its Network_Type and
       Protocol_Level as defined in Table 12-73) {
       V1 = READ (Network Port, NPx), P
       IF P is not in a higher Network Port object in this hierarchy THEN
```

```

        }
    }
}
    
```

VERIFY (Network Port, NP), P = V1

### 7.3.2.X62.4.3 Changes Reflected in Top Network Port Object

Purpose: To verify that changing properties in child Network Port objects result in the new property values reflected in the top Network Port object.

Test Concept: Write a writable, inheritable property within a Network Port's hierarchy and verify that the new value is reflected in the top Network Port object after activating the change, if required.

Test Configuration: Select a Network Port object, O1, which represents a configured network port, has a Protocol\_Level of BACNET\_APPLICATION, which references a Network Port object and for which there is a writable inherited property, P, within hierarchy. Let O2 be the Network Port object which contains P.

Test Steps:

1. V1 = READ O2, P
2. VERIFY O1, P = V1
3. WRITE O2, P = (V2: any valid value different that V1)
4. IF O2, Changes\_Pending THEN
  - TRANSMIT ReinitializeDevice
  - 'Reinitialized State of Device' = ACTIVATE\_CHANGES
  - RECEIVE BACnet-SimpleACK-PDU
  - MAKE(reconfigure the TD and other devices on the network to the new network settings)
  - WAIT Activate Changes Fail Time
5. VERIFY O1, P = V2

### 7.3.2.X62.4.4 Changes Reflected in Lower Network Port Objects

Purpose: To verify that changing properties in the top Network Port object results in the new property values reflected in the child Network Port objects.

Test Concept: Write a writable, inherited property within a top Network Port object and verify that the new value is reflected in the property from which value is inherited after activating the change, if required.

Test Configuration: Select a Network Port object, O1, which represents a configured network port, has a Protocol\_Level of BACNET\_APPLICATION, which references a Network Port object and for which there is a writable inherited property, P, in O1 which inherits its value from property P in O2.

Test Steps:

1. V1 = READ O2, P
2. VERIFY O1, P = V1
3. WRITE O1, P = (V2: any valid value different that V1)
4. IF O1, Changes\_Pending THEN
  - TRANSMIT ReinitializeDevice
  - 'Reinitialized State of Device' = ACTIVATE\_CHANGES
  - RECEIVE BACnet-SimpleACK-PDU
  - MAKE(reconfigure the TD and other devices on the network to the new network settings)
  - WAIT Activate Changes Fail Time
5. VERIFY O2, P = V2

### 7.3.2.X62.5 APDU\_Length Test

Purpose: To verify that the Device object does not report a Max\_APDU\_Length\_Accepted that is larger than the largest value reported by the configured and enabled Network Port objects.

Test Concept: Determine the largest APDU\_Length property for all configured and enabled Network Port objects with a Protocol\_Level of BACNET\_APPLICATION. Verify that each is larger than 50 and less than or equal the maximum allowed for the attached datalink. Verify that the Max\_APDU\_Length\_Supported property of the Device object is not larger than that maximum.

Notes to Tester: the maximum allowable APDU\_Length for a network type should be calculated from the maximum NPDU size minus 21 according to SSPC interpretation IC135-2020-2.

Test Steps:

1. MAX\_APDU = 0
2. REPEAT NP = (all configured and enabled Network Port objects with a Protocol\_Level of BACNET\_APPLICATION) {  
    IF NP.APDU\_Length < 50 THEN  
        ERROR "APDU\_Length must not be less than 50."  
    IF NP.APDU\_Length > (the maximum allowable for the Network\_Type) THEN  
        ERROR "APDU\_Length is too large for the connected Network\_Type"  
    IF MAX\_APDU <> NP.APDU\_Length THEN  
        MAX\_APDU = NP.APDU\_Length  
    }  
3. VERIFY (Device, 4194303), Max\_APDU\_Length\_Supported <= MAX\_APDU

### 7.3.2.X62.6 Routing\_Table Test

Purpose: To verify that routes are added to the Routing\_Table property of the Network Port object when they are found.

Test Concept: Starting with a clear routing table, send an I-Am-Router-To-Network message with multiple networks listed and verify that all are added to the Routing\_Table. Send the remaining Nmax-2 I-Am-Router-To-Network messages to the IUT and verify that the entries are placed into the Routing\_Table. Verify that no other Network Port objects are affected by the messages.

Configuration: All enabled Network Port objects, NP1 .. NPx, have empty Routing\_Table properties. NP1 is the Network Port for port A. Nmax is the smaller of the maximum number of entries the IUT can hold in its routing table, and the maximum that can be encoded in a single segment ReadProperty response. N1 .. Nmax is a set of random network numbers, none of which are in use by the IUT. R1 .. Rmax are the router MAC addresses for each of the network numbers in N1 .. Nmax. R1 and R2 shall be the same. The TD and IUT shall be on the same BACnet network and there shall be no other routers connected.

Notes to Tester: If the network cannot be configured with the TD and the IUT on the same network, the test shall be adjusted to include the router to the TDs network instead of having a cleared routing table from the start of the test.

Test Steps:

- verify that no routes are known
1. REPEAT NP = (all enabled Network Port objects with a Protocol\_Level of BACNET\_APPLICATION) {  
    VERIFY NP, Routing\_Table = ()  
    }  
  
-- verify that the IUT notices all routes in the I-Am-Router-To-Network
  2. IF the IUT supports storing the address of more than 1 router THEN  
TRANSMIT PORT A  
    DESTINATION = LOCAL BROADCAST,  
    SOURCE R1,  
    I-Am-Router-To-Network,  
    Network Numbers = N1, N2  
    VERIFY NP1, Routing\_Table = (

```

        (N1, R1, AVAILABLE, (optionally, any valid index)),
        (N2, R2, AVAILABLE, (optionally, any valid index))
    ) -- the order of the entries does not matter
ELSE
TRANSMIT PORT A
    DESTINATION = LOCAL BROADCAST,
    SOURCE R1,
    I-Am-Router-To-Network,
    Network Numbers = N1
    VERIFY NP1, Routing_Table = (
        (N1, R1, AVAILABLE, (optionally, any valid index)),
    )

-- verify that the IUT supports up to Nmax entries
4. REPEAT NP = (all enabled Network Port objects, except NP1, with a Protocol_Level of
    BACNET_APPLICATION) {
    VERIFY NP, Routing_Table = ()
}
5. REPEAT N,R = (N2 up to Nmax, R2 up to Rmax) {
    TRANSMIT PORT A
        DESTINATION = LOCAL BROADCAST,
        SOURCE = R
        I-Am-Router-To-Network,
        Network Numbers = N
    }
6. VERIFY NP1, Routing_Table = (
    N1, R1, AVAILABLE, (optionally, any valid index),
    N2, R2, AVAILABLE, (optionally, any valid index),
    ...
    Nmax, Rmax, AVAILABLE, (optionally, any valid index)
) -- the order of the entries does not matter

-- verify that the other Network Port objects are unaffected
7. REPEAT NP = (all enabled Network Port objects, except NP1, with a Protocol_Level of
    BACNET_APPLICATION) {
    VERIFY NP, Routing_Table = ()
}

```

### 7.3.2.X62.7 DHCP Tests

#### 7.3.2.X62.7.1 Basic IPv4 DHCP Test

Purpose: Verify that the IUT is able to participate in IPv4 DHCP and correctly report its DHCP status.

Test Concept: The DHCP server is removed from network. The IUT is then configured with an IPv4 network requiring DHCP, and if required, its DHCP settings are cleared. The related Network Port object is queried to verify that the DHCP related properties have the appropriate values indicating DHCP has not completed. The DHCP is connected to the network. It is verified that the IUT obtains network settings from the DHCP server, and that the DHCP properties reflect the current status.

Configuration Requirements: The DHCP is disconnected from the network or turned off. The IUT is configured for DHCP and any settings it previously received via DHCP are cleared.

Test Steps:

1. IF the IUT has a second enabled network port THEN  
     VERIFY IP\_DHCP\_Enable = True  
     IF IP\_DHCP\_Lease\_Time property is present THEN

- VERIFY IP\_DHCP\_Lease\_Time = 0
- IF IP\_DHCP\_Lease\_Time\_Remaining property is present THEN
  - VERIFY IP\_DHCP\_Lease\_Time\_Remaining = 0
- IF IP\_DHCP\_Server property is present THEN
  - VERIFY IP\_DHCP\_Server = X'00000000'
- 2. MAKE(connect the DHCP server to the network)
- 3. WAIT until the IUT obtains DHCP information
- 4. IF IP\_DHCP\_Lease\_Time property is present THEN
  - VERIFY IP\_DHCP\_Lease\_Time = (0 or the value provided by the DHCP server)
- 5. IF IP\_DHCP\_Lease\_Time\_Remaining property is present THEN
  - VERIFY IP\_DHCP\_Lease\_Time\_Remaining = (0 or a value less than that provided by the DHCP server)
- 6. IF IP\_DHCP\_Server property is present THEN
  - VERIFY IP\_DHCP\_Server = (the DHCP server's address or X'00000000')
- 7. VERIFY IP\_Address = (the value served by the DHCP server)
- 8. VERIFY IP\_Default\_Gateway = (the value served by the DHCP server)

### 7.3.2.X62.7.2 Basic IPv6 DHCP Test

Test Concept: The DHCP server is removed from network. The IUT is then configured with an IPv6 network requiring DHCP, and if required, its DHCP settings are cleared. The related Network Port object is queried to verify that the DHCP related properties have the appropriate values indicating DHCP has not completed. The DHCP is connected to the network. It is verified that the IUT obtains network settings from the DHCP server, and that the DHCP properties reflect the current DHCP status.

Configuration Requirements: The DHCP is disconnected from the network or turned off. The IUT is configured for DHCP and any settings it previously received via DHCP are cleared.

Test Steps:

1. IF the IUT has a second enabled network port THEN
  - VERIFY IPv6\_DHCP\_Enable = True
  - IF IPv6\_DHCP\_Lease\_Time property is present THEN
    - VERIFY IPv6\_DHCP\_Lease\_Time = 0
  - IF IPv6\_DHCP\_Lease\_Time\_Remaining property is present THEN
    - VERIFY IPv6\_DHCP\_Lease\_Time\_Remaining = 0
  - IF IPv6\_DHCP\_Server property is present THEN
    - VERIFY IPv6\_DHCP\_Server = X'00000000'
2. MAKE (connect the DHCP server to the network)
3. WAIT until the IUT obtains DHCP information
4. IF IPv6\_DHCP\_Lease\_Time property is present THEN
  - VERIFY IPv6\_DHCP\_Lease\_Time = (0 or the value provided by the DHCP server)
5. IF IPv6\_DHCP\_Lease\_Time\_Remaining property is present THEN
  - VERIFY IPv6\_DHCP\_Lease\_Time\_Remaining = (0 or a value less than that provided by the DHCP server)
6. IF IPv6\_DHCP\_Server property is present THEN
  - VERIFY IPv6\_DHCP\_Server = (the DHCP server's address or X'00000000')
7. VERIFY IPv6\_Address = (the value served by the DHCP server)
8. VERIFY IPv6\_Default\_Gateway = (the value served by the DHCP server)

[Add new **Clause 7.3.2.X63**, p. 185]

[Reviewer Note: Clause 7.3.2 (Object Specific Tests) contains no tests for the Timer object type.]

### 7.3.2.X63 Timer Object Tests



### 7.3.2.X63.1.1 Timer State\_Change\_Values

Purpose: Verify all State\_Change\_Values property transitions.

Test Concept: Start this test in the IDLE state. Write Timer\_Running and observe the object enter RUNNING state, and after time passes, observe it enters the EXPIRED state. Force it into the IDLE state. Write Timer\_Running again observe it enter RUNNING state, then write Timer\_Running again and observe it make the RUNNING\_TO\_RUNNING transition. Force it into the IDLE state again. Observe that per State\_Change\_Values transitions, all writes take place.

Configuration Requirements: The State\_Change\_Values property, if present, holds a valid configuration.

Test Steps:

1. WRITE Timer\_Running = TRUE
2. CHECK (IUT exhibits any changes configured in IDLE\_TO\_RUNNING transition)
3. READ RT = Present\_Value
4. WAIT RT
5. CHECK (IUT exhibits any changes configured in RUNNING\_TO\_EXPIRED transition)
6. WRITE Timer\_State = IDLE
7. CHECK (IUT exhibits any changes configured in EXPIRED\_TO\_IDLE transition)
8. WRITE Timer\_Running = TRUE
9. CHECK (IUT exhibits any changes configured in IDLE\_TO\_RUNNING transition)
10. BEFORE the timer expires  
    WRITE Timer\_Running = TRUE
11. CHECK (IUT exhibits any changes configured in RUNNING\_TO\_RUNNING transition)
12. WRITE Timer\_State = IDLE
13. CHECK (IUT exhibits any changes configured in RUNNING\_TO\_IDLE transition)
14. VERIFY Present\_Value = 0

### 7.3.2.X63.1.2 Timer Running then Expired Test

Purpose: Verify that Timer T1 when set to RUNNING, enters the EXPIRED state after the configured time.

Test Concept: Start this test in the IDLE state. Write Timer\_Running and observe it enter RUNNING state, with its current configuration. After time passes, observe it enters the EXPIRED state. Observe that specified properties take their required values.

Configuration Requirements: T1 starts this test with the Timer\_State equal to IDLE.

Test Steps:

1. VERIFY Timer\_State = IDLE
2. WRITE Timer\_Running = TRUE
3. CHECK (IUT exhibits any changes configured in IDLE\_TO\_RUNNING transition)
4. IF (Default\_Timeout property is present in T1) THEN {  
    READ DV = Default\_Timeout  
    VERIFY Initial\_Timeout = DV  
}
5. VERIFY Timer\_State = RUNNING
6. READ RT = Present\_Value
7. WAIT RT
8. CHECK (IUT exhibits any changes configured in RUNNING\_TO\_EXPIRED transition)
9. VERIFY Timer\_State = EXPIRED
10. VERIFY Present\_Value = 0
11. VERIFY Last\_State\_Change = RUNNING\_TO\_EXPIRED
12. IF (Update\_Time property is present in T1) THEN {  
    READ UT = Update\_Time  
    VERIFY UT ~= (the current date and time)  
    IF (Expiration\_Time property is present in T1) THEN

```

        VERIFY Expiration_Time = UT
    }
    ELSE IF (Expiration_Time property is present in T1) THEN
        VERIFY Expiration_Time ~= (the current date and time)

```

### 7.3.2.X63.1.3 Default\_Timeout Test

Purpose: Verify that starting the Timer running via Timer\_Running uses Default\_Timeout.

Test Concept: Start this test in the IDLE state. Default\_Timeout is a valid non-zero value, different from the value which is Initial\_Timeout at start of test. Write Timer\_Running and observe the Timer enter RUNNING state. Observe that specified properties take their required values.

Configuration Requirements: Timer starts this test with the Timer\_State equal to IDLE. IUT is configured with Initial\_Timeout and Default\_Timeout being different. If this cannot be done, then this test shall be skipped.

Test Steps:

1. READ PrevIT = Initial\_Timeout
2. VERIFY Default\_Timeout <> PrevIT
3. WRITE Timer\_Running = TRUE
4. VERIFY Present\_Value ~= Default\_Timeout

### 7.3.2.X63.1.4 Running Timer by writing the Present\_Value

Purpose: Start or Restart the Timer by writing the Present\_Value property.

Test Concept: Start the Timer with a non-zero value PV1, written to the Present\_Value property, and observe that the Timer counts down according to the new value written.

Configuration Requirements: This test can start with T1 in any of the IDLE, EXPIRED, or RUNNING states.

Test Steps:

1. IF (Default\_Timeout property is present in T1) THEN  
    READ DT = Default\_Timeout
2. READ PrevIT = Initial\_Timeout
3. WRITE Present\_Value = (PV1, any valid non-zero value, sufficiently different from DT and PrevIT so that it is clear that countdown is according to the new value written)
4. VERIFY Present\_Value <= PV1

### 7.3.2.X63.1.5 Restarting An Expired Timer

Purpose: Verify that writes to Timer\_Running with TRUE while in the EXPIRED state are successful.

Test Concept: Start this test with the Timer\_State equal to EXPIRED. Write TRUE to Timer\_Running.

Configuration Requirements: T1 starts this test with the Timer\_State equal to EXPIRED.

Test Steps:

1. VERIFY Timer\_State = EXPIRED
2. WRITE Timer\_Running = TRUE
3. VERIFY Timer\_State = RUNNING

### 7.3.2.X63.1.6 Already Running Timer restarted by writing the Present\_Value

Purpose: Verify Timer can be restarted while running by writing Present\_Value.

Test Concept: Configure and run the Timer T1 as necessary to put it into RUNNING state. Then write the Timer with a different non-zero value written to the Present\_Value property and observe that specified properties take their required values and all configured State\_Change\_Values transitions if any, take place. The timer counts down and observe that it operates according to the new value written.

Configuration Requirements: T1 starts this test with the Timer\_State equal to RUNNING. If Present\_Value is not writable, this test shall be skipped.

Test Steps:

1. VERIFY Timer\_State = RUNNING
2. WRITE Present\_Value = (any valid non-zero value)
3. CHECK (IUT exhibits any changes configured in RUNNING\_TO\_RUNNING transition)
4. VERIFY Timer\_Running = TRUE
5. VERIFY Last\_State\_Change = RUNNING\_TO\_RUNNING

### 7.3.2.X63.1.7 Already Running Timer restarted with Default\_Timeout

Purpose: Verify the success of writes to Timer\_Running with TRUE while already in the RUNNING state.

Test Concept: Configure and run the Timer T1 as necessary to put it into RUNNING state with an Initial\_Value different from Default\_Value. Then write the Timer\_Running property with TRUE and observe that Present\_Value restarts with the value from Default\_Timeout.

Configuration Requirements: T1 starts this test with the Timer\_State equal to RUNNING. In service of observing the change between step 3 and step 6, it is necessary that at the test start, the Timer went into RUNNING state with an Initial\_Value different from Default\_Value.

Test Steps:

1. VERIFY Timer\_State = RUNNING
2. READ DV = Default\_Timeout
3. VERIFY Initial\_Timeout <> DV
4. WRITE Timer\_Running = TRUE
5. CHECK (IUT exhibits any changes configured in RUNNING\_TO\_RUNNING transition)
6. VERIFY Initial\_Timeout = DV
7. VERIFY Present\_Value == DV
8. VERIFY Timer\_Running = TRUE
9. VERIFY Last\_State\_Change = RUNNING\_TO\_RUNNING

### 7.3.2.X63.1.8 Timer accepts all the required datatypes in an Internal Reference

Purpose: Verify that the IUT with a modifiable List\_Of\_Object\_Property\_References, accepts all the required datatypes.

Test Concept: Verify in a Timer object, T1, that supports modification, the IUT allows altering the List\_Of\_Object\_Property\_References to refer to an object within the IUT. Repeat for each of the datatypes required for Timers with writable List\_Of\_Object\_Property\_References. Also write State\_Change\_Values or its entries with values of that datatype.

Notes to Tester: It is required that the IUT allows modifying the Timer one property at a time.

#### Test Steps:

1. REPEAT (for all the required datatypes: values of type NULL, BOOLEAN, Unsigned, INTEGER, REAL, and ENUMERATED {  
     WRITE (the State\_Change\_Values and/or List\_Of\_Object\_Property\_References with that datatype, and which references an object within the IUT)  
   }  
 2. CHECK (Did the IUT accept the writes, and apply the modified values to properties correctly?)

### 7.3.2.X63.1.9 Timer supports writing an External Device

Purpose: Verify that IUT allows its List\_Of\_Object\_Property\_References to refer to an external device.

Test Concept: Verify in a Timer object, T1, that supports modification, the IUT allows altering the List\_Of\_Object\_Property\_References to refer to an object in an external device.

Configuration Requirements: If there is no Timer object in IUT which supports reference to an object in an external device, then this test shall be skipped.

#### Test Steps:

1. WRITE List\_Of\_Object\_Property\_References = (a value that is different, and which contains one or more references to objects which are in one or more external devices)
2. VERIFY List\_Of\_Object\_Property\_References = (the value written)

### 7.3.2.X63.1.10 Forcing Timer Expiration by writing Zero

Purpose: Interrupt the Timer while it is RUNNING, via a value of zero written to the Present\_Value property.

Test Concept: Configure and start the Timer, T1, to operate according to its values. Then write a value of zero to the Present\_Value property and observe that specified properties take their required values and that the State\_Change\_Values operations take place.

Configuration Requirements: T1 starts this test with the Timer\_State equal to RUNNING.

#### Test Steps:

1. VERIFY Timer\_Running = TRUE
2. VERIFY Timer\_State = RUNNING
3. WRITE Present\_Value = 0
4. CHECK (IUT exhibits any changes configured in FORCED\_TO\_EXPIRED transition)
5. VERIFY Timer\_State = EXPIRED
6. VERIFY Last\_State\_Change = FORCED\_TO\_EXPIRED
7. VERIFY Present\_Value = 0
8. IF (Update\_Time property is present in T1) THEN {  
     READ UT = Update\_Time  
     VERIFY UT ≈ (the current date and time)  
     IF (Expiration\_Time property is present in T1) THEN  
         VERIFY Expiration\_Time = UT  
   }  
   ELSE IF (Expiration\_Time property is present in T1) THEN  
     VERIFY Expiration\_Time ≈ (the current date and time)

### 7.3.2.X63.1.11 Forcing Timer Expiration by writing FALSE

Purpose: Interrupt the Timer while it is RUNNING, via a value of FALSE written to the Timer\_Running property.

Test Concept: Configure and start Timer, T1, to operate according to its values. Then write FALSE to Timer\_Running and observe that specified properties take their required values and all configured State\_Change\_Values transitions if any, take place.

Configuration Requirements: T1 starts this test with the Timer\_State equal to RUNNING.

Test Steps:

1. VERIFY Timer\_Running = TRUE
2. VERIFY Timer\_State = RUNNING
3. WRITE Timer\_Running = FALSE
4. CHECK (IUT exhibits any changes configured in FORCED\_TO\_EXPIRED transition)
5. VERIFY Timer\_State = EXPIRED
6. VERIFY Last\_State\_Change = FORCED\_TO\_EXPIRED
7. IF (Update\_Time property is present in T1) THEN {  
    READ UT = Update\_Time  
    VERIFY UT ~= (the current date and time)  
    IF (Expiration\_Time property is present in T1) THEN  
        VERIFY Expiration\_Time = UT  
    }  
ELSE IF (Expiration\_Time property is present in T1) THEN  
    VERIFY Expiration\_Time ~= (the current date and time)

### 7.3.2.X63.1.12 Forcing Timer Expiration by writing IDLE

Purpose: Interrupting the Timer while it is RUNNING, via a value of IDLE written to the Timer\_State property.

Test Concept: Configure and start the Timer, T1, to operate according to its values. Then write IDLE to Timer\_State and observe that specified properties take their required values and all configured State\_Change\_Values transitions if any, take place.

Configuration Requirements: T1 starts this test with the Timer\_State equal to RUNNING.

Test Steps:

1. VERIFY Timer\_Running = TRUE
2. VERIFY Timer\_State = RUNNING
3. WRITE Timer\_State = IDLE
4. CHECK (IUT exhibits any changes configured in RUNNING\_TO\_IDLE transition)
5. VERIFY Timer\_State = IDLE
6. VERIFY Last\_State\_Change = RUNNING\_TO\_IDLE
7. IF (Expiration\_Time property is present in T1) THEN  
    VERIFY Expiration\_Time = (the unspecified datetime value)
8. IF (Update\_Time property is present in T1) THEN  
    VERIFY Update\_Time = (the current date and time)
9. VERIFY Present\_Value = 0

### 7.3.2.X63.1.13 Resetting Timer by writing IDLE

Purpose: Verify the correct behaviors when Timer\_State is written from EXPIRED to IDLE value.

Test Concept: Configure and run the Timer as necessary to put it into EXPIRED state. Then write IDLE to Timer\_State and observe that specified properties take their required values and all configured State\_Change\_Values transitions if any, take place.

Configuration Requirements: T1 starts this test with the Timer\_State equal to EXPIRED.

Test Steps:

1. VERIFY Timer\_State = EXPIRED
2. WRITE Timer\_State = IDLE
3. CHECK (IUT exhibits any changes configured in EXPIRED\_TO\_IDLE transition)
4. VERIFY Timer\_State = IDLE
4. VERIFY Timer\_Running = FALSE
5. VERIFY Last\_State\_Change = EXPIRED\_TO\_IDLE
6. IF (Expiration\_Time property is present in T1) THEN  
    VERIFY Expiration\_Time = (the unspecified datetime value)
7. IF (Update\_Time property is present in T1) THEN  
    VERIFY Update\_Time = (the current date and time)
8. VERIFY Present\_Value = 0

### 7.3.2.X63.1.14 Timer Object Operation Unaffected by Changes to Local\_Time and Local\_Date

Purpose: Verify that Timer expiration is not affected by time changes.

Test Concept: Configure and start the Timer, T1, to operate according to its values. Then before the Timer expires, change Local\_Date / Local\_Time to a NewDate / NewTime in the past or future and observe that the length of time until Timer expiration is not affected, and that expiry still occurs at the time indicated in Present\_Value.

Configuration Requirements: T1 starts this test with the Timer\_State equal to RUNNING.

Test Steps:

1. VERIFY Timer\_Running = TRUE
2. VERIFY Timer\_State = RUNNING
3. READ PV\_beforeTimeChange = Present\_Value
4. MAKE (Local\_Date/ Local\_Time = NewDate/ NewTime)
4. VERIFY Present\_Value ~ PV\_beforeTimeChange, continuing its decreasing trend
5. VERIFY Local\_Date = NewDate
6. VERIFY Local\_Time ~ NewTime
7. WAIT PV\_beforeTimeChange
8. VERIFY Present\_Value = 0
9. IF (Update\_Time property is present in T1) THEN  
    VERIFY Update\_Time ~ (the current date and time)
10. IF (Expiration\_Time property is present in T1) THEN  
    VERIFY Expiration\_Time ~ (the current date and time)

Hints to Tester: To ensure that testing would detect an implementation which prematurely expires when the Local\_Time becomes a time that the Timer would not be RUNNING, select NewDate / NewTime which when converted to local date/time using UTC\_Offset and Daylight\_Saving\_Status, is earlier or later than the window of time that the Timer would be RUNNING when it started.

### 7.3.2.X63.1.15 Changes made by State\_Change\_Values are at Correct Priority

Purpose: Verify by changing the Priority\_for\_Writing property that subsequent State\_Change\_Values operations use the right Priority.

Test Concept: Write `Timer_Running` and observe it enter `RUNNING` state, with its current configuration. After time passes, observe it enters the `EXPIRED` state. Observe that specified properties take their required values.

Configuration Requirements: Start this test in the `IDLE` state. `O1 P1` is any commandable property amongst the elements of `List_Of_Object_Property_References`. `O1` should contain in its `Priority_Array` at the index which will change, a value which is different from the values in `State_Change_Values`, for the operations which will take place, for ease of ensuring that the Timer commanded the change.

Test Steps:

1. WRITE `Priority_for_Writing` = (any valid value, different from what it had)
2. READ `ValueItR` = `State_Change_Values`, `ARRAY INDEX` = `IDLE_TO_RUNNING`
3. WRITE `Timer_Running` = `TRUE`
4. VERIFY `Timer_State` = `RUNNING`
5. READ `RT` = `Present_Value`
6. IF (`ValueItR` is a value other than no-value) THEN  
    VERIFY (`O1`), `Priority_Array` = `ValueItR`, `ARRAY INDEX` = `Priority_for_Writing`
7. WAIT `RT`
8. READ `ValueRtE` = `State_Change_Values`, `ARRAY INDEX` = `RUNNING_TO_EXPIRED`
9. VERIFY `Timer_State` = `EXPIRED`
10. IF (`ValueRtE` is a value other than no-value) THEN  
    VERIFY (`O1`), `Priority_Array` = `ValueRtE`, `ARRAY INDEX` = `Priority_for_Writing`

### 7.3.2.X63.1.16 Changing Default\_Timeout Test

Purpose: Reconfigure the `Default_Timeout` and see that governs the length the timer runs.

Test Concept: Start this test in the `IDLE` state. Configure the Timer with an updated `Default_Time` and observe it operates according to the new value written.

Configuration Requirements: `T1` starts this test with the `Timer_State` equal to `IDLE`.

Test Steps:

1. READ `IT` = `Initial_Timeout`
2. WRITE `Default_Timeout` = (any valid value, different from `IT` and different from what it had)
3. WRITE `Timer_Running` = `TRUE`
4. VERIFY `Present_Value`  $\approx$  `Default_Timeout`

### 7.3.2.X63.2.1 Writing Timer with an Unsupported External Reference

Purpose: Verify the correct `Result(-)` when `List_Of_Object_Property_References` does not support objects in an external device.

Test Concept: Attempt writing `List_Of_Object_Property_References` of a Timer object, `T1`, which does not support referring to an object in an external device. Verify the IUT returns the correct `Result(-)`.

Configuration Requirements: If the IUT supports referring to an object in an external device in all of its Timer objects, then this test shall be skipped.

Test Steps:

1. TRANSMIT `WriteProperty-Request`,  
    '`Object Identifier`' = `T1`  
    '`Property Identifier`' = `List_Of_Object_Property_References`  
    '`Property Value`' = (a value that is different, and which references an object in an external device)
2. RECEIVE `BACnet-Error-PDU`,  
    '`Error Class`' = `PROPERTY`

'Error Code' = OPTIONAL\_FUNCTIONALITY\_NOT\_SUPPORTED

### 7.3.2.X63.2.2 Writing an Unsupported Datatype to State\_Change\_Values

Purpose: Verify the correct Result(-) when State\_Change\_Values is written with a datatype that instance does not support.

Test Concept: Attempt writing State\_Change\_Values of a Timer object T1 with a datatype that instance does not support. Verify the IUT returns the correct Result(-).

Configuration Requirements: The State\_Change\_Values property initially holds a valid configuration.

Test Steps:

1. TRANSMIT WriteProperty-Request,  
    'Object Identifier' = T1  
    'Property Identifier' = State\_Change\_Values  
    'Property Value' = (a value which contains a datatype that T1 does not support)
2. RECEIVE BACnet-Error-PDU,  
    'Error Class' = PROPERTY  
    'Error Code' = DATATYPE\_NOT\_SUPPORTED

### 7.3.2.X63.2.3 Invalid Property Writing in a Timer

Purpose: Verify the correct Result(-) when Timer\_State or Present\_Value is written with an invalid value.

Test Concept: Attempt writing of a Timer object T1 with a value outside the supported range and not zero being written to the Present\_Value property, or a value of other than IDLE written to the Timer\_State property. Verify the IUT returns the correct Result(-).

Configuration Requirements: The State\_Change\_Values property, if present, holds a valid configuration.

Test Steps:

1. TRANSMIT WriteProperty-Request,  
    'Object Identifier' = T1  
    'Property Identifier' = Present\_Value  
    'Property Value' = (a value that is outside the supported range and not zero)
2. RECEIVE BACnet-Error-PDU,  
    'Error Class' = PROPERTY  
    'Error Code' = VALUE\_OUT\_OF\_RANGE
3. TRANSMIT WriteProperty-Request,  
    'Object Identifier' = T1  
    'Property Identifier' = Timer\_State  
    'Property Value' = (a value other than IDLE)
4. RECEIVE BACnet-Error-PDU,  
    'Error Class' = PROPERTY  
    'Error Code' = VALUE\_OUT\_OF\_RANGE

### 7.3.2.X63.2.4 Expired Timer Ignores Writing Zero

Purpose: Verify the success of writes to Present\_Value of a Timer with an expiration value while in the EXPIRED state.

Test Concept: With a Timer object in the EXPIRED state, write 0 to Present\_Value and verify that the object remains in the EXPIRED state.



Configuration Requirements: The Timer object starts the test with Timer\_State equal to EXPIRED.

Test Steps:

1. VERIFY Timer\_State = EXPIRED
2. WRITE Present\_Value = 0
3. VERIFY Timer\_State = EXPIRED

#### **7.3.2.X63.2.5 Expired Timer Ignores Writing FALSE**

Purpose: Verify the success of writes to Timer\_Running property of a Timer with an expiration value while in the EXPIRED state.

Test Concept: With a Timer object in the EXPIRED state, write FALSE to Timer\_Running and verify that the object remains in the EXPIRED state.

Configuration Requirements: The Timer starts this test with Timer\_State equal to EXPIRED.

Test Steps:

1. VERIFY Timer\_State = EXPIRED
2. WRITE Timer\_Running = FALSE
3. VERIFY Timer\_State = EXPIRED

#### **7.3.2.X63.2.6 Idle Timer Ignores Writing Zero**

Purpose: Verify the success of writes to Present\_Value of a Timer with an expiration value while in the IDLE state.

Test Concept: With a Timer object in the IDLE state, write 0 to Present\_Value and verify that the object remains in the IDLE state.

Configuration Requirements: The Timer starts this test with Timer\_State equal to IDLE.

Test Steps:

1. VERIFY Timer\_State = IDLE
2. WRITE Present\_Value = 0
3. VERIFY Timer\_State = IDLE

#### **7.3.2.X63.2.7 Idle Timer Ignores Writing FALSE**

Purpose: Verify the success of writes to Timer\_Running property of a Timer with an expiration value while already in the IDLE state.

Test Concept: With a Timer object in the IDLE state, write FALSE to Timer\_Running and verify that the object remains in the IDLE state.

Configuration Requirements: The Timer starts this test with Timer\_State equal to IDLE.

Test Steps:

1. VERIFY Timer\_State = IDLE
2. WRITE Timer\_Running = FALSE
3. VERIFY Timer\_State = IDLE

### 7.3.2.X63.2.8 Idle Timer Ignores Writing IDLE

Purpose: Verify the success of writes to Timer\_State of a Timer with the reset value while already in the IDLE state.

Test Concept: With a Timer object in the IDLE state, write IDLE to Timer\_State and verify that the object remains in the IDLE state.

Configuration Requirements: The Timer starts this test with Timer\_State equal to IDLE.

Test Steps:

1. VERIFY Timer\_State = IDLE
2. WRITE Timer\_State = IDLE
3. VERIFY Timer\_State = IDLE

### 7.3.2.X63.2.9 Default\_Timeout Written Outside Supported Range

Purpose: Verify the correct Result(-) when Default\_Timeout is written with an invalid value.

Test Concept: Attempt writing Timer object, T1, with a value outside the supported range to the Default\_Timeout property. Verify the IUT returns the correct Result(-).

Configuration Requirements: If Default\_Timeout is not present or is not writable, this test shall be skipped.

Test Steps:

1. TRANSMIT WriteProperty-Request,  
    'Object Identifier' = T1  
    'Property Identifier' = Default\_Timeout  
    'Property Value' = (a value lower than Min\_Pres\_Value)
2. RECEIVE BACnet-Error-PDU,  
    'Error Class' = PROPERTY  
    'Error Code' = VALUE\_OUT\_OF\_RANGE
3. TRANSMIT WriteProperty-Request,  
    'Object Identifier' = T1  
    'Property Identifier' = Default\_Timeout  
    'Property Value' = (a value higher than Max\_Pres\_Value)
4. RECEIVE BACnet-Error-PDU,  
    'Error Class' = PROPERTY  
    'Error Code' = VALUE\_OUT\_OF\_RANGE

[Add new **Clause 7.3.2.X64**, p. 185]

[Reviewer Note: Clause 7.3.2 (Object Specific Tests) contains no tests for the Audit Log object type.]

## 7.3.2.X64 Audit Log Object Tests

### 7.3.2.X64.1 One Audit Log Holds all of an Objects History Test

Purpose: Ensure that, for any arbitrary object, there is at least one Audit Log into which all of the object's audit notifications are placed.

Test Concept: Send a sequence of audit notifications which contain entries for multiple objects to the IUT. At least some of the objects shall have multiple audit records in the sequence. For each object instance represented in the audit notifications sent to the IUT, verify that there is at least one Audit Log which contains all of the audit notifications for the object.

Configuration Requirements: S is a sequence of audit notifications which contain entries for multiple objects to the IUT where at least some of the objects shall have multiple audit records in the sequence.

Test Steps:

```

1. REPEAT AN = (each notification in S) DO {
    TRANSMIT UnconfirmedAuditNotification-Request,
    'Notifications' = AN
}
2. REPEAT O = (each object represented in S) DO {
    SO = (the sequence of notifications in S for object O)
    FOUND = (false)
    REPEAT AL = (each Audit Log object) DO {
        IF (AL contains all notifications in SO) THEN
            FOUND = (true)
    }
    IF (FOUND is false) THEN {
        ERROR "no audit log was found which contains all notifications for object"
    }
}

```

### 7.3.2.X64.2 Audit Notification Basic Combining Test

Purpose: Ensure that Audit Log objects correctly combine related audit notification records.

Test Concept: Send a sequence, SEQ1, of unrelated audit notifications to the IUT and verify that the notifications are not combined. Send a source audit notification, SN1, followed by a sequence, SEQ2, of unrelated audit notifications and verify that the notifications are not combined. Send a target audit notification, TN1, which should be combined with SN1. Verify that SN1 and TN1 are combined in the Audit Log. Repeat the process with new notifications but send the target notification before the source notification.'

Configuration Requirements: An audit log that should receive the combined SN/TN notification is AL. The Target Value and Current Value fields in SN and TN shall not be greater than 500 octets. D1 shall be the device sending the source notification and D2 shall be the device sending the target notification. It is acceptable if D1 is the same as D2.

Test Steps:

```

-- the source notification is sent before the target notification
1. REPEAT AN = (each notification in SEQ1) DO {
    TRANSMIT UnconfirmedAuditNotification-Request,
    SOURCE = (D1 or D2),
    'Notifications' = AN
}
2. TRANSMIT UnconfirmedAuditNotification-Request,
SOURCE = D1,
'Notifications' = SN1
3. REPEAT AN = (each notification in SEQ2) DO {
    SOURCE = (D1 or D2),
    TRANSMIT UnconfirmedAuditNotification-Request,
    'Notifications' = AN
}
4. TRANSMIT UnconfirmedAuditNotification-Request,
SOURCE = D2,
'Notifications' = TN1
5. CHECK(that no record exists in AL which is just SN1)

```

6. CHECK(that no record exists in AL which is just TN1)
  7. CHECK(that a record exists in AL which is the combination of SN1 and TN1)
  8. CHECK(that the combined record has all of the source and target fields provided in SN1 and TN1, and no more.)
- the target notification is sent before the source notification
9. REPEAT AN = (each notification in SEQ3) DO {
    - TRANSMIT UnconfirmedAuditNotification-Request,
    - SOURCE = (D1 or D2),
    - 'Notifications' = AN
  - }
  10. TRANSMIT UnconfirmedAuditNotification-Request,
  - SOURCE = D1,
  - 'Notifications' = TN2
  11. REPEAT AN = (each notification in SEQ4) DO {
    - SOURCE = (D1 or D2),
    - TRANSMIT UnconfirmedAuditNotification-Request,
    - 'Notifications' = AN
  - }
  12. TRANSMIT UnconfirmedAuditNotification-Request,
  - SOURCE = D2,
  - 'Notifications' = SN2
  13. CHECK(that no record exists in AL which is just SN2)
  14. CHECK(that no record exists in AL which is just TN2)
  15. CHECK(that a record exists in AL which is the combination of SN2 and TN2)
  16. CHECK(that the combined record has all of the source and target fields provided in SN2 and TN2, and no more.)

### 7.3.2.X64.3 Audit Notification Combining Failure Test

Purpose: Ensure that Audit Log objects correctly combine related audit notification records which indicate failed actions.

Test Concept: Send a source audit notification, SN. Send a target audit notification, TN, which should be combined with SN, and which indicates that the action failed. Verify that SN and TN are combined in the Audit Log.

Configuration Requirements: An audit log that should receive the combined SN/TN notification is AL. The Target Value and Current Value fields in SN and TN shall not be greater than 500 octets.

Test Steps:

1. TRANSMIT UnconfirmedAuditNotification-Request,
- 'Notifications' = SN
2. TRANSMIT UnconfirmedAuditNotification-Request,
- 'Notifications' = TN
3. CHECK(that no record exists in AL which is just SN)
4. CHECK(that no record exists in AL which is just TN)
5. CHECK(that a record exists in AL which is the combination of SN and TN)
6. CHECK(that the combined record has all of the source and target fields provided in SN and TN, and no more.)

### 7.3.2.X64.4 Audit Notification Non-combining Test

Purpose: Ensure that Audit Log objects correctly do not combine unrelated audit notification records.

Test Concept: Send a sequence of unrelated audit notifications to the IUT each differing by 1 field in the matching criteria. Verify that the notifications are not combined.

Configuration Requirements: SEQ is a sequence of audit notifications where each record differs from the previous record by 1 field. For the sequence, SN is a source notification with user-id, user-role, target-value fields, and without source-comment field, and TN is the matching target notification with user-id, user-role, target-value fields. The sequence is:

```
{
(SN),
(SN but with source-comment field),
(TN with differing operation-source),
(TN with differing operation),
(TN with differing invoke-id),
(TN with differing target-device),
(TN with differing target-property),
(TN with differing user-id),
(TN with differing user-role),
(TN with differing target-value),
(TN with target-timestamp equal to source-timestamp + APDU_Timeout * 3)
}
```

Test Steps:

1. REPEAT AN = (each notification in SEQ) DO {  
TRANSMIT UnconfirmedAuditNotification-Request,  
'Notifications' = AN  
}
2. REPEAT AN = (each notification in SEQ) DO {  
CHECK(that AN is in an Audit Log and is not combined)  
}

### 7.3.2.X64.5 Audit Notification Combining Duplicate Test

Test Concept: Send a source audit notification SN. Verify it is placed in the log. Send a sequence, SEQ1, of unrelated audit notifications and verify SN is not combined with any. Resend SN and verify that SN was not re-added to the log. Send a target audit notification, TN, which should be combined with SN. Verify that SN and TN are combined in the Audit Log. Send a sequence, SEQ2, of unrelated audit notifications. Resend TN and verify that TN was not re-added to the log.

Test Steps:

1. TRANSMIT UnconfirmedAuditNotification-Request,  
'Notifications' = SN
2. CHECK(that SN is in the Audit Log)
3. REPEAT AN = (each notification in SEQ1) DO {  
TRANSMIT UnconfirmedAuditNotification-Request,  
'Notifications' = AN  
}
4. CHECK(that SN is in the Audit Log and is not combined)
5. TRANSMIT UnconfirmedAuditNotification-Request,  
'Notifications' = SN
6. CHECK(that SN is in the Audit Log only once and is at its original position)
7. TRANSMIT UnconfirmedAuditNotification-Request,  
'Notifications' = TN
8. CHECK(that SN is in the Audit Log only once, is combined with TN, and is at its original position)
9. CHECK(that the combined record has all of the source and target fields provided in SN and TN, and no more.)
10. REPEAT AN = (each notification in SEQ2) DO {  
TRANSMIT UnconfirmedAuditNotification-Request,  
'Notifications' = AN  
}
11. TRANSMIT UnconfirmedAuditNotification-Request,  
'Notifications' = TN
12. CHECK(that TN is in the Audit Log only once, is combined with SN and is at SN's original position)

### 7.3.2.X64.6 Audit Notification Combining Target Value Preference Test

Purpose: Ensure that Audit Log objects use the Current Value from a target notification when it is provided in both the source and target notifications.

Test Concept: Send a target audit notification TN1 which includes the Current Value field with a value CV1-T. Send a source audit notification, SN1, which should be combined with TN1, and which contains a Current Value field with a value CV1-S (CV1-S is different than CV1-T). Verify that SN1 and TN1 are combined in the Audit Log and that the Current Value in the log uses CV1-T. Repeat the steps sending a target notification TN2 before the source notification SN2 where CV2-S is different than CV2-T.

Test Steps:

1. TRANSMIT UnconfirmedAuditNotification-Request,  
'Notifications' = TN1
2. TRANSMIT UnconfirmedAuditNotification-Request,  
'Notifications' = SN1
3. CHECK(that TN1 is in the Audit Log only once, is combined with SN1, and that target-value is CV1-T)
4. TRANSMIT UnconfirmedAuditNotification-Request,  
'Notifications' = SN2
5. TRANSMIT UnconfirmedAuditNotification-Request,  
'Notifications' = TN2
6. CHECK(that SN2 is in the Audit Log only once, is combined with TN2, and that target-value is CV2-T)

### 7.3.2.X64.7 Accepts Audit Notifications from an Audit Forwarder Test

Purpose: Ensure that Audit Log accepts forwarded audit notifications.

Test Concept: The notification forwarder, AF1, sends a forwarded source notification, SN1, from the original sending device D1, to the IUT. Verify that the IUT places the notification in the Audit Log. AF1 then sends a forwarded target notification, TN1, from the original target device D2, to the IUT. Verify that the IUT combines the target with the source notification.

Configuration Requirements: The test network consists of a source device D1, a target device D2, and a notification forwarder, AF1.

Test Steps:

1. TRANSMIT UnconfirmedAuditNotification-Request,  
SOURCE = AF1,  
'Notifications' = SN1
2. TRANSMIT UnconfirmedAuditNotification-Request,  
SOURCE = AF1,  
'Notifications' = TN1
3. CHECK(that SN1 and TN1 are combined in the Audit Log)

### 7.3.2.X64.8 Hierarchical Logging Test

Purpose: Ensure that an Audit Log configured with a parent correctly forwards notifications to the parent log.

Test Concept: An Audit Log, AL1, configured to reference a parent log located in another device, is sent a sequence of audit notifications. Within the sequence will some notifications which should be combined and some which should not be combined. Verify that the IUT forwards the notifications to the parent before the vendor specified maximum forwarding delay.

Configuration Requirements: The Audit Log, AL1, is configured with a Member\_Of set to AL2, where AL2 is in the TD. AL1's Delete\_On\_Forward shall be set to FALSE. SEQ is a sequence of audit notifications where at least 2 are related and should be combined.

Notes to Tester: The standard does not provide guidance on how long an Audit Log object has before it must forward audit notifications to its parent. As such, the vendor is allowed to specify the maximum time as long as it is not unreasonable (delays on the order of days are clearly unreasonable; delays on the order of minutes are clearly acceptable).

Notes to Tester: When receiving notifications from the IUT, those notifications which should be combined, may be sent combined or not at the IUT's discretion.

Test Steps:

1. REPEAT AN = (each notification in SEQ) DO {  
     TRANSMIT UnconfirmedAuditNotification-Request,  
     SOURCE =       (a value appropriate to the notification),  
     'Notifications' =       AN  
   }  
 }
2. IF the IUT is configured to send unconfirmed audit notifications THEN {  
     BEFORE **Audit Notification Forwarder Fail Time**  
     RECEIVE UnconfirmedAuditNotification-Request,  
     'Notifications' =       (one or more of the notifications from SEQ)  
 } ELSE {  
     BEFORE **Audit Notification Forwarder Fail Time**  
     RECEIVE ConfirmedAuditNotification-Request,  
     'Notifications' =       (one or more of the notifications from SEQ)  
     TRANSMIT BACnet-SimpleACK-PDU  
 }  
 }
3. WHILE (not all notifications in SEQ have been sent by the IUT) {  
     IF the IUT is configured to send unconfirmed audit notifications THEN {  
         RECEIVE UnconfirmedAuditNotification-Request,  
         'Notifications' =       (one or more of the as yet unreceived notifications from SEQ)  
     } ELSE {  
         RECEIVE ConfirmedAuditNotification-Request,  
         'Notifications' =       (one or more of the as yet unreceived notifications from SEQ)  
         TRANSMIT BACnet-SimpleACK-PDU  
     }  
 }  
 }
4. CHECK(that the notifications in SEQ are still in AL1)
5. CHECK(that the notifications in SEQ which are to be combined are combined in AL1)

[Add new **Clause 7.3.2.X65**, p. 185]

[Reviewer Note: Clause 7.3.2 (Object Specific Tests) contains no tests for the Audit Reporter object type.]

### 7.3.2.X65 Audit Reporter Object Tests

#### 7.3.2.X65.4 Target Audit Reporting - Basic Notification Test

Purpose: Verify that target audit notifications are properly formed.

Test Concept: The IUT is made to send a target audit notification. It is verified that the target fields are present, no source fields are present, and other notification fields represent the auditable operation performed.

Configuration Requirements: The IUT is configured to report all auditable operations. The IUT is configured so that the notification will be reported through Audit Reporter AR.

Test Steps:

1. MAKE(perform an auditable operation, O, on IUT)
2. IF the IUT is configured to send unconfirmed audit notifications THEN {
 

BEFORE AR.Maximum\_Send\_Delay + Notification Fail Time  
 RECEIVE UnconfirmedAuditNotification-Request,  
 'Notifications' = ( {
 

-- source-timestamp absent  
 target-timestamp = (IUT's local time),  
 source-device = TD,  
 -- source-object absent  
 operation = O,  
 -- source-comment absent  
 target-comment = (any valid value, or absent unless O is GENERAL),  
 invoke-id = (the invoke id from the operation, or absent if it was unconfirmed),  
 source-user-id = (the value from the operation if provided, otherwise absent),  
 source-user-role = (the value from the operation if provided, otherwise absent),  
 target-device = IUT,  
 target-object = (the target object or absent if the target is not an object),  
 target-property = (the target property or absent if the target is not a property),  
 target-priority = (the priority supplied, or absent if the target is not a property.  
     shall be 16 or absent if no priority supplied and the target is a  
     property),  
 target-value = (the target value or absent if no target value for the operation.  
     may be absent if the value size is larger than 32 encoded  
     octets),  
 current-value = (the value before the op or absent if no target value.  
     may be absent if the value size is larger than 32 encoded  
     octets),  
 result = (the reason for failure if the op failed, otherwise absent)  
 } )
- } ELSE {
 

BEFORE AR.Maximum\_Send\_Delay + Notification Fail Time  
 RECEIVE ConfirmedAuditNotification-Request,  
 'Notifications' = ( {
 

-- source-timestamp absent  
 target-timestamp = (IUT's local time),  
 source-device = TD,  
 -- source-object absent  
 operation = O,  
 -- source-comment absent  
 target-comment = (any valid value, or absent unless O is GENERAL),  
 invoke-id = (the invoke id from the operation, or absent if it was unconfirmed),  
 source-user-id = (the value from the operation if provided, otherwise absent),  
 source-user-role = (the value from the operation if provided, otherwise absent),  
 target-device = IUT,  
 target-object = (the target object or absent if the target is not an object),  
 target-property = (the target property or absent if the target is not a property),  
 target-priority = (the priority supplied, or absent if the target is not a property.  
     shall be 16 or absent if no priority supplied and the target is a  
     property),  
 target-value = (the target value or absent if no target value for the operation.  
     may be absent if the value size is larger than 32 encoded  
     octets),  
 current-value = (the value before the op or absent if no target value.  
     may be absent if the value size is larger than 32 encoded



```

        octets),
        result = (the reason for failure if the op failed, otherwise absent)
    } )
    TRANSMIT BACnet-SimpleACK-PDU
}

```

### 7.3.2.X65.5 Target Audit Reporting - Unconfirmed Service Operation Test

Purpose: Verify that target audit notifications for unconfirmed services do not contain InvokeId information.

Test Concept: An auditable unconfirmed service is performed on the IUT, and it is verified that the resulting target audit notification does not contain an InvokeId, and other notification fields represent the auditable operation performed.

Configuration Requirements: The IUT is configured to report audit notifications for an unconfirmed service. If the IUT does not support audit reporting for any unconfirmed services, this test shall be skipped. The IUT is configured so that the notification will be reported through Audit Reporter AR.

Test Steps:

1. MAKE(perform an auditable operation, O, which uses an unconfirmed service, on IUT)
2. IF the IUT is configured to send unconfirmed audit notifications for operation O THEN {
 

BEFORE AR.Maximum\_Send\_Delay + Notification Fail Time  
 RECEIVE UnconfirmedAuditNotification-Request,  
 'Notifications' = ( {
 

-- source-timestamp absent  
 target-timestamp = (IUT's local time),  
 source-device = TD,  
 -- source-object absent  
 operation = O,  
 -- source-comment absent  
 target-comment = (any valid value, or absent unless O is GENERAL),  
 -- invoke-id absent,  
 source-user-id = (the value from the operation if provided, otherwise absent),  
 source-user-role = (the value from the operation if provided, otherwise absent),  
 target-device = IUT,  
 target-object = (the target object or absent if the target is not an object),  
 target-property = (the target property or absent if the target is not a property),  
 target-priority = (the priority supplied, or absent if the target is not a property.  
     shall be 16 or absent if no priority supplied and the target is a  
     property),  
 target-value = (the target value or absent if no target value for the operation.  
     may be absent if the value size is larger than 32 encoded  
     octets),  
 current-value = (the value before the op or absent if no target value.  
     may be absent if the value size is larger than 32 encoded  
     octets),  
 result = (the reason for failure if the op failed, otherwise absent)

 } )
- } ELSE {
 

BEFORE AR.Maximum\_Send\_Delay + Notification Fail Time  
 RECEIVE ConfirmedAuditNotification-Request,  
 'Notifications' = ( {
 

-- source-timestamp absent  
 target-timestamp = (IUT's local time),  
 source-device = TD,  
 -- source-object absent  
 operation = O,  
 -- source-comment absent

```

target-comment = (any valid value, or absent unless O is GENERAL),
-- invoke-id absent,
source-user-id = (the value from the operation if provided, otherwise absent),
source-user-role = (the value from the operation if provided, otherwise absent),
target-device = IUT,
target-object = (the target object or absent if the target is not an object),
target-property = (the target property or absent if the target is not a property),
target-priority = (the priority supplied, or absent if the target is not a property.
    shall be 16 or absent if no priority supplied and the target is a
    property),
target-value = (the target value or absent if no target value for the operation.
    may be absent if the value size is larger than 32 encoded
    octets),
current-value = (the value before the op or absent if no target value.
    may be absent if the value size is larger than 32 encoded
    octets),
result = (the reason for failure if the op failed, otherwise absent)
    } )
    TRANSMIT BACnet-SimpleACK-PDU
}

```

### 7.3.2.X65.6 Target Audit Reporting - Confirmed Service Operation Audit Notification

Purpose: Verify that target audit notifications for confirmed services contain InvokeId information.

Test Concept: An auditable confirmed service is performed on the IUT, and it is verified that the resulting target audit notification contains an InvokeId, and other notification fields represent the auditable operation performed.

Configuration Requirements: The IUT is configured to report audit notifications for an unconfirmed service. The IUT is configured so that the notification will be reported through Audit Reporter AR.

Test Steps:

1. MAKE(perform an auditable operation, O, which uses an unconfirmed service, on IUT)
2. IF the IUT is configured to send unconfirmed audit notifications for operation O THEN {
 BEFORE AR.Maximum\_Send\_Delay + Notification Fail Time
 RECEIVE UnconfirmedAuditNotification-Request,
 'Notifications' = ( { -- source-timestamp absent
 target-timestamp = (IUT's local time),
 source-device = TD,
 -- source-object absent
 operation = O,
 -- source-comment absent
 target-comment = (any valid value, or absent unless O is GENERAL),
 -- invoke-id absent,
 source-user-id = (the value from the operation if provided, otherwise absent),
 source-user-role = (the value from the operation if provided, otherwise absent),
 target-device = IUT,
 target-object = (the target object or absent if the target is not an object),
 target-property = (the target property or absent if the target is not a property),
 target-priority = (the priority supplied, or absent if the target is not a property. shall be 16 or absent if no priority supplied and the target is a property),
 target-value = (the target value or absent if no target value for the operation. may be absent if the value size is larger than 32 encoded octets),
 } )
 }

```

        current-value = (the value before the op or absent if no target value.
                        may be absent if the value size is larger than 32 encoded
                        octets),
        result = (the reason for failure if the op failed, otherwise absent)
    } )
} ELSE {
    BEFORE AR.Maximum_Send_Delay + Notification Fail Time
    RECEIVE ConfirmedAuditNotification-Request,
    'Notifications' = ( { -- source-timestamp absent
        target-timestamp = (IUT's local time),
        source-device = TD,
        -- source-object absent
        operation = O,
        -- source-comment absent
        target-comment = (any valid value, or absent unless O is GENERAL),
        -- invoke-id absent,
        source-user-id = (the value from the operation if provided, otherwise
            absent),
        source-user-role = (the value from the operation if provided, otherwise
            absent),
        target-device = IUT,
        target-object = (the target object or absent if the target is not an object),
        target-property = (the target property or absent if the target is not a
            property),
        target-priority = (the priority supplied, or absent if the target is not a
            property. shall be 16 or absent if no priority supplied and the
            target is a property),
        target-value = (the target value or absent if no target value for the
            operation. may be absent if the value size is larger than 32
            encoded octets),
        current-value = (the value before the op or absent if no target value.
                        may be absent if the value size is larger than 32 encoded
                        octets),
        result = (the reason for failure if the op failed, otherwise absent)
    } )
    TRANSMIT BACnet-SimpleACK-PDU
}

```

### 7.3.2.X65.7 Target Audit Reporting - Operations with Priority Test

Purpose: Verify that target audit notifications which for writes which convey a priority include the priority in the notification.

Test Concept: An auditable write, which includes a priority, is performed on a commandable object in the IUT and it is verified that the resulting target audit notification contains a priority, and other notification fields represent the auditable operation performed.

Configuration Requirements: The IUT is configured to report audit notifications for writes on a commandable object, O1. The IUT is configured so that the notification will be reported through Audit Reporter AR using unconfirmed notifications. If the IUT does not support the Priority\_Array property in any object for which audit reporting can be configured, this test shall be skipped.

Test Steps:

1. TRANSMIT WriteProperty-Request,
  - 'Invoke Id' = I,
  - 'Object Identifier' = O1,
  - 'Property Identifier' = Present\_Value,
  - 'Property Value' = (V: any valid value),
  - 'Priority' = (PRIO: a priority in the range 1 - 15)

```

2. BEFORE Internal Processing Fail Time
   RECEIVE BACnet-SimpleACK-PDU
   |
   | (BACnet-Error-PDU,
   |   'Error Type' = (E : any error)
   | )
3. IF the IUT is configured to send unconfirmed audit notifications THEN {
   BEFORE AR.Maximum_Send_Delay + Notification Fail Time
   RECEIVE UnconfirmedAuditNotification-Request,
   'Notifications' = ( {
     -- source-timestamp absent
     target-timestamp = (IUT's local time),
     source-device = TD,
     -- source-object absent
     operation = WRITE,
     -- source-comment absent
     target-comment = (any valid value, or absent),
     invoke-id = I,
     source-user-id = (the value from the operation if provided, otherwise absent),
     source-user-role = (the value from the operation if provided, otherwise absent),
     target-device = IUT,
     target-object = O1,
     target-property = Present_Value,
     target-priority = PRIO,
     target-value = V,
     current-value = (the value before the write. may be absent if the value size is
       larger than 32 encoded octets),
     result = (E, if the op failed, otherwise absent)
   } )
} ELSE {
  BEFORE AR.Maximum_Send_Delay + Notification Fail Time
  RECEIVE ConfirmedAuditNotification-Request,
  'Notifications' = ( {
    -- source-timestamp absent
    target-timestamp = (IUT's local time),
    source-device = TD,
    -- source-object absent
    operation = WRITE,
    -- source-comment absent
    target-comment = (any valid value, or absent),
    invoke-id = I,
    source-user-id = (the value from the operation if provided, otherwise absent),
    source-user-role = (the value from the operation if provided, otherwise absent),
    target-device = IUT,
    target-object = O1,
    target-property = Present_Value,
    target-priority = PRIO,
    target-value = V,
    current-value = (the value before the write. may be absent if the value size is
      larger than 32 encoded octets),
    result = (E, if the op failed, otherwise absent)
  } )
  TRANSMIT BACnet-SimpleACK-PDU
}

```

### 7.3.2.X65.8 Target Audit Reporting - Target\_Value and Current\_Value Test

Purpose: Verify that the IUT reports target values and current values, when the audited operation contains a value (such as for writes).

Test Concept: An auditable operation, which contains a value, is performed on object O1 and property P1. The resulting audit notification is verified to contain the provided value and the value before the operation.

Configuration Requirements: The IUT is configured to report all audit notifications. If possible, a property which is not changing shall be the target of the operation so that the current value field can be validated. If the IUT does not have any objects which support reporting of operation which contain target value, this test shall be skipped. AR is the Audit Report through which O1 reports audit notifications.

Test Steps:

```

1. IF P1 is not changing outside of the operation THEN {
    READ IV = O1, P1
2. MAKE(perform an auditable operation, on O1, P1, which provides a target value, V,
    which is less than 32 octets in size)
3. IF the IUT is configured to send unconfirmed audit notifications THEN {
    BEFORE AR.Maximum_Send_Delay + Notification Fail Time
    RECEIVE UnconfirmedAuditNotification-Request,
    'Notifications' = ({
        -- source-timestamp absent
        target-timestamp = (IUT's local time),
        source-device = TD,
        -- source-object absent
        operation = (the operation performed),
        -- source-comment absent
        target-comment = (any valid value, or absent),
        invoke-id = (the invoke id from the operation, or absent if it was
            unconfirmed),
        source-user-id = (the value from the operation if provided, otherwise
            absent),
        source-user-role = (the value from the operation if provided, otherwise
            absent),
        target-device = IUT,
        target-object = O1,
        target-property = P1,
        target-priority = (the priority from the operation, or absent if 16 or not
            provided in the operation),
        target-value = V,
        current-value = (CV: any valid value),
        result = (E, if the op failed, otherwise absent)
    } )
} ELSE {
    BEFORE AR.Maximum_Send_Delay + Notification Fail Time
    RECEIVE ConfirmedAuditNotification-Request,
    'Notifications' = ({
        -- source-timestamp absent
        target-timestamp = (IUT's local time),
        source-device = TD,
        -- source-object absent
        operation = (the operation performed),
        -- source-comment absent
        target-comment = (any valid value, or absent),
        invoke-id = (the invoke id from the operation, or absent if it was
            unconfirmed),
        source-user-id = (the value from the operation if provided, otherwise
            absent),
        source-user-role = (the value from the operation if provided, otherwise

```

```

        absent),
        target-device = IUT,
        target-object = O1,
        target-property = P1,
        target-priority = (the priority from the operation, or absent if 16 or not
            provided in the operation),
        target-value = V,
        current-value = (CV: any valid value),
        result = (E, if the op failed, otherwise absent)
    } )
    TRANSMIT BACnet-SimpleACK-PDU
}
4. IF the P1 is not changing outside of the operation THEN
    CHECK(CV equals IV)

```

### 7.3.2.X65.9 Target Audit Reporting - Error Audit Notification Test

Purpose: Verify that operations that fail are properly reported in audit notifications.

Test Concept: An auditable operation, which will fail with a Result(-) or Result(+) with error information is performed on the IUT. It is verified that an audit notification is sent which contains the error that occurred. The auditable operation performed shall be one for which the IUT will report failures via audit notifications.

Configuration Requirements: The IUT is configured to report all audit notifications.

Test Steps:

1. MAKE(perform an auditable operation, O, on IUT which will fail (via return of a BACnetErrorPDU, or a Result(+) with error information)
2. IF the IUT is configured to send unconfirmed audit notifications for operation O THEN {
 BEFORE AR.Maximum\_Send\_Delay + Notification Fail Time
 RECEIVE UnconfirmedAuditNotification-Request,
 'Notifications' = ({
 -- source-timestamp absent
 target-timestamp = (IUT's local time),
 source-device = TD,
 -- source-object absent
 operation = O,
 -- source-comment absent
 target-comment = (any valid value, or absent),
 invoke-id = (the invoke id from the operation, or absent if it was unconfirmed),
 source-user-id = (the value from the operation if provided, otherwise absent),
 source-user-role = (the value from the operation if provided, otherwise absent),
 target-device = IUT,
 target-object = (the target object or absent if the target is not an object),
 target-property = (the target property or absent if the target is not a property),
 target-priority = (the priority supplied, or absent if the target is not a property. shall be 16 or absent if no priority supplied and the target is a property),
 target-value = (the target value or absent if no target value for the operation. may be absent if the value size is larger than 32 encoded octets),
 current-value = (the value before the op or absent if no target value. may be absent if the value size is larger than 32 encoded
 })
 }

```

        octets),
        result = (the error reported for the operation)
    } )
} ELSE {
    BEFORE AR.Maximum_Send_Delay + Notification Fail Time
    RECEIVE ConfirmedAuditNotification-Request,
    'Notifications' = ({
        -- source-timestamp absent
        target-timestamp = (IUT's local time),
        source-device = TD,
        -- source-object absent
        operation = (the operation performed),
        -- source-comment absent
        target-comment = (any valid value, or absent),
        invoke-id = (the invoke id from the operation, or absent if it was
            unconfirmed),
        source-user-id = (the value from the operation if provided, otherwise
            absent),
        source-user-role = (the value from the operation if provided, otherwise
            absent),
        target-device = IUT,
        target-object = (the target object or absent if the target is not an object),
        target-property = (the target property or absent if the target is not a
            property),
        target-priority = (the priority supplied, or absent if the target is not a
            property. shall be 16 or absent if no priority supplied and the
            target is a property),
        target-value = (the target value or absent if no target value for the
            operation. may be absent if the value size is larger than 32
            encoded octets),
        current-value = (the value before the op or absent if no target value.
            may be absent if the value size is larger than 32 encoded
            octets),
        result = (the error reported for the operation)
    } )
    TRANSMIT BACnet-SimpleACK-PDU
}

```

### 7.3.2.X65.10 Target Audit Reporting - GENERAL Operation Test

Purpose: Verify that GENERAL operation audit notifications contain a Target Comment.

Test Concept: An auditable GENERAL operation is performed on the IUT. It is verified that an audit notification is sent which contains the error that occurred.

Configuration Requirements: The IUT is configured to report all audit notifications. If the IUT does not generate GENERAL audit notifications, this test shall be skipped.

Test Steps:

1. MAKE(make the IUT generate a GENERAL audit notification)
2. IF the IUT is configured to send unconfirmed audit notifications THEN {
 

```

                BEFORE AR.Maximum_Send_Delay + Notification Fail Time
                RECEIVE UnconfirmedAuditNotification-Request,
                'Notifications' = ({
                    -- source-timestamp absent
                    target-timestamp = (IUT's local time),
                    source-device = TD,
                    -- source-object absent

```

```

        operation = GENERAL,
        -- source-comment absent
        target-comment = (any valid value),
        invoke-id = (the invoke id from the operation, or absent if it was
            unconfirmed),
        source-user-id = (the value from the operation if provided, otherwise
            absent),
        source-user-role = (the value from the operation if provided, otherwise
            absent),
        target-device = IUT,
        target-object = (the target object or absent if the target is not an object),
        target-property = (the target property or absent if the target is not a
            property),
        target-priority = (the priority supplied, or absent if the target is not a
            property. shall be 16 or absent if no priority supplied and the
            target is a property),
        target-value = (the target value or absent if no target value for the
            operation. may be absent if the value size is larger than 32
            encoded octets),
        current-value = (the value before the op or absent if no target value.
            may be absent if the value size is larger than 32 encoded
            octets),
        result = (the reason for failure if the op failed, otherwise absent)
    } )
} ELSE {
    BEFORE AR.Maximum_Send_Delay + Notification Fail Time
    RECEIVE ConfirmedAuditNotification-Request,
    'Notifications' = ({
        -- source-timestamp absent
        target-timestamp = (IUT's local time),
        source-device = TD,
        -- source-object absent
        operation = GENERAL,
        -- source-comment absent
        target-comment = (any valid value),
        invoke-id = (the invoke id from the operation, or absent if it was
            unconfirmed),
        source-user-id = (the value from the operation if provided, otherwise
            absent),
        source-user-role = (the value from the operation if provided, otherwise
            absent),
        target-device = IUT,
        target-object = (the target object or absent if the target is not an object),
        target-property = (the target property or absent if the target is not a
            property),
        target-priority = (the priority supplied, or absent if the target is not a
            property. shall be 16 or absent if no priority supplied and the
            target is a property),
        target-value = (the target value or absent if no target value for the
            operation. may be absent if the value size is larger than 32
            encoded octets),
        current-value = (the value before the op or absent if no target value.
            may be absent if the value size is larger than 32 encoded
            octets),
        result = (the reason for failure if the op failed, otherwise absent)
    } )
    TRANSMIT BACnet-SimpleACK-PDU
}

```



### 7.3.2.X65.11 Source Audit Reporting - Basic Notification Test

Purpose: Verify that source audit notifications are properly formed.

Test Concept: The IUT is made to send a source audit notification. It is verified that the source fields are present, no target fields are present, and other notification fields represent the auditable operation performed.

Configuration Requirements: The IUT is configured to report all auditable source operations. The IUT is configured with AR as the source Audit Reporter object.

Test Steps:

1. MAKE(the IUT perform an auditable operation, O, on the TD)
2. IF the IUT is configured to send unconfirmed audit notifications THEN {
 

BEFORE AR.Maximum\_Send\_Delay + Notification Fail Time  
 RECEIVE UnconfirmedAuditNotification-Request,  
 'Notifications' = ( {  
   source-timestamp = (IUT's local time),  
   -- target-timestamp absent  
   source-device = IUT,  
   source-object = (the object which initiated the op or absent if not initiated by an  
                   object),  
   operation = O,  
   source-comment = (any valid value or absent),  
   -- target-comment absent  
   invoke-id = (the invoke id from the operation, or absent if it was unconfirmed),  
   source-user-id = (the value from the operation if provided, otherwise absent),  
   source-user-role = (the value from the operation if provided, otherwise absent),  
   target-device = TD,  
   target-object = (the target object or absent if the target is not an object),  
   target-property = (the target property or absent if the target is not a property),  
   target-priority = (the priority supplied, or absent if the target is not a property.  
                     shall be 16 or absent if no priority supplied and the target is a  
                     property),  
   target-value = (the target value or absent if no target value for the operation.  
                   may be absent if the value size is larger than 32 encoded  
                   octets),  
   current-value = (the value before the op if the op targeted a property, or absent.  
                   May be absent even if targeting a property),  
   result = (the reason for failure if the op failed, otherwise absent)  
   } )
- } ELSE {
 

BEFORE AR.Maximum\_Send\_Delay + Notification Fail Time  
 RECEIVE ConfirmedAuditNotification-Request,  
 'Notifications' = ( {  
   source-timestamp = (IUT's local time),  
   -- target-timestamp absent  
   source-device = IUT,  
   source-object = (the object which initiated the op or absent if not initiated by an  
                   object),  
   operation = O,  
   source-comment = (any valid value or absent),  
   -- target-comment absent  
   invoke-id = (the invoke id from the operation, or absent if it was unconfirmed),  
   source-user-id = (the value from the operation if provided, otherwise absent),  
   source-user-role = (the value from the operation if provided, otherwise absent),  
   target-device = TD,  
   target-object = (the target object or absent if the target is not an object),

```

    target-property = (the target property or absent if the target is not a property),
    target-priority = (the priority supplied, or absent if the target is not a property.
        shall be 16 or absent if no priority supplied and the target is a
        property),
    target-value = (the target value or absent if no target value for the operation.
        may be absent if the value size is larger than 32 encoded
        octets),
    current-value = (the value before the op if the op targeted a property, or absent.
        May be absent even if targeting a property),
    result = (the reason for failure if the op failed, otherwise absent)
    } )
    TRANSMIT BACnet-SimpleACK-PDU
}

```

### 7.3.2.X65.12 Source Audit Reporting - Same Device Notification Test

Purpose: Verify that source and target fields in audit notifications are performed by the IUT on the IUT.

Test Concept: The IUT is made to perform an auditable operation on itself. It is verified that the source and target fields are present, and other notification fields represent the auditable operation performed.

Configuration Requirements: The IUT is configured to report all auditable operations. The IUT is configured with AR as the source Audit Reporter object. If the IUT is unable to perform an auditable operation on itself, this test shall be skipped.

Notes to Tester: The IUT is allowed to send the notifications as 2 separate notifications in the same audit notification message, or in separate messages. When sending separate notifications, one shall be a correctly formed target notification and the other a correctly formed source notification for the operation performed

Test Steps:

```

1. MAKE(the IUT perform an auditable operation, O, on the itself)
2. IF the IUT is configured to send unconfirmed audit notifications THEN {
    BEFORE AR.Maximum_Send_Delay + Notification Fail Time
    RECEIVE UnconfirmedAuditNotification-Request,
    'Notifications' = ( {
        source-timestamp = (T: IUT's local time),
        target-timestamp = T,
        source-device = IUT,
        source-object = (the object which initiated the op or absent if not initiated by an
            object),
        operation = O,
        source-comment = (any valid value or absent),
        target-comment = (any valid value or absent),
        -- invoke-id absent
        source-user-id = (the value from the operation if provided, otherwise absent),
        source-user-role = (the value from the operation if provided, otherwise absent),
        target-device = IUT,
        target-object = (the target object or absent if the target is not an object),
        target-property = (the target property or absent if the target is not a property),
        target-priority = (the priority supplied, or absent if the target is not a property.
            shall be 16 or absent if no priority supplied and the target is a
            property),
        target-value = (the target value or absent if no target value for the operation.
            may be absent if the value size is larger than 32 encoded
            octets),
        current-value = (the value before the op if the op targeted a property, or absent),
        result = (the reason for failure if the op failed, otherwise absent)
    } )
} ELSE {

```

```

BEFORE AR.Maximum_Send_Delay + Notification Fail Time
RECEIVE ConfirmedAuditNotification-Request,
  'Notifications' = ( {
    source-timestamp = (T: IUT's local time),
    target-timestamp = T,
    source-device = IUT,
    source-object = (the object which initiated the op or absent if not initiated by an
                     object),
    operation = O,
    source-comment = (any valid value or absent),
    target-comment = (any valid value or absent),
    -- invoke-id absent
    source-user-id = (the value from the operation if provided, otherwise absent),
    source-user-role = (the value from the operation if provided, otherwise absent),
    target-device = IUT,
    target-object = (the target object or absent if the target is not an object),
    target-property = (the target property or absent if the target is not a property),
    target-priority = (the priority supplied, or absent if the target is not a property.
                      shall be 16 or absent if no priority supplied and the target is a
                      property),
    target-value = (the target value or absent if no target value for the operation.
                   may be absent if the value size is larger than 32 encoded
                   octets),
    current-value = (the value before the op if the op targeted a property, or absent),
    result = (the reason for failure if the op failed, otherwise absent)
  } )
TRANSMIT BACnet-SimpleACK-PDU
}

```

### 7.3.2.X65.13 Source Audit Reporting - Unconfirmed Service Operation Test

Purpose: Verify that source audit notifications for unconfirmed services do not contain InvokeId information.

Test Concept: An auditable unconfirmed service is performed by the IUT, and it is verified that the resulting source audit notification does not contain an InvokeId, and other notification fields represent the auditable operation performed.

Configuration Requirements: The IUT is configured to report source audit notifications for an unconfirmed service. If the IUT does not support source audit reporting for any unconfirmed services, this test shall be skipped. The IUT is configured with AR as the source Audit Reporter object.

Test Steps:

1. MAKE(the IUT perform an auditable operation, O, on the TD which uses an unconfirmed service)
2. IF the IUT is configured to send unconfirmed audit notifications THEN {
 

```

BEFORE AR.Maximum_Send_Delay + Notification Fail Time
RECEIVE UnconfirmedAuditNotification-Request,
  'Notifications' = ( {
    source-timestamp = (IUT's local time),
    -- target-timestamp absent
    source-device = IUT,
    source-object = (the object which initiated the op or absent if not initiated by an
                     object),
    operation = O,
    source-comment = (any valid value or absent),
    -- target-comment absent
    -- invoke-id absent
    source-user-id = (the value from the operation if provided, otherwise absent),

```

```

    source-user-role = (the value from the operation if provided, otherwise absent),
    target-device = TD,
    target-object = (the target object or absent if the target is not an object),
    target-property = (the target property or absent if the target is not a property),
    target-priority = (the priority supplied, or absent if the target is not a property.
        shall be 16 or absent if no priority supplied and the target is a
        property),
    target-value = (the target value or absent if no target value for the operation.
        may be absent if the value size is larger than 32 encoded
        octets),
    current-value = (the value before the op if the op targeted a property, or absent.
        May be absent even if targeting a property),
    result = (the reason for failure if the op failed, otherwise absent)
  } )
} ELSE {
  BEFORE AR.Maximum_Send_Delay + Notification Fail Time
  RECEIVE ConfirmedAuditNotification-Request,
  'Notifications' = ( {
    source-timestamp = (IUT's local time),
    -- target-timestamp absent
    source-device = IUT,
    source-object = (the object which initiated the op or absent if not initiated by an
        object),
    operation = O,
    source-comment = (any valid value or absent),
    -- target-comment absent
    -- invoke-id absent
    source-user-id = (the value from the operation if provided, otherwise absent),
    source-user-role = (the value from the operation if provided, otherwise absent),
    target-device = TD,
    target-object = (the target object or absent if the target is not an object),
    target-property = (the target property or absent if the target is not a property),
    target-priority = (the priority supplied, or absent if the target is not a property.
        shall be 16 or absent if no priority supplied and the target is a
        property),
    target-value = (the target value or absent if no target value for the operation.
        may be absent if the value size is larger than 32 encoded
        octets),
    current-value = (the value before the op if the op targeted a property, or absent.
        May be absent even if targeting a property),
    result = (the reason for failure if the op failed, otherwise absent)
  } )
  TRANSMIT BACnet-SimpleACK-PDU
}

```

### 7.3.2.X65.14 Source Audit Reporting - Confirmed Service Operation Audit Notification

Purpose: Verify that source audit notifications for confirmed services contain InvokeId information.

Test Concept: An auditable confirmed service is performed by the IUT, and it is verified that the resulting source audit notification contains an InvokeId, and other notification fields represent the auditable operation performed.

Configuration Requirements: The IUT is configured to report source audit notifications for a confirmed service. If the IUT does not support source audit reporting for any confirmed services, this test shall be skipped. The IUT is configured with AR as the source Audit Reporter object.

Test Steps:

1. MAKE(the IUT perform an auditable operation, O, on the TD which uses an confirmed service)

```

2. IF the IUT is configured to send unconfirmed audit notifications THEN {
    BEFORE AR.Maximum_Send_Delay + Notification Fail Time
    RECEIVE UnconfirmedAuditNotification-Request,
        'Notifications' = ( {
            source-timestamp = (IUT's local time),
            -- target-timestamp absent
            source-device = IUT,
            source-object = (the object which initiated the op or absent if not initiated by an
                           object),
            operation = O,
            source-comment = (any valid value or absent),
            -- target-comment absent
            invoke-id = (the Invoke Id from the operation),
            source-user-id = (the value from the operation if provided, otherwise absent),
            source-user-role = (the value from the operation if provided, otherwise absent),
            target-device = TD,
            target-object = (the target object or absent if the target is not an object),
            target-property = (the target property or absent if the target is not a property),
            target-priority = (the priority supplied, or absent if the target is not a property.
                           shall be 16 or absent if no priority supplied and the target is a
                           property),
            target-value = (the target value or absent if no target value for the operation.
                           may be absent if the value size is larger than 32 encoded
                           octets),
            current-value = (the value before the op if the op targeted a property, or absent.
                           May be absent even if targeting a property),
            result = (the reason for failure if the op failed, otherwise absent)
        } )
    } ELSE {
        BEFORE AR.Maximum_Send_Delay + Notification Fail Time
        RECEIVE ConfirmedAuditNotification-Request,
            'Notifications' = ( {
                source-timestamp = (IUT's local time),
                -- target-timestamp absent
                source-device = IUT,
                source-object = (the object which initiated the op or absent if not initiated by an
                               object),
                operation = O,
                source-comment = (any valid value or absent),
                -- target-comment absent
                invoke-id = (the Invoke Id from the operation),
                source-user-id = (the value from the operation if provided, otherwise absent),
                source-user-role = (the value from the operation if provided, otherwise absent),
                target-device = TD,
                target-object = (the target object or absent if the target is not an object),
                target-property = (the target property or absent if the target is not a property),
                target-priority = (the priority supplied, or absent if the target is not a property.
                               shall be 16 or absent if no priority supplied and the target is a
                               property),
                target-value = (the target value or absent if no target value for the operation.
                               may be absent if the value size is larger than 32 encoded
                               octets),
                current-value = (the value before the op if the op targeted a property, or absent.
                               May be absent even if targeting a property),
                result = (the reason for failure if the op failed, otherwise absent)
            } )
        TRANSMIT BACnet-SimpleACK-PDU
    }
}

```

### 7.3.2.X65.15 Source Audit Reporting - Operations with Priority Test

Purpose: Verify that source audit notifications which for writes which convey a priority include the priority in the notification.

Test Concept: An auditable write, which includes a priority, is performed on a commandable object by the IUT and it is verified that the resulting source audit notification contains a priority, and other notification fields represent the auditable operation performed.

Configuration Requirements: The IUT is configured to report audit notifications for writes on a commandable object, O1. The IUT is configured with AR as the source Audit Reporter object. If the IUT does not provide priorities in auditable operations it performed, this test shall be skipped.

Test Steps:

1. MAKE(the IUT perform an auditable operation in containing a priority, other than 16, on the TD)

2. IF the IUT is configured to send unconfirmed audit notifications THEN {

BEFORE AR.Maximum\_Send\_Delay + Notification Fail Time

RECEIVE UnconfirmedAuditNotification-Request,

'Notifications' = ( {

source-timestamp = (IUT's local time),

-- target-timestamp absent

source-device = IUT,

source-object = (the object which initiated the op or absent if not initiated by an object),

operation = O,

source-comment = (any valid value or absent),

-- target-comment absent

invoke-id = (the Invoke Id from the operation),

source-user-id = (the value from the operation if provided, otherwise absent),

source-user-role = (the value from the operation if provided, otherwise absent),

target-device = TD,

target-object = (the target object),

target-property = (the target property),

target-priority = (the priority supplied),

target-value = (the target value.

may be absent if the value size is larger than 32 encoded octets),

current-value = (any valid value, or absent),

result = (the reason for failure if the op failed, otherwise absent)

} )

} ELSE {

BEFORE AR.Maximum\_Send\_Delay + Notification Fail Time

RECEIVE ConfirmedAuditNotification-Request,

'Notifications' = ( {

source-timestamp = (IUT's local time),

-- target-timestamp absent

source-device = IUT,

source-object = (the object which initiated the op or absent if not initiated by an object),

operation = O,

source-comment = (any valid value or absent),

-- target-comment absent

invoke-id = (the Invoke Id from the operation),

source-user-id = (the value from the operation if provided, otherwise absent),

source-user-role = (the value from the operation if provided, otherwise absent),

target-device = TD,

target-object = (the target object),

target-property = (the target property),

target-priority = (the priority supplied),

target-value = (the target value.

may be absent if the value size is larger than 32 encoded

```

        octets),
        current-value = (any valid value, or absent),
        result = (the reason for failure if the op failed, otherwise absent)
    } )
    TRANSMIT BACnet-SimpleACK-PDU
}

```

### 7.3.2.X65.16 Source Audit Reporting - Error Audit Notification Test

Purpose: Verify that operations performed by the IUT which fail are properly reported in audit notifications.

Test Concept: The IUT is made to perform an auditable operation on the TD and the TD returns an Error-PDU. It is verified that a source audit notification is sent which contains the error that occurred. This is repeated twice more with the TD returning a Reject PDU, and then an Abort PDU.

Configuration Requirements: The IUT is configured to report all audit notifications. The IUT is configured with AR as the source Audit Reporter object.

Test Steps:

-- Error-PDU

1. MAKE(the IUT perform an auditable operation on TD which will fail (via return of a BACnetErrorPDU, or a Result(+) with error information)

2. IF the IUT is configured to send unconfirmed audit notifications THEN {

BEFORE AR.Maximum\_Send\_Delay + Notification Fail Time

RECEIVE UnconfirmedAuditNotification-Request,

'Notifications' = ( {

source-timestamp = (IUT's local time),

-- target-timestamp absent

source-device = IUT,

source-object = (the object which initiated the op or absent if not initiated by an object),

operation = O,

source-comment = (any valid value or absent),

-- target-comment absent

invoke-id = (the invoke id from the operation, or absent if it was unconfirmed),

source-user-id = (the value from the operation if provided, otherwise absent),

source-user-role = (the value from the operation if provided, otherwise absent),

target-device = TD,

target-object = (the target object or absent if the target is not an object),

target-property = (the target property or absent if the target is not a property),

target-priority = (the priority supplied, or absent if the target is not a property. shall be 16 or absent if no priority supplied and the target is a property),

target-value = (the target value or absent if no target value for the operation. may be absent if the value size is larger than 32 encoded octets),

current-value = (the value before the op if the op targeted a property, or absent. May be absent even if targeting a property),

result = (the error reported for the operation)

} )

} ELSE {

BEFORE AR.Maximum\_Send\_Delay + Notification Fail Time

```

RECEIVE UnconfirmedAuditNotification-Request,
'Notifications' = ( {
    source-timestamp = (IUT's local time),
    -- target-timestamp absent
    source-device = IUT,
    source-object = (the object which initiated the op or absent if not
        initiated by an object),
    operation = O,
    source-comment = (any valid value or absent),
    -- target-comment absent
    invoke-id = (the invoke id from the operation, or absent if it was
        unconfirmed),
    source-user-id = (the value from the operation if provided, otherwise
        absent),
    source-user-role = (the value from the operation if provided, otherwise
        absent),
    target-device = TD,
    target-object = (the target object or absent if the target is not an object),
    target-property = (the target property or absent if the target is not a
        property),
    target-priority = (the priority supplied, or absent if the target is not a
        property. shall be 16 or absent if no priority supplied and the
        target is a property),
    target-value = (the target value or absent if no target value for the
        operation. may be absent if the value size is larger than 32
        encoded octets),
    current-value = (the value before the op if the op targeted a property, or
        absent. May be absent even if targeting a property),
    result = (the error reported for the operation)
} )

```

TRANSMIT BACnet-SimpleACK-PDU

}

-- Reject-PDU

3. MAKE(the IUT perform an auditable operation on TD which will fail (via return of a BACnetRejectPDU))

4. IF the IUT is configured to send unconfirmed audit notifications THEN {

BEFORE AR.Maximum\_Send\_Delay + Notification Fail Time

RECEIVE UnconfirmedAuditNotification-Request,

```

'Notifications' = ( {
    source-timestamp = (IUT's local time),
    -- target-timestamp absent
    source-device = IUT,
    source-object = (the object which initiated the op or absent if not initiated by an
        object),
    operation = O,
    source-comment = (any valid value or absent),
    -- target-comment absent
    invoke-id = (the invoke id from the operation, or absent if it was unconfirmed),
    source-user-id = (the value from the operation if provided, otherwise absent),
    source-user-role = (the value from the operation if provided, otherwise absent),
    target-device = TD,
    target-object = (the target object or absent if the target is not an object),
    target-property = (the target property or absent if the target is not a property),
    target-priority = (the priority supplied, or absent if the target is not a property.
        shall be 16 or absent if no priority supplied and the target is a
        property),
    target-value = (the target value or absent if no target value for the operation.
        may be absent if the value size is larger than 32 encoded
        octets),
    current-value = (the value before the op if the op targeted a property, or absent.

```



```

        May be absent even if targeting a property),
        result = (the error reported for the operation)
    } )
} ELSE {
    BEFORE AR.Maximum_Send_Delay + Notification Fail Time
    RECEIVE ConfirmedAuditNotification-Request,
    'Notifications' = ( {
        source-timestamp = (IUT's local time),
        -- target-timestamp absent
        source-device = IUT,
        source-object = (the object which initiated the op or absent if not initiated by an
            object),
        operation = O,
        source-comment = (any valid value or absent),
        -- target-comment absent
        invoke-id = (the invoke id from the operation, or absent if it was unconfirmed),
        source-user-id = (the value from the operation if provided, otherwise absent),
        source-user-role = (the value from the operation if provided, otherwise absent),
        target-device = TD,
        target-object = (the target object or absent if the target is not an object),
        target-property = (the target property or absent if the target is not a property),
        target-priority = (the priority supplied, or absent if the target is not a property.
            shall be 16 or absent if no priority supplied and the target is a
            property),
        target-value = (the target value or absent if no target value for the operation.
            may be absent if the value size is larger than 32 encoded
            octets),
        current-value = (the value before the op if the op targeted a property, or absent.
            May be absent even if targeting a property),
        result = (the error reported for the operation)
    } ) TRANSMIT BACnet-SimpleACK-PDU
}

```

-- Abort-PDU

3. MAKE(the IUT perform an auditable operation on TD which will fail (via return of a BACnetAbortPDU))

4. IF the IUT is configured to send unconfirmed audit notifications THEN {

BEFORE AR.Maximum\_Send\_Delay + Notification Fail Time

RECEIVE UnconfirmedAuditNotification-Request,

```

'Notifications' = ( {
    source-timestamp = (IUT's local time),
    -- target-timestamp absent
    source-device = IUT,
    source-object = (the object which initiated the op or absent if not initiated by an
        object),
    operation = O,
    source-comment = (any valid value or absent),
    -- target-comment absent
    invoke-id = (the invoke id from the operation, or absent if it was unconfirmed),
    source-user-id = (the value from the operation if provided, otherwise absent),
    source-user-role = (the value from the operation if provided, otherwise absent),
    target-device = TD,
    target-object = (the target object or absent if the target is not an object),
    target-property = (the target property or absent if the target is not a property),
    target-priority = (the priority supplied, or absent if the target is not a property.
        shall be 16 or absent if no priority supplied and the target is a
        property),
    target-value = (the target value or absent if no target value for the operation.
        may be absent if the value size is larger than 32 encoded
        octets),
    current-value = (the value before the op if the op targeted a property, or absent.

```

```

        May be absent even if targeting a property),
        result = (the error reported for the operation)
    } )
} ELSE {
    BEFORE AR.Maximum_Send_Delay + Notification Fail Time
    RECEIVE ConfirmedAuditNotification-Request,
    'Notifications' = ( {
        source-timestamp = (IUT's local time),
        -- target-timestamp absent
        source-device = IUT,
        source-object = (the object which initiated the op or absent if not initiated by an
            object),
        operation = O,
        source-comment = (any valid value or absent),
        -- target-comment absent
        invoke-id = (the invoke id from the operation, or absent if it was unconfirmed),
        source-user-id = (the value from the operation if provided, otherwise absent),
        source-user-role = (the value from the operation if provided, otherwise absent),
        target-device = TD,
        target-object = (the target object or absent if the target is not an object),
        target-property = (the target property or absent if the target is not a property),
        target-priority = (the priority supplied, or absent if the target is not a property.
            shall be 16 or absent if no priority supplied and the target is a
            property),
        target-value = (the target value or absent if no target value for the operation.
            may be absent if the value size is larger than 32 encoded
            octets),
        current-value = (the value before the op if the op targeted a property, or absent.
            May be absent even if targeting a property),
        result = (the error reported for the operation)
    } )
    TRANSMIT BACnet-SimpleACK-PDU
}

```

### 7.3.2.X65.17 Source Audit Reporting - Single Source Audit Reporter Object Test

Purpose: Verify that the IUT contains a single Audit Report for source audit reporting.

Test Concept: Check all Audit Reporter objects in the IUT and verify that only one is for source audit reporting.

Test Steps:

1. SourceAR = NONE
2. REPEAT AR = (each Audit Reporter object) DO {
  - IF AR.Audit\_Source\_Reporter is TRUE THEN
    - IF SourceAR is not NONE THEN
 ERROR "Multiple Audit Reporter objects setup for source reporting."
3. CHECK(SourceAR is not NONE)

### 7.3.2.X65.18 Audit Forwarding Test

Purpose: Verify that the IUT forwards received audit notifications.

Test Concept: Send a sequence of confirmed and unconfirmed, unicast, and broadcast audit notifications to the IUT and verify they are forwarded.

Configuration Requirements: The IUT is configured with an Audit Log, AL, which is setup to forward and delete audit notifications with no delay. The IUT is configured to send notifications unconfirmed.

#### Test Steps:

##### -- Unicast confirmed

1. TRANSMIT ConfirmedAuditNotification-Request,  
    'Notifications' = (N1: a list of 1 or more notifications)
2. RECEIVE BACnet-SimpleACK-PDU
3. BEFORE Notification Fail Time  
    RECEIVE UnconfirmedAuditNotification-Request,  
    'Notifications' = N1
4. TRANSMIT ReadRange-Request,  
    'Object Identifier' = AL,  
    'Property Identifier' = Log\_Buffer,  
    'Reference Index' = 1,  
    'Count' = 10
5. RECEIVE ReadRange-Ack,  
    'Object Identifier' = AL,  
    'Property Identifier' = Log\_Buffer,  
    'Result Flags' = (False, False, False),  
    'Count' = 0

##### -- Unicast unconfirmed

6. TRANSMIT UnconfirmedAuditNotification-Request,  
    'Notifications' = (N2: a list of 1 or more notifications)
7. BEFORE Notification Fail Time  
    RECEIVE UnconfirmedAuditNotification-Request,  
    'Notifications' = N2
8. TRANSMIT ReadRange-Request,  
    'Object Identifier' = AL,  
    'Property Identifier' = Log\_Buffer,  
    'Reference Index' = 1,  
    'Count' = 10
9. RECEIVE ReadRange-Ack,  
    'Object Identifier' = AL,  
    'Property Identifier' = Log\_Buffer,  
    'Result Flags' = (False, False, False),  
    'Count' = 0

##### -- Local Broadcast

10. TRANSMIT UnconfirmedAuditNotification-Request,  
    DESTINATION = LOCAL BROADCAST,  
    'Notifications' = (N3: a list of 1 or more notifications)
11. BEFORE Notification Fail Time  
    RECEIVE UnconfirmedAuditNotification-Request,  
    'Notifications' = N3
12. TRANSMIT ReadRange-Request,  
    'Object Identifier' = AL,  
    'Property Identifier' = Log\_Buffer,  
    'Reference Index' = 1,  
    'Count' = 10
13. RECEIVE ReadRange-Ack,  
    'Object Identifier' = AL,  
    'Property Identifier' = Log\_Buffer,  
    'Result Flags' = (False, False, False),  
    'Count' = 0

##### -- Global Broadcast

10. TRANSMIT UnconfirmedAuditNotification-Request,  
    DESTINATION = GLOBAL BROADCAST,

'Notifications' = (N4: a list of 1 or more notifications)

11. BEFORE Notification Fail Time

RECEIVE UnconfirmedAuditNotification-Request,

'Notifications' = N4

12. TRANSMIT ReadRange-Request,

'Object Identifier' = AL,

'Property Identifier' = Log\_Buffer,

'Reference Index' = 1,

'Count' = 10

13. RECEIVE ReadRange-Ack,

'Object Identifier' = AL,

'Property Identifier' = Log\_Buffer,

'Result Flags' = (False, False, False),

'Count' = 0

[Add new **Clause 7.3.2.X66**, p. 185]

[Reviewer Note: Clause 7.3.2 (Object Specific Tests) contains no tests for the Staging object type.]

### 7.3.2.X66 Staging Object Tests

#### 7.3.2.X66.1 Clamping Present\_Value to Max\_Pres\_Value or Min\_Pres\_Value

Purpose: To verify that Present\_Value will be modified internally to stay within the boundaries of Min\_Pres\_Value or Max\_Pres\_Value.

Test Concept: Present\_Value is written with a value greater than Max\_Pres\_Value. If the value is accepted, Present\_Value is read to verify that it clamped to Max\_Pres\_Value. If Stages is writable, an attempt is made to reduce the limit defined in the last stage. If successful, Present\_Value is checked to verify it changed to match the new limit. Present\_Value is then written with a value less than Min\_Pres\_Value. If the value is accepted, Present\_Value is read to verify that it clamped to Min\_Pres\_Value. If Min\_Pres\_Value is writable, the value is increased and Present\_Value is read to verify that it matches the new Min\_Pres\_Value.

Configuration Requirements: None

Test Steps:

1. READ MAXPV1 = Max\_Pres\_Value
2. READ PV1 = Present\_Value
3. TRANSMIT WriteProperty-Request
  - 'Object-Identifier' = (the Staging object under test),
  - 'Property Identifier' = Present\_Value,
  - 'Property Value' = (a value greater than MAXPV1)
4. RECEIVE BACnet-SimpleACK-PDU |
  - (BACnet-Error-PDU,
  - 'Error Class' = Property,
  - 'Error Code' = VALUE\_OUT\_OF\_RANGE)
5. IF (a BACnet-SimpleACK-PDU was received) THEN
  - VERIFY Present\_Value = MAXPV1
 ELSE
  - VERIFY Present\_Value = PV1
  - WRITE Present\_Value = MAXPV1
6. IF (Stages is writable) THEN
  - READ NS = Stages[0]
  - READ STGN = Stages, ARRAY INDEX = NS
  - TRANSMIT WriteProperty-Request
    - 'Object-Identifier' = (the Staging object under test),
    - 'Property Identifier' = Stages,

```

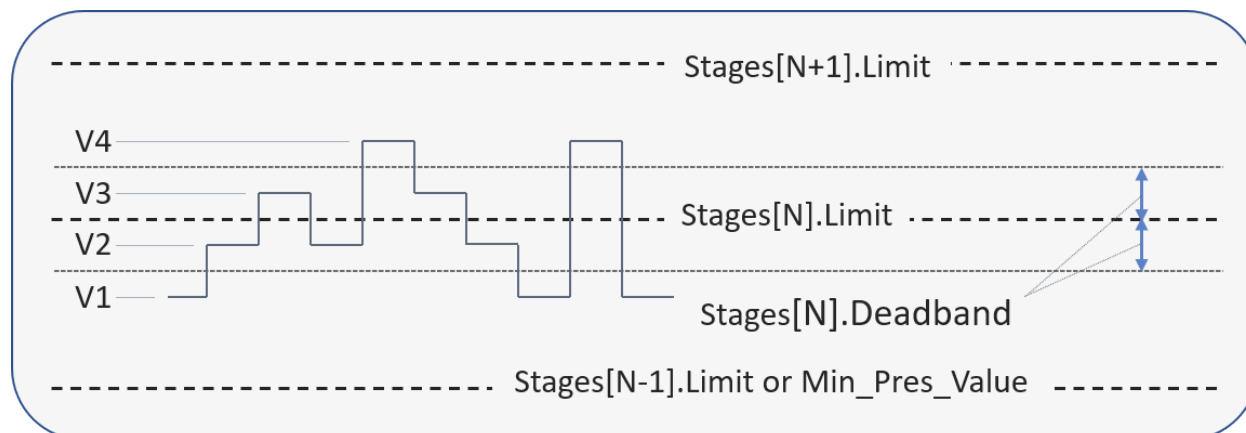
'Property Array Index' = NS,
'Property Value' = {
    Limit = (STAGEPV1: any value less than STGN.Limit)
    Values = STGN.Values,
    DeadBand = STGN.Deadband
}
RECEIVE BACnet-SimpleACK-PDU |
(BACnet-Error-PDU,
  'Error Class' = PROPERTY,
  'Error Code' = VALUE_OUT_OF_RANGE)
IF (a BACnet-SimpleACK-PDU was received) THEN
  VERIFY Present_Value = STAGEPV1
7. READ MINPV1 = Min_Pres_Value
8. READ PV2 = Present_Value
9. TRANSMIT WriteProperty-Request
  'Object-Identifier' = (the Staging object under test),
  'Property Identifier' = Present_Value,
  'Property Value' = (a value less than MINPV1)
10. RECEIVE BACnet-SimpleACK-PDU |
  (BACnet-Error-PDU,
    'Error Class' = PROPERTY,
    'Error Code' = VALUE_OUT_OF_RANGE)
11. IF (a BACnet-SimpleACK-PDU was received) THEN
  VERIFY Present_Value = MINPV1
  ELSE
  VERIFY Present_Value = PV2
  WRITE Present_Value = MINPV1
12. IF (Min_Pres_Value is writable) THEN
  READ STG1 = Stages, ARRAY INDEX = 1
  TRANSMIT WriteProperty-Request
    'Object-Identifier' = (the Staging object under test),
    'Property Identifier' = Min_Pres_Value,
    'Property Value' = (MINPV2: MINPV1 < MINPV2 < (STG1.Limit - STG1.Deadband))
  WAIT Internal Processing Fail Time
  VERIFY Present_Value = MINPV2

```

### 7.3.2.X66.2 Present\_Stage Evaluation

Purpose: To verify that Present\_Stage evaluates correctly based on the Present\_Value, stage limits, and deadband values.

Test Concept: Present\_Value is written with different values that exercise the Present\_Stage evaluation algorithm. After each write to Present\_Value, Present\_Stage is read to verify that the algorithm evaluates correctly.



Configuration Requirements: The Staging object used for this test must be configured with at least two stages. If supported, Deadband shall be configured with a non-zero value for stage N ( $\text{Stage}[N].\text{deadband} > 0$ ). At the start of the test, the Staging object is configured with  $\text{Present\_Value} = V1$ .

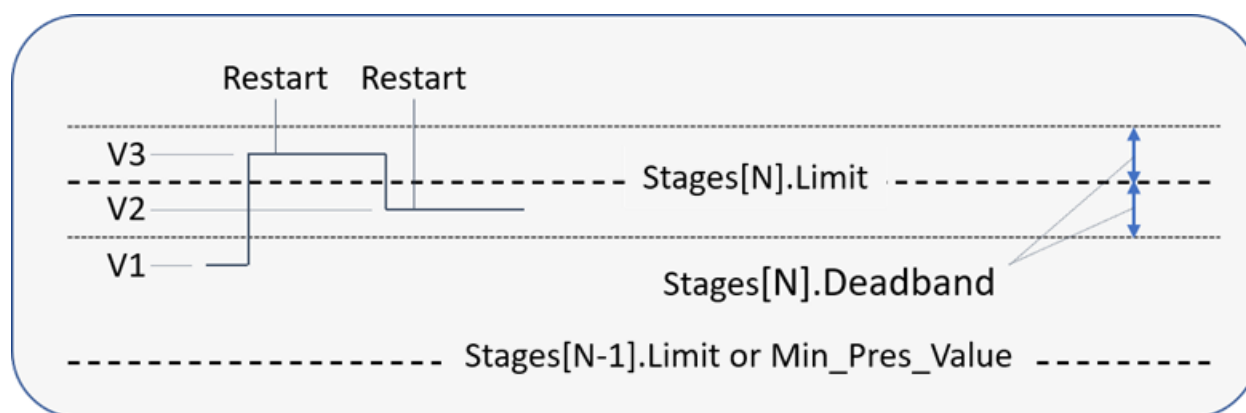
Test Steps:

1. READ  $N = \text{Present\_Stage}$
2. VERIFY  $\text{Present\_Value} = V1$
3. If ( $\text{Stages}[N].\text{Deadband} > 0$ ) THEN {
  - WRITE  $\text{Present\_Value} = (V2: \text{Stages}[N].\text{Limit} - \text{Stages}[N].\text{Deadband} < V2 < \text{Stages}[N].\text{Limit})$
  - VERIFY  $\text{Present\_Stage} = N$
  - WRITE  $\text{Present\_Value} = (V3: \text{Stages}[N].\text{Limit} < V3 < \text{Stages}[N].\text{Limit} + \text{Stages}[N].\text{Deadband})$
  - VERIFY  $\text{Present\_Stage} = N$
  - WRITE  $\text{Present\_Value} = V2$
  - VERIFY  $\text{Present\_Stage} = N$
  - WRITE  $\text{Present\_Value} = (V4: \text{Stages}[N].\text{Limit} + \text{Stages}[N].\text{Deadband} < V4 < \text{Stages}[N+1].\text{Limit})$
  - VERIFY  $\text{Present\_Stage} = N+1$
  - WRITE  $\text{Present\_Value} = V3$
  - VERIFY  $\text{Present\_Stage} = N+1$
  - WRITE  $\text{Present\_Value} = V2$
  - VERIFY  $\text{Present\_Stage} = N+1$
  - WRITE  $\text{Present\_Value} = V1$
  - VERIFY  $\text{Present\_Stage} = N$
4. WRITE  $\text{Present\_Value} = V4$
5. VERIFY  $\text{Present\_Stage} = N+1$
6. WRITE  $\text{Present\_Value} = V1$
7. VERIFY  $\text{Present\_Stage} = N$

### 7.3.2.X66.3 Present\_Stage Evaluates on Restart

Purpose: To verify that  $\text{Present\_Stage}$  is re-evaluated on device restart.

Test Concept:  $\text{Present\_Value}$  is written with a value,  $V3$ , that exceeds  $\text{Stages}[N].\text{limit}$  but does not exceed the deadband threshold and cause a change to  $\text{Present\_Stage}$ . The IUT is restarted and  $\text{Present\_Stage}$  is read to verify that it is now  $(N+1)$ .  $\text{Present\_Value}$  is then written with a value,  $V2$ , that is below  $\text{Stages}[N].\text{limit}$  but above the deadband threshold so  $\text{Present\_Stage}$  remains at  $(N+1)$ . The IUT is restarted and  $\text{Present\_Stage}$  is read to verify that it is now  $N$ .



Configuration Requirements: The Staging object used for this test must be configured with at least two stages. Deadband shall be configured with a non-zero value for stage N ( $\text{Stage}[N].\text{Deadband} > 0$ ). If deadband for stage N cannot be configured this way in a Staging object which does not support  $\text{Default\_Present\_Value}$ , this test shall be skipped. At the start of the test, the Staging object is configured with  $\text{Present\_Value} = V1$  and  $\text{Present\_Stage} = N$ . If the IUT supports remote  $\text{Target\_References}$ , then at least 1 shall be set to an object outside the IUT.

Test Steps:

1. VERIFY  $\text{Present\_Stage} = N$
2. VERIFY  $\text{Present\_Value} = V1$

3. WRITE Present\_Value = (V3: Stages[N].Limit < V3 < (Stages[N].Limit + Stages[N].Deadband))
4. VERIFY Present\_Stage = N
5. IF (ReinitializeDevice - WARMSTART execution is supported) THEN {
  - TRANSMIT ReinitializeDevice-Request,
  - 'Reinitialized State of Device' = WARMSTART
  - 'Password' = (any valid password)
  - RECEIVE BACnet-SimpleACK-PDU
6. } ELSE {
  - MAKE (power cycle the IUT to make it reinitialize)
7. }
8. WAIT for the IUT to complete its restart
9. CHECK(that the IUT wrote to all Target References which are outside the device)
10. VERIFY Present\_Value = V3
11. VERIFY Present\_Stage = N+1
12. WRITE Present\_Value = (V2: (Stages[N].Limit - Stages[N].Deadband) < V2 < Stages[N].Limit)
13. VERIFY Present\_Stage = N+1
14. IF (ReinitializeDevice - WARMSTART execution is supported) THEN {
  - TRANSMIT ReinitializeDevice-Request,
  - 'Reinitialized State of Device' = WARMSTART
  - 'Password' = (any valid password)
  - RECEIVE BACnet-SimpleACK-PDU
15. } ELSE {
  - MAKE (power cycle the IUT to make it reinitialize)
16. }
17. WAIT for the IUT to complete its restart
18. CHECK(that the IUT wrote to all Target References which are outside the device)
19. VERIFY Present\_Value = V2
20. VERIFY Present\_Stage = N+1

### 7.3.2.X66.4 Default\_Present\_Value is Abided on Restart

Purpose: To verify that Default\_Present\_Value defines the Staging object's value on device restart.

Test Concept: A staging object which contains Default\_Present\_Value. The stage associated with Default\_Present\_Value is S1. The staging object starts with the value V2, which evaluates to a different stage, S2. The IUT is restarted, and it is verified that the staging object takes on Default\_Present\_Value, changes to the stage S1 and performs the associated writes. The IUT is restarted again, and it is verified that the staging object maintains its value, remains in stage S1 and performs the associated writes for the stage S1.

Configuration Requirements: If the IUT supports remote Target\_References, then at least 1 shall be set to an object in the TD.

Test Steps:

1. VERIFY Default\_Present\_Value = V1
2. VERIFY Present\_Value = V2
3. VERIFY Present\_Stage = S2
4. IF (ReinitializeDevice - WARMSTART execution is supported) THEN {
  - TRANSMIT ReinitializeDevice-Request,
  - 'Reinitialized State of Device' = WARMSTART
  - 'Password' = (any valid password)
  - RECEIVE BACnet-SimpleACK-PDU
5. } ELSE {
  - MAKE (power cycle the IUT to make it reinitialize)
6. }
7. WAIT for the IUT to complete its restart
8. CHECK(that the IUT wrote to all Target References which are outside the device)
9. VERIFY Present\_Value = V1
10. VERIFY Present\_Stage = S1
11. IF (ReinitializeDevice - WARMSTART execution is supported) THEN {
  - TRANSMIT ReinitializeDevice-Request,

```

'Reinitialized State of Device' = WARMSTART
'Password' = (any valid password)
RECEIVE BACnet-SimpleACK-PDU
} ELSE {
    MAKE (power cycle the IUT to make it reinitialize)
}
10. WAIT for the IUT to complete its restart
11. CHECK(that the IUT wrote to all Target References which are outside the device)
12. VERIFY Present_Value = V1
13. VERIFY Present_Stage = S1

```

### 7.3.2.X66.5 Writing to Target References

Purpose: To verify that a change of Present\_Stage results in the target references being written as per the stage definition.

Test Concept: A Stage object, O1, is selected for testing. O1's Present\_Value is written with a value that results in a change of Present\_Stage. Each Present\_Value of the Target\_References is monitored to verify that its value is set in accordance with Stage[Present\_Stage].Values. O1's Present\_Value is written again with a value that returns Present\_Stage to its initial value. Again, the Target\_References are monitored to verify that they have been written with the appropriate values.

Configuration Requirements: Target\_References is configured with references to existing binary objects with writable Present\_Value properties. The Stages property is configured with at least two stages, X and Y, such that Stages[X].Values <> Stages[Y].Values. Present\_Stage shall be X at the start of the test. Throughout the test, O1 is expected to be properly configured such that Reliability is NO\_FAULT\_DETECTED.

Test Steps:

1. VERIFY Present\_Stage = X
2. WRITE Present\_Value = (any value that causes Present\_Stage to change to Y)
3. VERIFY Present\_Stage = Y
4. REPEAT J = (1 ... Target\_References[0]) = DO {
  - READ O = Target\_References, ARRAY INDEX = J
  - VERIFY O, Present\_Value = Stages[Y].Values[J]
5. WRITE Present\_Value = (any value that causes Present\_Stage to change to X)
6. VERIFY Present\_Stage = X
7. REPEAT J = (1.. Target\_References[0]) = DO {
  - READ O = Target\_References, ARRAY INDEX = J
  - VERIFY O, Present\_Value = Stages[X].Values[J]

### 7.3.2.X66.6 Stage Value Bitstring is Same Length as Target\_References

Purpose: To verify that the bitstring length for the Values component of each stage is equal and corresponds to the number of entries in the Target\_References property.

Test Concept: For each staging object in the IUT, the Stages and Target\_References properties are read. For each object, the length of the 'Values' bitstring from the first stage is extracted. This length is compared to the length of the 'Values' bitstring in every other stage and the size of the Target\_References property to verify equality.

Configuration Requirements: None

Test Steps:

1. REPEAT O = (each Staging object in the IUT) DO {
  - READ NS = O, Stages, ARRAY INDEX = 0
  - READ STG1 = Stages, ARRAY INDEX = 1
  - NUMBITS = (number bits in STG1.Values)
  - REPEAT N = (2 through NS) DO {
    - check that the length of Stages[1].Values equals length of Stages[N].Values.
    - READ STGN = Stages, ARRAY INDEX = N
    - IF number of bits in STGN.Values <> NUMBITS THEN
      - ERROR "Length of the Values bitstrings are not the same in all stages."



```

    }
    VERIFY Target_References = NUMBITS, ARRAY_INDEX = 0
  }

```

### 7.3.2.X66.7 Max\_Pres\_Value Equals Last Stage Limit

Purpose: To verify that Max\_Pres\_Value is equivalent to the Limit defined in the last Stage.

Test Concept: Max\_Pres\_Value is read and checked for equality with the Limit defined in the last element of the Stages array.

Configuration Requirements: None

Test Steps:

1. READ N = Stages, ARRAY INDEX = 0
2. VERIFY Max\_Pres\_Value = Stages[N].Limit

### 7.3.2.X66.8 CONFIGURATION\_ERROR when Min\_Pres\_Value is too Large

Purpose: To verify that Reliability has a value of CONFIGURATION\_ERROR when Min\_Pres\_Value has a value greater than Stages[1].Limit - Stages[1].Deadband.

Test Concept: Min\_Pres\_Value is made to exceed the value of Stages[1].Limit - Stages[1].Deadband by first writing directly to Min\_Pres\_Value, then by making a change to Stages[1].Limit, and then by making a change to Stages[1].Deadband. After each modification, if it is successful, Reliability is verified to have a value of CONFIGURATION\_ERROR and then the modification is reversed, and Reliability is verified to have a value of NO\_FAULT\_DETECTED.

Configuration Requirements: At the start of the test, the Staging object used for this test, O1, shall be properly configured such that Reliability = NO\_FAULT\_DETECTED. At the start of the test, Present\_Value shall be equal to Min\_Pres\_Value.

Test Steps:

1. READ MINPV1 = Min\_Pres\_Value
2. VERIFY Present\_Value = MINPV1
3. VERIFY Reliability = NO\_FAULT\_DETECTED
4. READ STG1 = Stages, ARRAY INDEX = 1
5. SL = STG1.Limit
6. SV = STG1.Values
7. SD = STG1.Deadband
8. IF (Min\_Pres\_Value is writable) THEN
  - TRANSMIT WriteProperty-Request,
    - 'Object Identifier' = O1
    - 'Property Identifier' = Min\_Pres\_Value,
    - 'Property Value' = (MINPV2: where MINPV2 > (SL-SD))
  - RECEIVE BACnet-SimpleACK-PDU |
  - (BACnet-Error-PDU,
    - 'Error Class' = Property,
    - 'Error Code' = VALUE\_OUT\_OF\_RANGE)
  - IF (a BACnet-SimpleACK-PDU was received) THEN
    - VERIFY Min\_Pres\_Value = MINPV2
    - VERIFY Reliability = CONFIGURATION\_ERROR
    - VERIFY Present\_Value = MINPV2
    - VERIFY Present\_Stage = 1
    - WRITE Min\_Pres\_Value = MINPV1
    - VERIFY Reliability = NO\_FAULT\_DETECTED
  - ELSE
    - VERIFY Present\_Value = MINPV1
    - VERIFY Reliability = NO\_FAULT\_DETECTED
9. IF (Stages is writable) THEN
  - TRANSMIT WriteProperty-Request,

```

'Object Identifier' =      (the staging object under test),
'Property Identifier' =    Stages,
'Property Array Index' =  1,
'Property Value' =        {
                            Limit = (NL: where NL-SD < Min_Pres_Value),
                            Values = SV,
                            DeadBand = SD
                            }
RECEIVE BACnet-SimpleACK-PDU |
(BACnet-Error-PDU,
  'Error Class' = PROPERTY,
  'Error Code' = VALUE_OUT_OF_RANGE)
IF (a BACnet-SimpleACK-PDU was received) THEN
  VERIFY Stages = { Limit=NL, Values=SV, Deadband=SD }, ARRAY INDEX = 1
  VERIFY Reliability = CONFIGURATION_ERROR
  VERIFY Present_Value = MINPV1
  VERIFY Present_Stage = 1
  WRITE Stages = { Limit=SL, Values=SV, Deadband=SD }, ARRAY INDEX = 1
  VERIFY Reliability = NO_FAULT_DETECTED
ELSE
  VERIFY Stages = { Limit=SL, Values=SV, Deadband=SD }, ARRAY INDEX = 1
  VERIFY Reliability = NO_FAULT_DETECTED
TRANSMIT WriteProperty-Request,
'Object Identifier' =      (the staging object under test),
'Property Identifier' =    Stages,
'Property Array Index' =  1,
'Property Value' =        {
                            Limit=SL,
                            Values=SV,
                            Deadband=(ND: where SL-ND < Min_Pres_Value)
                            }
RECEIVE BACnet-SimpleACK-PDU |
(BACnet-Error-PDU,
  'Error Class' = PROPERTY,
  'Error Code' = VALUE_OUT_OF_RANGE)
IF (a BACnet-SimpleACK-PDU was received) THEN
  VERIFY Stages = { Limit=SL, Values=SV, Deadband=ND }, ARRAY INDEX = 1
  VERIFY Reliability = CONFIGURATION_ERROR
  VERIFY Present_Value = MINPV1
  VERIFY Present_Stage = 1
  WRITE Stages = { Limit=SL, Values=SV, Deadband=SD }, ARRAY INDEX = 1
  VERIFY Reliability = NO_FAULT_DETECTED
ELSE
  VERIFY Stages = { Limit=SL, Values=SV, Deadband=SD }, ARRAY INDEX = 1
  VERIFY Reliability = NO_FAULT_DETECTED

```

### 7.3.2.X66.9 COMMUNICATION\_FAILURE on Failed Write to External Target Reference

Purpose: To verify that Reliability is set to COMMUNICATION\_FAILURE when an attempt to write to a remote target fails.

Test Concept: The Staging object is configured with a Target\_Reference aimed at an object in the TD. The Staging object's Present\_Value is written such that a change to Present\_Stage occurs. When the external target property is written by the IUT, the TD shall not respond. The test verifies that the write to Present\_Value returns a Result(+) and Reliability is set to COMMUNICATION\_FAILURE.

Configuration Requirements: The Staging object used for this test shall be configured with at least one object, O1, in the Target\_References property which is located in the TD. The Stages property shall be configured with two stages such that Stages[S].Values = {...V1...} and Stages[S+1].Values = {...V2...} where V1 and V2 correspond to the target, O1, and V1 <> V2. At the start of the test, the Staging object is properly configured such that Reliability = NO\_FAULT\_DETECTED and

Present\_Stage = S. If no Staging object in the IUT supports external references in the Target\_References property, this test shall be skipped.

Test Steps:

1. READ S = Present\_Stage
2. WRITE Present\_Value = (X: a value that will change Present\_Stage to S+1)
3. RECEIVE WriteProperty-Request,  
     'Object Identifier' = O1,  
     'Property Identifier' = Present\_Value,  
     'Property Value' = V2
4. WAIT (Number\_Of\_APDU\_Retries + 1) \* (**Internal Processing Fail Time** + APDU\_Timeout)
5. VERIFY Reliability = COMMUNICATION\_FAILURE

### 7.3.2.X66.10 Fault Indicated on Failed Write to Local Target Reference

Purpose: To verify that Reliability is set when an attempt to write to a local target fails.

Test Concept: The Staging object is configured with a Target\_Reference aimed at an object in the IUT which is not writable or non-existent. The Staging object's Present\_Value is written such that a change to Present\_Stage occurs. The test verifies that the write to the Staging object's Present\_Value returns a Result(+) and Reliability is set to indicate a failure to write one of the targets.

Configuration Requirements: The Staging object used for this test shall be configured with at least one object, O1, in the Target\_References property, which is local to the IUT, yet not writable or non-existent. The Stages property shall be configured with two stages such that Stages[S].Values = {...V1...} and Stages[S+1].Values = {...V2...} where V1 and V2 correspond to the target, O1, and V1 <> V2. At the start of the test, the Staging object is properly configured such that Reliability = NO\_FAULT\_DETECTED and Present\_Stage = S. If no Staging object can be configured to reference a non-writable or non-existent local object, this test shall be skipped.

Test Steps:

1. READ S = Present\_Stage
- cause the staging object to write to the non-existent or non-writable target object
2. WRITE Present\_Value = (X: a value that will change Present\_Stage to S+1)
3. VERIFY Reliability <> NO\_FAULT\_DETECTED

### 7.3.2.X66.11 Out\_Of\_Service, Status\_Flags, and Reliability for Staging Object

Purpose: To verify that Present\_Value and Reliability are writable when Out\_Of\_Service is TRUE, to verify the relationship between Out\_Of\_Service, Status\_Flags, and Reliability, and to verify that writes to Target\_References only occur when Out\_Of\_Service is FALSE.

Test Concept: The Out\_Of\_Service property is set to TRUE and the value of the Status\_Flags property is validated. Present\_Value is modified to verify that Present\_Stage evaluates but writes to Target\_References do not occur. If the IUT supports Reliability values other than NO\_FAULT\_DETECTED, writability for that property is tested and the value of the Status\_Flags property is validated. The Out\_Of\_Service property is set to FALSE and the value of the Status\_Flags property is validated. The Present\_Value for one of the Target\_References is checked to verify that it has the correct value, indicative of a write that occurred when transitioning Out\_Of\_Service from TRUE to FALSE.

Configuration Requirements: The Staging object used for this test shall be configured with at least one object in the Target\_References property. The Stages property shall be configured with two stages such that Stages[S].Values = {V1...} and Stages[S+1].Values = {V2...} where V1 <> V2. At the start of the test, the Staging object is properly configured such that Reliability = NO\_FAULT\_DETECTED and Present\_Stage = S.

Test Steps:

1. READ SF1 = Status\_Flags
2. VERIFY Reliability = NO\_FAULT\_DETECTED
3. VERIFY Present\_Stage = S
4. READ O1 = Target\_References, ARRAY INDEX = 1
5. VERIFY O1, Present\_Value = V1

```

6. IF (Out_Of_Service is writable) THEN
    WRITE Out_Of_Service = TRUE
ELSE
    MAKE (Out_Of_Service TRUE)
7. VERIFY Out_Of_Service = TRUE
8. VERIFY Status_Flags = (?, ?, ?, TRUE)
9. WRITE Present_Value = (PV: (Stages[S].Limit + Stages[S].Deadband) < PV < Stages[S+1].Limit)
10. VERIFY Present_Value = PV
11. VERIFY Present_Stage = S+1
12. VERIFY O1, Present_Value = V2
13. IF (the IUT supports Reliability values other than NO_FAULT_DETECTED) THEN
    REPEAT X = (all values of the Reliability enumeration appropriate to the object type except
        NO_FAULT_DETECTED) DO {
        WRITE Reliability = X
        VERIFY Reliability = X
        VERIFY Status_Flags = (?, TRUE, ?, TRUE)
        WRITE Reliability = NO_FAULT_DETECTED
        VERIFY Reliability = NO_FAULT_DETECTED
        VERIFY Status_Flags = (?, FALSE, ?, TRUE)
    }
14. IF (Out_Of_Service is writable) THEN
    WRITE Out_Of_Service = FALSE
ELSE
    MAKE (Out_Of_Service FALSE)
15. VERIFY Status_Flags = SF1
16. VERIFY Reliability = NO_FAULT_DETECTED
17. IF (Present_Stage = S+1) THEN
    VERIFY O1, Present_Value = V2

```

### 7.3.2.X66.12 Stages Array Sizing Test

Purpose: This test case verifies that, when the size of the of the Stages array is changed by writing to the ARRAY INDEX, the size of the array changes accordingly and any new entries are properly initialized.

Test Concept: The Stages array is increased by writing the array size. It is verified that the Stages property is extended and that the new entries contain 'Limit' = 0.0, 'Values' = {0...0}, and 'Deadband' = 0.0. Reliability is verified to be CONFIGURATION\_ERROR. Present\_Stage is verified to be 1. Present\_Value is verified to be Min\_Pres\_Value. If the Stage\_Names property is present, the size of the array is checked to verify that it matches the size of the Stages array.

Throughout the test, the array size of the Stage\_Names property is checked to verify it is consistent with the array size of the Stages property.

Configuration Requirements: At the start of the test, the Staging object is properly configured such that Reliability = NO\_FAULT\_DETECTED and Present\_Stage = S. The size of the Stages array is greater than 1 and less than the maximum array size.

Test Steps:

```

1. READ N = Stages, ARRAY INDEX = 0
2. IF (Stage_Names is present) THEN {
    VERIFY = Stage_Names = N, ARRAY INDEX = 0
}
3. READ NT = Target_References, ARRAY INDEX = 0
4. WRITE Stages = N+X, ARRAY INDEX = 0 -- where (X ≥ 1)
5. VERIFY Stages = N+X, ARRAY INDEX = 0
6. VERIFY Stages = (0.0,{0...0},0.0), ARRAY INDEX = N+X -- where the number of bits in Values is NT
7. VERIFY Reliability = CONFIGURATION_ERROR
8. VERIFY Present_Stage = 1
9. READ MV = Min_Pres_Value
10. VERIFY Present_Value = MV
11. IF (Stage_Names is present) THEN {

```

```

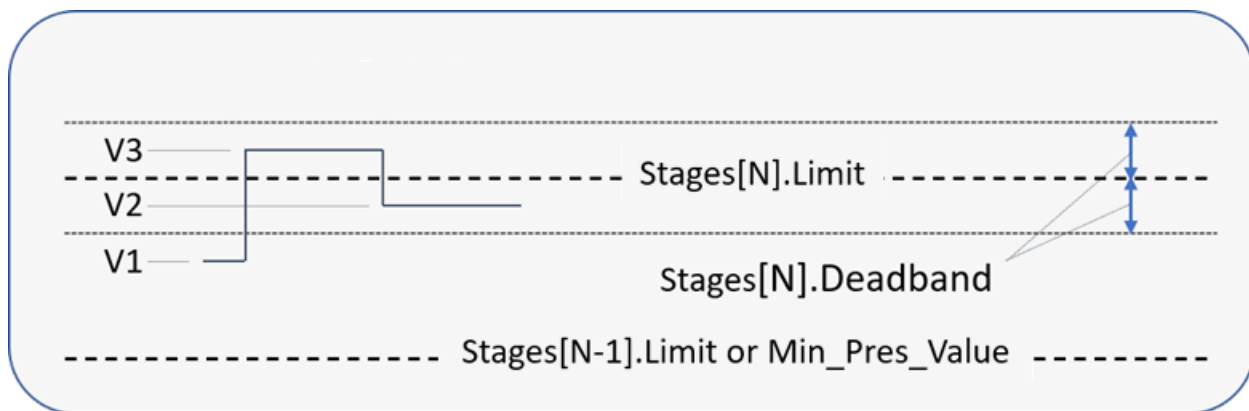
VERIFY Stage_Names = N+X, ARRAY INDEX = 0
WRITE Stages = N, ARRAY INDEX = 0
VERIFY Stage_Names = N, ARRAY INDEX = 0
}

```

### 7.3.2.X66.13 Present\_Stage Evaluates on Change to Stages Property

Purpose: To verify that Present\_Stage gets re-evaluated when the Stages property is changed.

Test Concept: Present\_Value is written with a value, V3, that exceeds Stages[N].limit but does not exceed the deadband threshold and cause a change to Present\_Stage. The Stages property is written with a new value such that Stage[N] is unaffected by the change. Present\_Stage is read to verify that it is now (N+1). Present\_Value is then written with a value, V2, that is below Stages[N].limit but above the deadband threshold so Present\_Stage remains at (N+1). The Stages property is written with a new value such that Stage[N] is unaffected by the change. Present\_Stage is read to verify that it is now N.



Configuration Requirements: The Staging object used for this test must be configured with at least two stages. Deadband shall be configured with a non-zero value for stage N ( $\text{Stage}[N].\text{Deadband} > 0$ ). If deadband for stage N cannot be configured this way, this test shall be skipped. At the start of the test, the Staging object is configured with  $\text{Present\_Value} = V1$  and  $\text{Present\_Stage} = N$ .

Test Steps:

1. VERIFY  $\text{Present\_Stage} = N$
2. VERIFY  $\text{Present\_Value} = V1$
3. READ  $\text{STAGES1} = \text{Stages}$
4. WRITE  $\text{Present\_Value} = (V3: \text{Stages}[N].\text{Limit} < V3 < (\text{Stages}[N].\text{Limit} + \text{Stages}[N].\text{Deadband}))$
5. VERIFY  $\text{Present\_Stage} = N$
6. IF (Stages is writable) THEN
  - WRITE  $\text{Stages} = (\text{STAGES2}: \text{any valid value different from STAGES1 but with the same value for Stage}[N])$
- ELSE
  - MAKE  $\text{Stages} = (\text{STAGES2}: \text{any valid value different from STAGES1 but with the same value for Stage}[N])$
7. VERIFY  $\text{Present\_Value} = V3$
8. VERIFY  $\text{Present\_Stage} = N+1$
9. WRITE  $\text{Present\_Value} = (V2: (\text{Stages}[N].\text{Limit} - \text{Stages}[N].\text{Deadband}) < V2 < \text{Stages}[N].\text{Limit})$
10. VERIFY  $\text{Present\_Stage} = N+1$
11. IF (Stages is writable) THEN
  - WRITE  $\text{Stages} = (\text{STAGES3}: \text{any valid value different from STAGES2 but with the same value for Stage}[N])$
- ELSE
  - MAKE  $\text{Stages} = (\text{STAGES3}: \text{any valid value different from STAGES2 but with the same value for Stage}[N])$
12. VERIFY  $\text{Present\_Value} = V2$
13. VERIFY  $\text{Present\_Stage} = N$

### 7.3.2.X66.14 CONFIGURATION\_ERROR when Limits are Out of Order

Purpose: To verify that Stages defined in the staging object are arranged in ascending order and, if not, Reliability is set to CONFIGURATION\_ERROR.

Test Concept: Write Stages out of order; use specific values that violate the limit value ascension rule. Verify that the object identifies the problem and sets Reliability.

Configuration Requirements: At the start of the test, the Staging object is properly configured such that Reliability = NO\_FAULT\_DETECTED.

Test Steps:

1. VERIFY Reliability = NO\_FAULT\_DETECTED
2. READ NS = Stages, ARRAY INDEX = 0
3. N = (any value where  $1 \leq N < NS$ )
4. WRITE Stages = {  
     Limit = (LIM: where  $LIM > Stages[N+1].Limit$ ),  
     Values = (any valid value of the correct length),  
     Deadband = (any valid value)  
   }, ARRAY INDEX = N
5. VERIFY Reliability = CONFIGURATION\_ERROR
6. VERIFY Present\_Value = Min\_Pres\_Value
7. VERIFY Present\_Stage = 1

### 7.3.2.X66.15 CONFIGURATION\_ERROR when Deadband < 0

Purpose: To verify that Stages defined in the staging object do not have a Deadband value less than 0, or if they do, when Deadband is less than 0, Reliability is set to CONFIGURATION\_ERROR.

Test Concept: Write an entry in the Stages property, changing the deadband to a negative value. Verify that either the write fails, or that Reliability is set to CONFIGURATION\_ERROR.

Configuration Requirements: At the start of the test, the Staging object is properly configured such that Reliability = NO\_FAULT\_DETECTED.

Test Steps:

1. VERIFY Reliability = NO\_FAULT\_DETECTED
2. READ NS = Stages, ARRAY INDEX = 0
2. N = (any value where  $1 \leq N < NS$ )
3. TRANSMIT WriteProperty-Request,  
     'Object Identifier' = (the staging object under test),  
     'Property Identifier' = Stages,  
     'Property Array Index' = N,  
     'Property Value' = {  
         Limit = Stages[N].Limit,  
         Values = Stages[N].Values,  
         Deadband = (any negative value)  
     }
4. RECEIVE BACnet-SimpleACK-PDU |  
     (BACnet-Error-PDU,  
         'Error Class' = PROPERTY,  
         'Error Code' = VALUE\_OUT\_OF\_RANGE)
5. IF (a BACnet-SimpleACK-PDU was received) THEN {  
     VERIFY Reliability = CONFIGURATION\_ERROR  
     VERIFY Present\_Value = Min\_Pres\_Value  
     VERIFY Present\_Stage = 1  
   }

### 7.3.2.X66.16 CONFIGURATION\_ERROR when Stages Size is less than Two

Purpose: To verify that the Stages array has a minimum length of two, and if not, Reliability is set to CONFIGURATION\_ERROR.

Test Concept: Write the Stages property, without an array index, setting the length of the array to 1. Verify that either the write fails, or that Reliability is set to CONFIGURATION\_ERROR.

Configuration Requirements: At the start of the test, the Staging object is properly configured such that Reliability = NO\_FAULT\_DETECTED.

Test Steps:

1. VERIFY Reliability = NO\_FAULT\_DETECTED
2. TRANSMIT WriteProperty-Request,  
     'Object Identifier' = (the staging object under test),  
     'Property Identifier' = Stages,  
     'Property Array Index' = 0,  
     'Property Value' = (0 or 1)
3. RECEIVE BACnet-SimpleACK-PDU |  
     (BACnet-Error-PDU,  
         'Error Class' = PROPERTY,  
         'Error Code' = VALUE\_OUT\_OF\_RANGE)
5. IF (a BACnet-SimpleACK-PDU was received) THEN {  
     VERIFY Reliability = CONFIGURATION\_ERROR  
     VERIFY Present\_Value = Min\_Pres\_Value  
     VERIFY Present\_Stage = 1  
 }

### 7.3.2.X66.17 Stage\_Names and Stages Size Equality Test

Purpose: To verify that the size of the Stage\_Names array is equal to the size of the Stages array.

Test Concept: Verify that the Stages array and Stage\_Names array are of the same length.

Test Steps:

1. READ N = Stages, ARRAY INDEX = 0
2. VERIFY Stage\_Names = N, ARRAY INDEX = 0

### 7.3.2.X66.18 Stage\_Names Array Sizing Test

Purpose: This test case verifies that, when the size of the Stage\_Names array is changed by writing to the ARRAY INDEX, the size of the array and the size of the Stages array changes accordingly and new values added to the Stages array are initialized to contain 'Limit' = 0.0, 'Values' = {0...0}, and 'Deadband' = 0.0, and Reliability is set to CONFIGURATION\_ERROR.

Test Concept: Resize the Stage\_Names array larger by writing the size and verify that Stages is also resized. Shrink the array back and verify Stages. Resize once more by writing the whole array and verify that Stages resizes correctly. Each time verify that new stages are correct.

Configuration Requirements: If the Stages property is not resizable by writing to it, this test shall be skipped.

Test Steps:

1. READ N = Stage\_Names, ARRAY INDEX = 0
2. WRITE Stage\_Names = N+1, ARRAY INDEX = 0
3. VERIFY Stages = N+1, ARRAY INDEX = 0
4. VERIFY Stages = {Limit=0.0, Values={0...0}, Deadband=0.0}, ARRAY INDEX = N+1
5. WRITE Stage\_Names = N, ARRAY INDEX = 0
6. VERIFY Stages = N, ARRAY INDEX = 0
7. WRITE Stage\_Names = (an array, of strings, with a length, N2, which the IUT will accept other than N)
8. VERIFY Stages = N2, ARRAY INDEX = 0

```

9. VERIFY Stages = (an array of length N2 of stages consistent with the object's configuration)
10. IF (N2 > N) THEN {
    REPEAT J = (N ... N2) {
        VERIFY Stages = {Limit=0.0, Values={0...0}, Deadband=0.0}, ARRAY INDEX = J
    }
}

```

### 7.3.2.X66.19 Target\_References Array Sizing Test

Purpose: To verify that a change to size of the Target\_References array results in an equivalent change to the length of the 'Values' portion of all elements of the Stages property and that new bits in the 'Values' are set to '0'.

Test Concept: Resize the Target\_References array larger and verify that the values field in each stage is updated with new bits. Resize the array smaller and verify that the values field in each stage is resized smaller.

Configuration Requirements: The staging object is configured with at least 1 Target\_Reference.

Test Steps:

```

1. READ STAGES1 = Stages
2. NTR = (length of STAGES1[0].Values)
3. WRITE Target_References = NTR+1, ARRAY INDEX = 0
4. REPEAT J = (1 ... STAGES1[0]) DO {
    VERIFY Stages = {
        Limit=STAGES1[J].Limit,
        Values=(the value of STAGES1[J].Values with 1 more 0 tacked on the end),
        DeadBand=STAGES1 [J].Deadband
    }, ARRAY INDEX = J
}

5. WRITE Target_References = NTR, ARRAY INDEX = 0
6. REPEAT J = (1 ... STAGES1[0]) DO {
    VERIFY Stages = {
        Limit=STAGES1[J].Limit,
        Values=(the value of STAGES1[J].Values),
        DeadBand=STAGES1[J].Deadband
    }, ARRAY INDEX = J
}

```

### 7.3.2.X66.20 Writing Target\_References with an Unsupported External Reference

Purpose: To verify the correct Result(-) when Target\_References does not support objects in an external device.

Test Concept: Attempt writing Target\_References of a Staging object with an external object reference. Verify the IUT returns the correct Result(-).

Configuration Requirements: The IUT is configured with a Staging Value object which does not support references to external objects. If the IUT cannot be configured this way, this test shall be skipped.

Test Steps:

```

1. READ X = Target_References
2. TRANSMIT WriteProperty-Request,
    'Object Identifier' = (the staging object under test),
    'Property Identifier' = Target_References,
    'Property Array Index' = 1,
    'Property Value' = (a reference to a binary object in the TD)

3. RECEIVE BACnet-Error-PDU,
    'Error Class' = PROPERTY
    'Error Code' = OPTIONAL_FUNCTIONALITY_NOT_SUPPORTED

```





## 135.1-2019s-8 Add new and correct existing Application Service Initiation Tests

### Rationale

Errors have been identified in a number of application service initiation tests in ANSI/ASHRAE Standard 135.1-2019.

In addition, the test coverage is increased by the addition of new tests.

## 8. APPLICATION SERVICE INITIATION TESTS

[Change and renumber **Clause 8.1**, p. 186]

[Reviewer Note: Corrected the 'Event State Acknowledged' field to allow for OFFNORMAL for any of the off-normal states. Moved test 8.1 to new subclause 8.1.1.]

### 8.1 AcknowledgeAlarm Service Initiation Tests

#### 8.1.1 AcknowledgeAlarm Service Initiation Test

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

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

Test Steps:

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

[Add new **Clause 8.1.X2**, p. 187]

[Reviewer Note: Added a new test ensuring correct behavior acknowledging an event notification that has been forwarded by a Notification Forwarder.]

### **8.1.X2 Successful Alarm Acknowledgment of Confirmed Event Notifications Using the 'Initiating Device Identifier' Parameter**

**Purpose:** To verify that the IUT is correctly implemented to send AcknowledgeAlarm directly to the device indicated by the 'Initiating Device Identifier' parameter of the event notification, in alarms and events that are reported to the IUT via the ConfirmedEventNotification and UnconfirmedEventNotification services. Requests having the 'Initiating Device Identifier' parameter equal-to and different from SOURCE, are both tested.

**Test Concept:** Two times a purposefully constructed event notification is sent, and it is observed that IUT is acknowledging directly to the device indicated by the 'Initiating Device Identifier' parameter of the event notification.

**Configuration:** For this test, the tester shall choose an object, O1, and tester places O1 into an alarm state such that the transition requires an acknowledgment. Then purposefully the EventNotification packet which is sent is crafted to represent that a Notification Forwarder was involved, so though SOURCE is from the TD, the O1 resides in a different device TD1. Then the steps are repeated but with 'Initiating Device Identifier' representing that O1 is in TD and thus the same device. Each time it is observed that the AcknowledgeAlarm is sent to the device represented as the 'Initiating Device Identifier'.

**Test Steps:**

1. TRANSMIT ConfirmedEventNotification-Request | UnconfirmedEventNotification-Request,  
     'Subscriber Process Identifier' = (a value acceptable to the IUT configured in the Notification Class object for the IUT),  
     'Initiating Device Identifier' = TD1, // representing that O1 is in different device from SOURCE  
     'Event Object Identifier' = O1,  
     'Time Stamp' = (any valid value, T1),  
     'Notification Class' = (the value configured in O1),  
     'Priority' = (any value selected by the TD),  
     'Event Type' = (any value selected by the TD),  
     'Notify Type' = ALARM | EVENT,  
     'AckRequired' = TRUE,  
     'From State' = (any valid value),  
     'To State' = (any valid value, S1),  
     'Event Values' = (any event values appropriate to the event type)
2. IF (the ConfirmedEventNotification choice was selected) THEN  
     RECEIVE BACnet-SimpleACK-PDU
3. MAKE (the IUT acknowledge O1)
4. RECEIVE AcknowledgeAlarm-Request, DESTINATION=TD1  
     'Acknowledging Process Identifier' = (any process identifier),  
     'Event Object Identifier' = O1,  
     'Event State Acknowledged' = S1, or OFFNORMAL if S1 is an off-normal state  
     'Time Stamp' = T1,  
     'Acknowledgement Source' = (any valid value),  
     'Time of Acknowledgement' = (any valid value)
5. TRANSMIT BACnet-SimpleACK-PDU
6. TRANSMIT ConfirmedEventNotification-Request | UnconfirmedEventNotification-Request,  
     'Subscriber Process Identifier' = (a value acceptable to the IUT configured in the Notification Class object for the IUT),  
     'Initiating Device Identifier' = TD, // representing that O1 is present in same device as SOURCE  
     'Event Object Identifier' = O1,  
     'Time Stamp' = (any valid value, T1),  
     'Notification Class' = (the value configured in O1),  
     'Priority' = (any value selected by the TD),  
     'Event Type' = (any value selected by the TD),

- 'Notify Type' = ALARM | EVENT,
- 'AckRequired' = TRUE,
- 'From State' = (any valid value),
- 'To State' = (any valid value, S1),
- 'Event Values' = (any event values appropriate to the event type)
- 7. IF (the ConfirmedEventNotification choice was selected) THEN  
RECEIVE BACnet-SimpleACK-PDU
- 8. MAKE (the IUT acknowledge O1)
- 9. RECEIVE AcknowledgeAlarm-Request,  
'Acknowledging Process Identifier' = (any process identifier),  
'Event Object Identifier' = O1,  
'Event State Acknowledged' = S1, or OFFNORMAL if S1 is an off-normal state  
'Time Stamp' = T1,  
'Acknowledgement Source' = (any valid value),  
'Time of Acknowledgement' = (any valid value)
- 10. TRANSMIT BACnet-SimpleACK-PDU

## 8.2 ConfirmedCOVNotification Service Initiation Tests

[Change **Clause 8.2.1**, p. 187]

[Reviewer Note: Updated description of the 'List of Values' to improve readability. Updated 'Configuration Requirements'. Add clarification to test that the last COVNotification shall reflect the correct values. Removed unnecessary RECEIVE BACnet-SimpleACK-PDU steps. Improved test name and wording to include generic object references.]

### 8.2.1 Change of Value Notification ~~from an Analog Input, Analog Output, and Analog Value~~ Object's for Changes to Present\_Value in Objects with a COV\_Increment

Purpose: To verify that the IUT can initiate ConfirmedCOVNotification service requests conveying a change of the Present\_Value property of ~~Analog Input, Analog Output, Lighting Output, and Analog Value, Large Analog Value, Integer Value, and Positive Integer Value~~ in Numeric Objects.

Test Concept: A subscription for COV notifications is established, using a Lifetime of L. L shall be set to a value less than 24 hours and large enough to complete the test. The Present\_Value of the monitored object is changed by an amount less than the COV increment and it is verified that no COV notification is received. The Present\_Value is then changed by an amount greater than the COV increment and a notification shall be received. The Present\_Value may be changed using the WriteProperty service or by another means such as changing the input signal represented by an Analog Input object. For some implementations it may be necessary to write to the Out\_Of\_Service property first to accomplish this task. For implementations where it is not possible to write to these properties at all the vendor shall provide an alternative trigger mechanism to accomplish this task. All of these methods are equally acceptable.

Configuration Requirements: At the beginning of the test, the Out\_Of\_Service property shall have a value of FALSE. *Select an object where Present\_Value is not expected to change outside the tester's control by more than COV\_Increment or which has a writable Out\_Of\_Service. In devices where the COV\_Increment is always less than the minimal change that Present\_Value can make, skip steps 8 through 10.*

*Notes to Tester: The IUT may initiate additional COVNotifications. The final COVNotification shall accurately reflect Present\_Value and Status\_Flags.*

Test Steps:

REPEAT X = (one supported object of each type ~~from the set Analog Input, Analog Output, and Analog Value~~) DO {

1. TRANSMIT SubscribeCOV-Request,  
'Subscriber Process Identifier' = (any value > 0 chosen by the TD),  
'Monitored Object Identifier' = X,  
'Issue Confirmed Notifications' = TRUE,  
'Lifetime' = L

2. RECEIVE BACnet-SimpleACK-PDU
3. BEFORE **Notification Fail Time**  
RECEIVE ConfirmedCOVNotification-Request,  
'Subscriber Process Identifier' = (the same value used in step 1),  
'Initiating Device Identifier' = IUT,  
'Monitored Object Identifier' = X,  
'Time Remaining' = (any value appropriate for the Lifetime selected),  
'List of Values' = (the initial Present\_Value and initial Status\_Flags)
4. TRANSMIT BACnet-SimpleACK-PDU
5. TRANSMIT ReadProperty-Request,  
'Object Identifier' = X,  
'Property Identifier' = COV\_Increment
6. RECEIVE BACnet-ComplexACK-PDU,  
'Object Identifier' = X,  
'Property Identifier' = COV\_Increment,  
'Property Value' = (a value "increment" that will be used below)
7. IF (Out\_Of\_Service is writable) THEN  
WRITE X, Out\_Of\_Service = TRUE  
~~RECEIVE BACnet-SimpleACK-PDU~~  
BEFORE **Notification Fail Time**  
RECEIVE ConfirmedCOVNotification-Request,  
'Subscriber Process Identifier' = (the same value used in step 1),  
'Initiating Device Identifier' = IUT,  
'Monitored Object Identifier' = X,  
'Time Remaining' = (any value appropriate for the Lifetime selected),  
'List of Values' = (the initial ReportedPV = the current Present\_Value, and new Status\_Flags)
- TRANSMIT BACnet-SimpleACK-PDU
8. IF (Present\_Value is now writable) THEN  
WRITE X, Present\_Value = (any value that differs from "initial Present\_Value" ReportedPV by less than "increment")  
~~RECEIVE BACnet-SimpleACK-PDU~~  
ELSE  
MAKE (Present\_Value = any value that differs from "initial Present\_Value" ReportedPV by less than "increment")
9. WAIT **Notification Fail Time**
10. CHECK (verify that no COV notification was transmitted)
11. IF (Present\_Value is now writable) THEN  
WRITE X, Present\_Value = (any value that differs from "initial Present\_Value" ReportedPV by an amount greater than "increment")  
~~RECEIVE BACnet-SimpleACK-PDU~~  
ELSE  
MAKE (Present\_Value = any value that differs from "initial Present\_Value" ReportedPV by an amount greater than "increment")
12. BEFORE **NotificationFailTime**  
RECEIVE ConfirmedCOVNotification-Request,  
'Subscriber Process Identifier' = (the same value used in step 1),  
'Initiating Device Identifier' = IUT,  
'Monitored Object Identifier' = X,  
'Time Remaining' = (any value appropriate for the Lifetime selected),  
'List of Values' = (the new Present\_Value and new Status\_Flags)
13. TRANSMIT BACnet-SimpleACK-PDU
14. TRANSMIT SubscribeCOV-Request,  
'Subscriber Process Identifier' = (the same value used in step 1),  
'Monitored Object Identifier' = X
15. RECEIVE BACnet-SimpleACK-PDU
16. IF (Out\_Of\_Service is writable) THEN  
WRITE X, Out\_Of\_Service = FALSE  
~~RECEIVE BACnet-SimpleACK-PDU~~

[Change **Clause 8.2.2**, p. 188]

[Reviewer Note: Add more primitive value objects. Updated 'Configuration Requirements'. Removed extraneous SimpleACKs after WRITE statements. Updated descriptive text for 'List of Value' property.]

## 8.2.2 Change of Value Notification ~~from an Analog Input, Analog Output, and Analog Value Object's~~ for Changes to 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, using a Lifetime of L. L shall be set to a value less than 24 hours and large enough to complete the test. 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

Configuration Requirements: At the beginning of the test, the Out\_Of\_Service property shall have a value of FALSE. *Select an object where Present\_Value is not expected to change outside the tester's control by more than COV\_Increment. If the COV\_Increment is supported, or which has a writable Out\_Of\_Service.*

Test Steps:

REPEAT X = (one supported object of each type ~~from the set Analog Input, Analog Output and Analog Value~~) DO {

1. TRANSMIT SubscribeCOV-Request,  
     'Subscriber Process Identifier' = (any value > 0 chosen by the TD),  
     'Monitored Object Identifier' = X,  
     'Issue Confirmed Notifications' = TRUE,  
     'Lifetime' = L
2. RECEIVE BACnet-SimpleACK-PDU
3. **BEFORE Notification Fail Time**  
     RECEIVE ConfirmedCOVNotification-Request,  
     'Subscriber Process Identifier' = (the same value used in step 1),  
     'Initiating Device Identifier' = IUT,  
     'Monitored Object Identifier' = X,  
     'Time Remaining' = (any value appropriate for the Lifetime selected),  
     'List of Values' = (the initial Present\_Value and initial Status\_Flags)
4. TRANSMIT BACnet-SimpleACK-PDU
5. WRITE X, Out\_Of\_Service = TRUE | WRITE X, Status\_Flags = (a value that differs from "initial Status\_Flags") |  
     MAKE (Status\_Flags = any value that differs from "initial Status\_Flags")
6. ~~IF (WriteProperty is used in step 5) THEN~~  
     ~~RECEIVE BACnet-SimpleACK-PDU~~
76. **BEFORE Notification Fail Time**  
     RECEIVE ConfirmedCOVNotification-Request,  
     'Subscriber Process Identifier' = (the same value used in step 1),  
     'Initiating Device Identifier' = IUT,  
     'Monitored Object Identifier' = X,  
     'Time Remaining' = (any value appropriate for the Lifetime selected),  
     'List of Values' = ~~(the initial~~the current Present\_Value and new Status\_Flags)
87. TRANSMIT BACnet-SimpleACK-PDU
98. TRANSMIT SubscribeCOV-Request,  
     'Subscriber Process Identifier' = (the same value used in step 1),  
     'Monitored Object Identifier' = X
109. RECEIVE BACnet-SimpleACK-PDU
110. IF (Out\_Of\_Service was changed in step 5) THEN  
     WRITE X, Out\_Of\_Service = FALSE  
     ~~RECEIVE BACnet-SimpleACK-PDU~~

[Change **Clause 8.2.3**, p. 189]

[Reviewer Note: Updated the 'Configuration Requirements'. Removed extraneous SimpleACKs that appear after WRITE statements. Modified descriptive text for 'List of Values' properties. Add clarification to test that the last COVNotification shall reflect the correct values.]

### 8.2.3 Change of Value Notification ~~from a Binary Input, Binary Output, and Binary Value Object's~~ for Changes to Present\_Value Property in Objects without a COV\_Increment

Purpose: To verify that the IUT can initiate ConfirmedCOVNotification service requests conveying a change of the Present\_Value property of ~~Binary Input, Binary Output, and Binary Value~~ objects *that do not support COV\_Increment*.

Test Concept: A subscription for COV notifications is established, using a Lifetime of L. L shall be set to a value less than 24 hours and large enough to complete the test. The Present\_Value of the monitored object is changed and a notification shall be received. The Present\_Value may be changed using the WriteProperty service or by another means such as changing the input signal represented by a Binary Input object. For some implementations it may be necessary to write to the Out\_Of\_Service property first to accomplish this task. For implementations where it is not possible to write to these properties at all the vendor shall provide an alternative trigger mechanism to accomplish this task. All of these methods are equally acceptable.

Configuration Requirements: At the beginning of the test, the Out\_Of\_Service property shall have a value of FALSE. *Select an object where Present\_Value is not expected to change outside the tester's control or which has a writable Out\_Of\_Service.*

Notes to Tester: *The IUT may initiate additional COVNotifications. The final COVNotification shall accurately reflect Present\_Value and Status\_Flags.*

Test Steps:

REPEAT X = (one supported object of each type ~~from the set Binary Input, Binary Output, and Binary Value~~) DO {

1. TRANSMIT SubscribeCOV-Request,
  - 'Subscriber Process Identifier' = (any value  $\geq 0$  chosen by the TD),
  - 'Monitored Object Identifier' = X,
  - 'Issue Confirmed Notifications' = TRUE,
  - 'Lifetime' = L
2. RECEIVE BACnet-SimpleACK-PDU
3. BEFORE **Notification Fail Time**
  - RECEIVE ConfirmedCOVNotification-Request,
    - 'Subscriber Process Identifier' = (the same value used in step 1),
    - 'Initiating Device Identifier' = IUT,
    - 'Monitored Object Identifier' = X,
    - 'Time Remaining' = (any value appropriate for the Lifetime selected),
    - 'List of Values' = (the initial Present\_Value and initial Status\_Flags)
4. TRANSMIT BACnet-SimpleACK-PDU
5. IF (Out\_Of\_Service is writable) THEN
  - WRITE X, Out\_Of\_Service = TRUE
  - BEFORE **Notification Fail Time**
    - RECEIVE ConfirmedCOVNotification-Request,
      - 'Subscriber Process Identifier' = (the same value used in step 1),
      - 'Initiating Device Identifier' = IUT,
      - 'Monitored Object Identifier' = X,
      - 'Time Remaining' = (any value appropriate for the Lifetime selected),
      - 'List of Values' = (the initial ReportedPV = the current Present\_Value, and new current Status\_Flags)
    - TRANSMIT BACnet-SimpleACK-PDU
6. IF (Present\_Value is now writable) THEN
  - WRITE X, Present\_Value = (any value that differs from "initial Present\_Value" ReportedPV)
  - ELSE
    - MAKE (Present\_Value = any value that differs from "initial Present\_Value" ReportedPV)

7. BEFORE Notification Fail Time  
RECEIVE ConfirmedCOVNotification-Request,  
    'Subscriber Process Identifier' = (the same value used in step 1),  
    'Initiating Device Identifier' = IUT,  
    'Monitored Object Identifier' = X,  
    'Time Remaining' = (any value appropriate for the Lifetime selected),  
    'List of Values' = (the new Present\_Value and new-current Status\_Flags)
8. TRANSMIT BACnet-SimpleACK-PDU
9. TRANSMIT SubscribeCOV-Request,  
    'Subscriber Process Identifier' = (the same value used in step 1),  
    'Monitored Object Identifier' = X
10. RECEIVE BACnet-SimpleACK-PDU
11. IF (Out\_Of\_Service is writable) THEN  
    WRITE X, Out\_Of\_Service = FALSE  
    RECEIVE BACnet-SimpleACK-PDU

[Remove Clause 8.2.4, p. 190]

#### **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, using a Lifetime of L. L shall be set to a value less than 24 hours and large enough to complete the test. 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.

**Configuration Requirements:** At the beginning of the test, the Out\_Of\_Service property shall have a value of FALSE.

**Test Steps:**

REPEAT X = (one supported object of each type from the set Binary Input, Binary Output, and Binary Value) DO {

1. TRANSMIT SubscribeCOV-Request,  
    'Subscriber Process Identifier' = (any value > 0 chosen by the TD),  
    'Monitored Object Identifier' = X,  
    'Issue Confirmed Notifications' = TRUE,  
    'Lifetime' = L
2. RECEIVE BACnet-SimpleACK-PDU
3. RECEIVE ConfirmedCOVNotification-Request,  
    'Subscriber Process Identifier' = (the same value used in step 1),  
    'Initiating Device Identifier' = IUT,  
    'Monitored Object Identifier' = X,  
    'Time Remaining' = (any value appropriate for the Lifetime selected),  
    'List of Values' = (the initial Present\_Value and initial Status\_Flags)
4. TRANSMIT BACnet-SimpleACK-PDU
5. WRITE X, Out\_Of\_Service = TRUE | WRITE X, Status\_Flags = (a value that differs from "initial Status\_Flags") |  
    MAKE (Status\_Flags = any value that differs from "initial Status\_Flags")
6. IF (WriteProperty is used in step 5) THEN  
    RECEIVE BACnet-SimpleACK-PDU
7. BEFORE Notification Fail Time  
    RECEIVE ConfirmedCOVNotification-Request,  
    'Subscriber Process Identifier' = (the same value used in step 1),  
    'Initiating Device Identifier' = IUT,  
    'Monitored Object Identifier' = X,  
    'Time Remaining' = (any value appropriate for the Lifetime selected),



~~\_\_\_\_\_ 'List of Values' = \_\_\_\_\_ (the initial Present\_Value and new Status\_Flags)~~  
~~8. TRANSMIT BACnet SimpleACK PDU~~  
~~9. TRANSMIT SubscribeCOV Request,~~  
~~\_\_\_\_\_ 'Subscriber Process Identifier' = \_\_\_\_\_ (the same value used in step 1),~~  
~~\_\_\_\_\_ 'Monitored Object Identifier' = \_\_\_\_\_ X~~  
~~10. RECEIVE BACnet SimpleACK PDU~~  
~~11. IF (Out\_Of\_Service was changed in step 5) THEN~~  
~~\_\_\_\_\_ WRITE X, Out\_Of\_Service = FALSE~~  
~~\_\_\_\_\_ RECEIVE BACnet SimpleACK PDU~~

[Remove Clause 8.2.5, p. 191]

### **8.2.5 Change of Value Notification from a Multi-state Input, Multi-state Output, Multi-state Value, Life Safety Point, or Life Safety Zone Object Present\_Value Property**

~~Purpose: To verify that the IUT can initiate ConfirmedCOVNotification service requests conveying a change of the Present\_Value 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, using a Lifetime of L. L shall be set to a value less than 24 hours and large enough to complete the test. The Present\_Value of the monitored object is changed and a notification shall be received. The Present\_Value may be changed using the WriteProperty service or by another means such as changing the input signal represented by the object. For some implementations it may be necessary to write to the Out\_Of\_Service property first to accomplish this task. For implementations where it is not possible to write to these properties at all, the vendor shall provide an alternative trigger mechanism to accomplish this task. All of these methods are equally acceptable.~~

~~Configuration Requirements: At the beginning of the test, the Out\_Of\_Service property shall have a value of FALSE.~~

~~Test Steps:~~

~~REPEAT X = (one supported object of each type from the set Multi-state Input, Multi-state Output, Multi-state Value, Life Safety Point, and Life Safety Zone) DO {~~

~~1. TRANSMIT SubscribeCOV Request,~~  
~~\_\_\_\_\_ 'Subscriber Process Identifier' = \_\_\_\_\_ (any value > 0 chosen by the TD),~~  
~~\_\_\_\_\_ 'Monitored Object Identifier' = \_\_\_\_\_ X,~~  
~~\_\_\_\_\_ 'Issue Confirmed Notifications' = \_\_\_\_\_ TRUE,~~  
~~\_\_\_\_\_ 'Lifetime' = \_\_\_\_\_ L~~  
~~2. RECEIVE BACnet SimpleACK PDU~~  
~~3. RECEIVE ConfirmedCOVNotification Request,~~  
~~\_\_\_\_\_ 'Subscriber Process Identifier' = \_\_\_\_\_ (the same value used in step 1),~~  
~~\_\_\_\_\_ 'Initiating Device Identifier' = \_\_\_\_\_ IUT,~~  
~~\_\_\_\_\_ 'Monitored Object Identifier' = \_\_\_\_\_ X,~~  
~~\_\_\_\_\_ 'Time Remaining' = \_\_\_\_\_ (any value appropriate for the Lifetime selected),~~  
~~\_\_\_\_\_ 'List of Values' = \_\_\_\_\_ (the initial Present\_Value and initial Status\_Flags)~~  
~~4. TRANSMIT BACnet SimpleACK PDU~~  
~~5. IF (Out\_Of\_Service is writable) THEN~~  
~~\_\_\_\_\_ WRITE X, Out\_Of\_Service = TRUE~~  
~~\_\_\_\_\_ RECEIVE BACnet SimpleACK PDU~~  
~~\_\_\_\_\_ BEFORE Notification Fail Time~~  
~~\_\_\_\_\_ RECEIVE ConfirmedCOVNotification Request,~~  
~~\_\_\_\_\_ 'Subscriber Process Identifier' = \_\_\_\_\_ (the same value used in step 1),~~  
~~\_\_\_\_\_ 'Initiating Device Identifier' = \_\_\_\_\_ IUT,~~  
~~\_\_\_\_\_ 'Monitored Object Identifier' = \_\_\_\_\_ X,~~  
~~\_\_\_\_\_ 'Time Remaining' = \_\_\_\_\_ (any value appropriate for the Lifetime selected),~~  
~~\_\_\_\_\_ 'List of Values' = \_\_\_\_\_ (the initial Present\_Value and the new Status\_Flags)~~  
~~\_\_\_\_\_ TRANSMIT BACnet SimpleACK PDU~~  
~~6. IF (Present\_Value is now writable) THEN~~  
~~\_\_\_\_\_ WRITE X, Present\_Value = (any value that differs from "initial value")~~

```

RECEIVE BACnet SimpleACK PDU
ELSE
MAKE (Present_Value = any value that differs from "initial value")
7. BEFORE Notification Fail Time
RECEIVE ConfirmedCOVNotification Request,
'Subscriber Process Identifier' = (the same value used in step 1),
'Initiating Device Identifier' = IUT,
'Monitored Object Identifier' = X,
'Time Remaining' = (any value appropriate for the Lifetime selected),
'List of Values' = (the new Present_Value and new Status_Flags)
8. TRANSMIT BACnet SimpleACK PDU
9. TRANSMIT SubscribeCOV Request,
'Subscriber Process Identifier' = (the same value used in step 1),
'Monitored Object Identifier' = X
10. RECEIVE BACnet SimpleACK PDU
11. IF (Out_Of_Service is writable) THEN
WRITE X, Out_Of_Service = FALSE
RECEIVE BACnet SimpleACK PDU

```

[Remove Clause 8.2.6, p. 193]

### 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, using a Lifetime of L. L shall be set to a value less than 24 hours and large enough to complete the test. 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.~~

~~Configuration Requirements: At the beginning of the test, the Out\_Of\_Service property shall have a value of FALSE.~~

~~Test Steps:~~

~~REPEAT X = (one supported object of each type from the set Multi-state input, Multi-state Output, Multi-state Value, Life Safety Point, and Life Safety Zone) DO {~~

```

1. TRANSMIT SubscribeCOV Request,
'Subscriber Process Identifier' = (any value > 0 chosen by the TD),
'Monitored Object Identifier' = X,
'Issue Confirmed Notifications' = TRUE,
'Lifetime' = L
2. RECEIVE BACnet SimpleACK PDU
3. RECEIVE ConfirmedCOVNotification Request,
'Subscriber Process Identifier' = (the same value used in step 1),
'Initiating Device Identifier' = IUT,
'Monitored Object Identifier' = X,
'Time Remaining' = (any value appropriate for the Lifetime selected),
'List of Values' = (the initial Present_Value and initial Status_Flags)
4. TRANSMIT BACnet SimpleACK PDU
5. WRITE X, Out_Of_Service = TRUE | WRITE X, Status_Flags = (a value that differs from "initial Status_Flags") |
MAKE (Status_Flags = any value that differs from "initial Status_Flags")
6. IF (WriteProperty is used in step 5) THEN
RECEIVE BACnet SimpleACK PDU

```

```

7. BEFORE Notification Fail Time
   RECEIVE ConfirmedCOVNotification-Request,
   'Subscriber Process Identifier' = (the same value used in step 1),
   'Initiating Device Identifier' = IUT,
   'Monitored Object Identifier' = X,
   'Time Remaining' = (any value appropriate for the Lifetime selected),
   'List of Values' = (the initial Present_Value and new Status_Flags)
8. TRANSMIT BACnet SimpleACK PDU
9. TRANSMIT SubscribeCOV Request,
   'Subscriber Process Identifier' = (the same value used in step 1),
   'Monitored Object Identifier' = X
10. RECEIVE BACnet SimpleACK PDU
11. IF (Out_Of_Service was changed in step 5) THEN
   WRITE X, Out_Of_Service = FALSE
   RECEIVE BACnet SimpleACK PDU

```

[Change and renumber **Clause 8.2.7**, p. 194]

[Reviewer Note: Added 'Configuration Requirements'. Corrected object reference in step 11. Updated wording for 'List of Values' properties. Removed extraneous SimpleACKs after WRITE statements.]

#### 8.2.78.2.4 Change of Value Notification from Loop Object Present\_Value Property

Purpose: To verify that the IUT can initiate ConfirmedCOVNotification service requests conveying a change of the Present\_Value property of a loop object.

Test Concept: A subscription for COV notifications is established, using a Lifetime of L. L shall be set to a value less than 24 hours and large enough to complete the test. The Present\_Value of the monitored object is changed by an amount less than the COV increment and it is verified that no COV notification is received. The Present\_Value is then changed by an amount greater than the COV increment and a notification shall be received.

The Present\_Value may be changed by placing the Loop Out\_Of\_Service and writing directly to the Present\_Value. For implementations where this option is not possible an alternative trigger mechanism shall be provided to accomplish this task, such as changing the Setpoint or the Setpoint\_Reference. All of these methods are equally acceptable.

The object identifier of the Loop object being tested is designated as O1 in the test steps below.

*Configuration Requirements: At the beginning of the test, the Out\_Of\_Service property shall have a value of FALSE. Select an object where Present\_Value is not expected to change outside the tester's control by more than COV\_Increment, or which has a writable Out\_Of\_Service.*

Test Steps:

1. TRANSMIT SubscribeCOV-Request,
 

'Subscriber Process Identifier' =	(any value $\geq$ 0 chosen by the TD),
'Monitored Object Identifier' =	O1,
'Issue Confirmed Notifications' =	TRUE,
'Lifetime' =	L
2. RECEIVE BACnet-SimpleACK-PDU
3. **BEFORE Notification Fail Time**

```

RECEIVE ConfirmedCOVNotification-Request,
'Subscriber Process Identifier' = (the same value used in step 1),
'Initiating Device Identifier' = IUT,
'Monitored Object Identifier' = O1,
'Time Remaining' = (any value appropriate for the Lifetime selected),
'List of Values' = (the initial Present_Value, initial Status_Flags, initial Setpoint, and initial Controlled_Variable_Value)

```
4. TRANSMIT BACnet-SimpleACK-PDU
5. TRANSMIT ReadProperty-Request,

```

        'Object Identifier' = O1,
        'Property Identifier' = COV_Increment
6.  RECEIVE BACnet-ComplexACK-PDU,
        'Object Identifier' = O1,
        'Property Identifier' = COV_Increment,
        'Property Value' = (a value "increment" that will be used below)
7.  IF (Out_Of_Service is writable) THEN
        WRITE O1, Out_Of_Service = TRUE
        BEFORE Notification Fail Time
            RECEIVE ConfirmedCOVNotification-Request,
                'Subscriber Process Identifier' = (the same value used in step 1),
                'Initiating Device Identifier' = IUT,
                'Monitored Object Identifier' = O1,
                'Time Remaining' = (any value appropriate for the Lifetime selected),
                'List of Values' = (the initial ReportedPV = the current Present_Value, new Status_Flags,
                                initial current Setpoint, and initial current Controlled_Variable_Value)
            TRANSMIT BACnet-SimpleACK-PDU
8.  IF (Present_Value is now writable) THEN
        WRITE O1, Present_Value = (any value that differs from "initial Present_Value" ReportedPV by less than
                                "increment")
    ELSE
        MAKE (Present_Value = any value that differs from "initial Present_Value" ReportedPV by less than "increment")
9.  WAIT Notification Fail Time
10. CHECK (verify that no COV notification was transmitted)
11. IF (Present_Value is now writable) THEN
        WRITE O1, Present_Value = (any value that differs from "initial Present_Value" ReportedPV by an amount greater
                                than "increment")
    ELSE
        MAKE (Present_Value = any value that differs from "initial Present_Value" ReportedPV by an amount greater than
                                "increment")
12. BEFORE Notification Fail Time
        RECEIVE ConfirmedCOVNotification-Request,
            'Subscriber Process Identifier' = (the same value used in step 1),
            'Initiating Device Identifier' = IUT,
            'Monitored Object Identifier' = O1,
            'Time Remaining' = (any value appropriate for the Lifetime selected),
            'List of Values' = (the new Present_Value, new Status_Flags, initial current Setpoint, and
                            initial current Controlled_Variable_Value)
13. TRANSMIT BACnet-SimpleACK-PDU
14. TRANSMIT SubscribeCOV-Request,
        'Subscriber Process Identifier' = (the same value used in step 1),
        'Monitored Object Identifier' = O1
15. RECEIVE BACnet-SimpleACK-PDU
16. IF (Out_Of_Service is writable) THEN
        WRITE L O1, Out_Of_Service = FALSE

```

[Remove **Clause 8.2.8**, p. 195]

### **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, using a Lifetime of L. L shall be set to a value less than 24 hours and large enough to complete the test. 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 object identifier of the Loop object being tested is designated as O1 in the test steps below.

**Configuration Requirements:** At the beginning of the test, the Out\_Of\_Service property shall have a value of FALSE.

**Test Steps:**

1. TRANSMIT SubscribeCOV Request,  
     'Subscriber Process Identifier' = \_\_\_\_\_ (any value > 0 chosen by the TD),  
     'Monitored Object Identifier' = \_\_\_\_\_ O1,  
     'Issue Confirmed Notifications' = \_\_\_\_\_ TRUE,  
     'Lifetime' = \_\_\_\_\_ L
2. RECEIVE BACnet SimpleACK PDU
3. RECEIVE ConfirmedCOVNotification Request,  
     'Subscriber Process Identifier' = \_\_\_\_\_ (the same value used in step 1),  
     'Initiating Device Identifier' = \_\_\_\_\_ IUT,  
     'Monitored Object Identifier' = \_\_\_\_\_ O1,  
     'Time Remaining' = \_\_\_\_\_ (any value appropriate for the Lifetime selected),  
     'List of Values' = \_\_\_\_\_ (the initial Present\_Value, initial Status\_Flags, initial Setpoint, and  
     initial Controlled\_Variable\_Value)
4. TRANSMIT BACnet SimpleACK PDU
5. WRITE O1, Out\_Of\_Service = TRUE | WRITE O1, Status\_Flags = (a value that differs from "initial Status\_Flags") |  
     MAKE (Status\_Flags = any value that differs from "initial Status\_Flags")
6. IF (WriteProperty is used in step 5) THEN  
     RECEIVE BACnet SimpleACK PDU
7. BEFORE **Notification Fail Time**  
     RECEIVE ConfirmedCOVNotification Request,  
     'Subscriber Process Identifier' = \_\_\_\_\_ (the same value used in step 1),  
     'Initiating Device Identifier' = \_\_\_\_\_ IUT,  
     'Monitored Object Identifier' = \_\_\_\_\_ O1,  
     'Time Remaining' = \_\_\_\_\_ (any value appropriate for the Lifetime selected),  
     'List of Values' = \_\_\_\_\_ (the initial Present\_Value, new Status\_Flags, initial Setpoint, and initial  
     Controlled\_Variable\_Value)
8. TRANSMIT BACnet SimpleACK PDU
9. TRANSMIT SubscribeCOV Request,  
     'Subscriber Process Identifier' = \_\_\_\_\_ (the same value used in step 1),  
     'Monitored Object Identifier' = \_\_\_\_\_ O1
10. RECEIVE BACnet SimpleACK PDU
11. IF (Out\_Of\_Service was changed in step 5) THEN  
     WRITE L, Out\_Of\_Service = FALSE  
     RECEIVE BACnet SimpleACK PDU

[Renumber **Clause 8.2.9**, p. 196]

### 8.2.98.2.5 Missing Lifetime Test

**Purpose:** This test case verifies the special case of the SubscribeCOV where a missing Lifetime parameter shall imply an indefinite Lifetime subscription.

**Test Concept:** A subscription for COV notification is established with the IUT. The subscribe message shall omit the Lifetime parameter. The COV notification is received from the IUT and the 'Time Remaining' value is verified to be 0.

**Test Steps:**

1. TRANSMIT SubscribeCOV-Request,  
     'Subscriber Process Identifier' = \_\_\_\_\_ (any value > 0 chosen by the TD),  
     'Monitored Object Identifier' = \_\_\_\_\_ X,  
     'Issue Confirmed Notifications' = \_\_\_\_\_ TRUE
2. RECEIVE BACnet-SimpleACK-PDU,

3. RECEIVE ConfirmedCOVNotification-Request,
  - 'Subscriber Process Identifier' = (the same value used in step 1),
  - 'Initiating Device Identifier' = IUT,
  - 'Monitored Object Identifier' = X,
  - 'Time Remaining' = 0,
  - 'List of Values' = (the initial Present\_Value and initial Status\_Flags)
4. TRANSMIT BACnet-SimpleACK-PDU

[Add new **Clause 8.2.X9**, p. 196]

### 8.2.X9 ConfirmedCOVNotification Pulse Converter changing Present\_Value

**Purpose:** To verify the correct operation of COV in the Pulse Converter object. The Pulse Converter initiates periodic COV Notifications every COV\_Period, even when there are no changes in the object, in addition to the COV notifications that this object type generates due to changes in the Present\_Value property.

**Test Concept:** A subscription for COV notifications is established, using a Lifetime of L. L shall be set to a value less than 24 hours and large enough to complete the test. The Present\_Value of the monitored object is changed by an amount less than the COV increment, and it is verified that no COV notification is received. The Present\_Value property can be changed by using the WriteProperty service or by another means. For some implementations, writing to the Out\_Of\_Service property will enable the Present\_Value property to be changed by the WriteProperty service. The object identifier of the Pulse Converter object being tested is designated as O1 in the test steps below.

**Configuration Requirements:** At the beginning of the test, the Out\_Of\_Service property shall have a value of FALSE. Select an object where Present\_Value is not expected to change outside the tester's control by more than COV\_Increment or which has a writable Out\_Of\_Service.

**Notes to Tester:** The IUT may initiate additional COVNotifications. The final COVNotification shall accurately reflect Present\_Value and Status\_Flags.

**Test Steps:**

1. TRANSMIT SubscribeCOV-Request,
  - 'Subscriber Process Identifier' = (any value > 0 chosen by the TD),
  - 'Monitored Object Identifier' = O1,
  - 'Issue Confirmed Notifications' = TRUE,
  - 'Lifetime' = L
2. RECEIVE BACnet-SimpleACK-PDU
3. BEFORE **Notification Fail Time**
  - RECEIVE ConfirmedCOVNotification-Request,
    - 'Subscriber Process Identifier' = (the same value used in step 1),
    - 'Initiating Device Identifier' = IUT,
    - 'Monitored Object Identifier' = O1,
    - 'Time Remaining' = (any value appropriate for the Lifetime selected),
    - 'List of Values' = (the initial Present\_Value, initial Status\_Flags, and Update\_Time)
4. TRANSMIT BACnet-SimpleACK-PDU
5. TRANSMIT ReadProperty-Request,
  - 'Object Identifier' = O1,
  - 'Property Identifier' = COV\_Increment
6. RECEIVE BACnet-ComplexACK-PDU,
  - 'Object Identifier' = O1,
  - 'Property Identifier' = COV\_Increment,
  - 'Property Value' = (a value "increment" that will be used below)
7. IF (Out\_Of\_Service is writable) THEN
  - WRITE O1, Out\_Of\_Service = TRUE
  - BEFORE **Notification Fail Time**
    - RECEIVE ConfirmedCOVNotification-Request,
      - 'Subscriber Process Identifier' = (the same value used in step 1),

```

        'Initiating Device Identifier' = IUT,
        'Monitored Object Identifier' = O1,
        'Time Remaining' = (any value appropriate for the Lifetime selected),
        'List of Values' = (ReportedPV = the current Present_Value, new Status_Flags, and current
Update_Time)
8. TRANSMIT BACnet-SimpleACK-PDU
9. IF (Present_Value is now writable) THEN
    WRITE O1, Present_Value = (any value that differs from ReportedPV by less than "increment")
ELSE
    MAKE (Present_Value = any value that differs from ReportedPV by less than "increment")
10. WAIT Notification Fail Time
11. CHECK (verify that no COV notification was transmitted)
12. IF (Present_Value is now writable) THEN
    WRITE O1, Present_Value = (any value that differs from ReportedPV by an amount greater than "increment")
ELSE
    MAKE (Present_Value = any value that differs from ReportedPV by an amount greater than "increment")

13. BEFORE Notification Fail Time
    RECEIVE ConfirmedCOVNotification-Request,
        'Subscriber Process Identifier' = (the same value used in step 1),
        'Initiating Device Identifier' = IUT,
        'Monitored Object Identifier' = O1,
        'Time Remaining' = (any value appropriate for the Lifetime selected),
        'List of Values' = (the new Present_Value, new Status_Flags, and current Update_Time)
14. TRANSMIT BACnet-SimpleACK-PDU
15. TRANSMIT SubscribeCOV-Request,
    'Subscriber Process Identifier' = (the same value used in step 1),
    'Monitored Object Identifier' = O1
16. RECEIVE BACnet-SimpleACK-PDU
17. IF (Out_Of_Service was changed in step 7) THEN
    WRITE O1, Out_Of_Service = FALSE

```

[Add new **Clause 8.2.X10**, p. 196]

### 8.2.X10 ConfirmedCOVNotification Pulse Converter changing Status\_Flags

**Purpose:** To verify the correct operation of COV in the Pulse Converter object. The Pulse Converter initiates periodic COV Notifications every COV\_Period, even when there are no changes in the object, in addition to the COV notifications that this object type generates due to changes in the Status\_Flags property.

**Test Concept:** A subscription for COV notifications is established, using a Lifetime of L. L shall be set to a value less than 24 hours and large enough to complete the test. The Status\_Flags property of the monitored object is then changed, and a notification shall be received. For some implementations writing to the Out\_Of\_Service property will accomplish this task. For implementations where it is not possible to write Out\_Of\_Service or change the Status\_Flags by any other means, this test shall be skipped. The object identifier of the Pulse Converter object being tested is designated as O1 in the test steps below.

**Configuration Requirements:** At the beginning of the test, the Out\_Of\_Service property shall have a value of FALSE. Select an object where Present\_Value is not expected to change outside the tester's control by more than COV\_Increment. COV\_Period is configured high enough that it does not trigger many COV notifications during the execution of the test.

**Test Steps:**

```

1. TRANSMIT SubscribeCOV-Request,
    'Subscriber Process Identifier' = (any value > 0 chosen by the TD),
    'Monitored Object Identifier' = O1,
    'Issue Confirmed Notifications' = TRUE,
    'Lifetime' = L
2. RECEIVE BACnet-SimpleACK-PDU
3. BEFORE Notification Fail Time

```

```

RECEIVE ConfirmedCOVNotification-Request,
  'Subscriber Process Identifier' = (the same value used in step 1),
  'Initiating Device Identifier' = IUT,
  'Monitored Object Identifier' = O1,
  'Time Remaining' = (any value appropriate for the Lifetime selected),
  'List of Values' = (the initial Present_Value, initial Status_Flags, and
                     Update_Time)
4. TRANSMIT BACnet-SimpleACK-PDU
5. IF (Out_Of_Service is writable) THEN
    WRITE Out_Of_Service = TRUE
  ELSE
    MAKE (Status_Flags = any value that differs from initial Status_Flags)
6. BEFORE Notification Fail Time
  RECEIVE ConfirmedCOVNotification-Request,
    'Subscriber Process Identifier' = (the same value used in step 1),
    'Initiating Device Identifier' = IUT,
    'Monitored Object Identifier' = O1,
    'Time Remaining' = (any value appropriate for the Lifetime selected),
    'List of Values' = (the current Present_Value, new Status_Flags, and Update_Time)
7. TRANSMIT BACnet-SimpleACK-PDU
8. TRANSMIT SubscribeCOV-Request,
  'Subscriber Process Identifier' = (the same value used in step 1),
  'Monitored Object Identifier' = O1
9. RECEIVE BACnet-SimpleACK-PDU
10. IF (Out_Of_Service is writable) THEN
    WRITE Out_Of_Service = FALSE

```

[Add new **Clause 8.2.X11**, p. 196]

## 8.2.X11 Change of Value Notification from an Access Door object Present\_Value, Status\_Flags and Door\_Alarm\_State property

Purpose: To verify that the IUT can initiate ConfirmedCOVNotification service requests conveying a change of the Present\_Value property of Access Door objects.

Test Concept: A subscription for COV notifications is established, using a Lifetime of L. L shall be set to a value less than 24 hours and large enough to complete the test. The Present\_Value of the monitored object is changed, and a notification shall be received. The Present\_Value may be changed using the WriteProperty service or by another means. For some implementations it may be necessary to write to the Out\_Of\_Service property first to accomplish this task. For implementations where it is not possible to write to these properties at all, the vendor shall provide an alternative trigger mechanism to accomplish this task. All these methods are equally acceptable.

Configuration Requirements: At the beginning of the test, the Out\_Of\_Service property shall have a value of FALSE. Select an object where Present\_Value is not expected to change outside the tester's control, or which has a writable Out\_Of\_Service. If no object has a Door\_Alarm\_State property, then steps 9,10, and 11 shall be skipped. For implementations where it is not possible to write Out\_Of\_Service or change the Status\_Flags by any other means, steps 5,6, and 7 shall be skipped

Test Steps:

```

REPEAT X = (one supported object of type Access Door) DO {
1. TRANSMIT SubscribeCOV-Request,
  'Subscriber Process Identifier' = (any value > 0 chosen by the TD),
  'Monitored Object Identifier' = X,
  'Issue Confirmed Notifications' = TRUE,
  'Lifetime' = L
2. RECEIVE BACnet-SimpleACK-PDU
3. BEFORE Notification Fail Time
  RECEIVE ConfirmedCOVNotification-Request,
    'Subscriber Process Identifier' = (the same value used in step 1),

```



- 'Initiating Device Identifier' = IUT,
- 'Monitored Object Identifier' = X,
- 'Time Remaining' = (any value appropriate for the Lifetime selected),
- 'List of Values' = (the initial Present\_Value, initial Status\_Flags, and Door\_Alarm\_State if X has a Door\_Alarm\_State property)
- 4. TRANSMIT BACnet-SimpleACK-PDU
- 5. IF (Out\_Of\_Service is writable) THEN
  - WRITE Out\_Of\_Service = TRUE
  - ELSE
    - MAKE (Status\_Flags = any value that differs from initial Status\_Flags)
- 6. BEFORE **Notification Fail Time**
  - RECEIVE ConfirmedCOVNotification-Request,
  - 'Subscriber Process Identifier' = (the same value used in step 1),
  - 'Initiating Device Identifier' = IUT,
  - 'Monitored Object Identifier' = X,
  - 'Time Remaining' = (any value appropriate for the Lifetime selected),
  - 'List of Values' = (ReportedPV=current Present\_Value, new Status\_Flags, and Door\_Alarm\_State if X has a Door\_Alarm\_State property)
- 7. TRANSMIT BACnet-SimpleACK-PDU
- 8. IF (Present\_Value is writable) THEN
  - WRITE X,Present\_Value = (any value that differs from ReportedPV)
  - ELSE
    - MAKE (Present\_Value = any value that differs from ReportedPV)
- 9. BEFORE **Notification Fail Time**
  - RECEIVE ConfirmedCOVNotification-Request,
  - 'Subscriber Process Identifier' = (the same value used in step 1),
  - 'Initiating Device Identifier' = IUT,
  - 'Monitored Object Identifier' = X,
  - 'Time Remaining' = (any value appropriate for the Lifetime selected),
  - 'List of Values' = (the new Present\_Value, new Status\_Flags, and Door\_Alarm\_State if X has a Door\_Alarm\_State property)
- 10. TRANSMIT BACnet-SimpleACK-PDU
- 11. IF (Door\_Alarm\_State is now writable) THEN
  - WRITE Door\_Alarm\_State = (any value that differs from its initial Door\_Alarm\_State)
  - ELSE
    - MAKE (Door\_Alarm\_State = any value that differs from its initial Door\_Alarm\_State)
- 12. BEFORE **Notification Fail Time**
  - RECEIVE ConfirmedCOVNotification-Request,
  - 'Subscriber Process Identifier' = (the same value used in Step 1),
  - 'Initiating Device Identifier' = IUT,
  - 'Monitored Object Identifier' = X,
  - 'Time Remaining' = (any value appropriate for the Lifetime selected),
  - 'List of Values' = (the new Present\_Value, new Status\_Flags, and Door\_Alarm\_State)
- 13. TRANSMIT BACnet-SimpleACK-PDU
- 14. TRANSMIT SubscribeCOV-Request,
  - 'Subscriber Process Identifier' = (the same value used in the SubscribeCOV-Request),
  - 'Monitored Object Identifier' = X
- 15. RECEIVE BACnet-SimpleACK-PDU
- 16. IF (Out\_Of\_Service is writable) THEN
  - WRITE Out\_Of\_Service = FALSE

[Add new **Clause 8.2.X12**, p. 196]

## 8.2.X12 Change of Value Notification from an Access Point object

Purpose: To verify that the IUT can initiate ConfirmedCOVNotification service requests conveying a change of the Status\_Flags and Access\_Event\_Time properties of Access Point objects.

Test Concept: A subscription for COV notifications is established, using a Lifetime of L. L shall be set to a value less than 24 hours and large enough to complete the test. The Access\_Event\_Time and Status\_Flags of the monitored object is changed, and a notification shall be received. The properties may be changed using the WriteProperty service or by another means. For some implementations it may be necessary to write to the Out\_Of\_Service property first to accomplish this task. For implementations where it is not possible to write to these properties at all, the vendor shall provide an alternative trigger mechanism to accomplish this task. All of these methods are equally acceptable. For implementations where it is not possible to write Out\_Of\_Service or change the Status\_Flags by any other means, steps 5,6, and 7 shall be skipped.

Configuration Requirements: At the beginning of the test, the Out\_Of\_Service property shall have a value of FALSE.

Test Steps:

1. TRANSMIT SubscribeCOV-Request,  
     'Subscriber Process Identifier' = (PI: any value > 0 chosen by the TD),  
     'Monitored Object Identifier' = X,  
     'Issue Confirmed Notifications' = TRUE,  
     'Lifetime' = L
2. RECEIVE BACnet-SimpleACK-PDU
3. BEFORE **Notification Fail Time**  
     RECEIVE ConfirmedCOVNotification-Request,  
     'Subscriber Process Identifier' = PI,  
     'Initiating Device Identifier' = IUT,  
     'Monitored Object Identifier' = X,  
     'Time Remaining' = (any value appropriate for the Lifetime selected),  
     'List of Values' = (the initial Access\_Event, Status\_Flags, Access\_Event\_Tag,  
                             Access\_Event\_Time, Access\_Event\_Credential and  
                             Access\_Event\_Authentication\_Factor if X has an  
                             Access\_Event\_Authentication\_Factor property)
4. TRANSMIT BACnet-SimpleACK-PDU
5. IF (Out\_Of\_Service is writable) THEN  
     WRITE X, Out\_Of\_Service = TRUE  
   ELSE  
     MAKE (Status\_Flags = any value that differs from initial Status\_Flags)
6. BEFORE **Notification Fail Time**  
     RECEIVE ConfirmedCOVNotification-Request,  
     'Subscriber Process Identifier' = PI,  
     'Initiating Device Identifier' = IUT,  
     'Monitored Object Identifier' = X,  
     'Time Remaining' = (any value appropriate for the Lifetime selected),  
     'List of Values' = (the initial Access\_Event, new Status\_Flags, initial Access\_Event\_Tag,  
                             Access\_Event\_Time, Access\_Event\_Credential and  
                             Access\_Event\_Authentication\_Factor if X has a Access\_Event\_Authentication\_Factor  
                             property)
7. TRANSMIT BACnet-SimpleACK-PDU
8. IF (Access\_Event\_Time is now writable) THEN  
     WRITE Access\_Event\_Time = (any value that differs from initial Access\_Event\_Time)  
   ELSE  
     MAKE (Access\_Event\_Time = any value that differs from initial Access\_Event\_Time)
9. BEFORE **Notification Fail Time**  
     RECEIVE ConfirmedCOVNotification-Request,  
     'Subscriber Process Identifier' = PI,  
     'Initiating Device Identifier' = IUT,  
     'Monitored Object Identifier' = X,  
     'Time Remaining' = (any value appropriate for the Lifetime selected),  
     'List of Values' = (the new values of Access\_Event, Access\_Event\_Tag, Access\_Event\_Time,  
                             Access\_Event\_Credential, and Access\_Event\_Authentication\_Factor if X has  
                             Access\_Event\_Authentication\_Factor property)
10. TRANSMIT BACnet-SimpleACK-PDU
11. TRANSMIT SubscribeCOV-Request,



MAKE (Present\_Value = any value that differs from initial Present\_Value)

9. BEFORE **Notification Fail Time**

RECEIVE ConfirmedCOVNotification-Request,

'Subscriber Process Identifier' = PI,

'Initiating Device Identifier' = IUT,

'Monitored Object Identifier' = X,

'Time Remaining' = (any value appropriate for the Lifetime selected),

'List of Values' = (the new Present\_Value, new Status\_Flags, and Update\_Time

(most recent update time when the Present\_Value was updated))

10. TRANSMIT BACnet-SimpleACK-PDU

11. Verify Update\_Time received in step 7.

12. TRANSMIT SubscribeCOV-Request,

'Subscriber Process Identifier' = PI,

'Monitored Object Identifier' = X

13. RECEIVE BACnet-SimpleACK-PDU

14. IF (Out\_Of\_Service is writable) THEN

WRITE Out\_Of\_Service = FALSE

15. CHECK (verify that no notification message has been transmitted)

[Add new **Clause 8.2.X17**, p. 196]

### 8.2.X17 Change of Value Notification of Staging Object Present\_Value Property

Purpose: To verify that the IUT can initiate ConfirmedCOVNotification service requests conveying a change of the Present\_Value property in Staging objects that support COV\_Increment.

Test Concept: A CPV subscription is established, using a Lifetime of L. L shall be set to a value less than 24 hours and large enough to complete the test. The Present\_Value of the monitored object is changed by an amount less than the COV increment, and it is verified that no COV notification is received. The Present\_Value is then changed by an amount greater than the COV increment, and a notification shall be received.

Configuration Requirements: Select a Staging object where Present\_Value is not expected to change outside the tester's control. The object is configured such that the change in Present\_Value required to change stages is larger than COV\_Increment. If the IUT cannot be configured with such a Staging object, this test shall be skipped.

Test Steps:

1. READ PV1 = Present\_Value

2. READ SF1 = Status\_Flags

3. READ PS1 = Present\_Stage

-- subscribe for COV and receive initial notification

4. TRANSMIT SubscribeCOV-Request,

'Subscriber Process Identifier' = (PID1: any value > 0 chosen by the TD),

'Monitored Object Identifier' = X,

'Issue Confirmed Notifications' = TRUE,

'Lifetime' = L

5. RECEIVE BACnet-SimpleACK-PDU

6. BEFORE **Notification Fail Time**

RECEIVE ConfirmedCOVNotification-Request,

'Subscriber Process Identifier' = PID1,

'Initiating Device Identifier' = IUT,

'Monitored Object Identifier' = X,

'Time Remaining' = (any value appropriate for the Lifetime selected),

'List of Values' = (PV1, SF1, PS1)

7. TRANSMIT BACnet-SimpleACK-PDU

```
-- change Present_Value by less than COV_Increment, and not enough to change the stage
8.  WRITE X, Present_Value = (PV2: a value that differs from PV1 by less than COV_Increment and which is in
    the range for the current stage)

9.  WAIT Notification Fail Time
10. CHECK (verify that no COV notification was transmitted)

-- change Present_Value by more than COV_Increment, but not enough to change the stage
11. WRITE X, Present_Value = (PV3: a value that differs from PV1 by an amount greater than COV_Increment
    and which is in the range for the current stage)

12. BEFORE NotificationFailTime
    RECEIVE ConfirmedCOVNotification-Request,
        'Subscriber Process Identifier' = PID1,
        'Initiating Device Identifier' = IUT,
        'Monitored Object Identifier' = X,
        'Time Remaining' = (any value appropriate for the Lifetime selected),
        'List of Values' = (PV3, SF1, PS1)
13. TRANSMIT BACnet-SimpleACK-PDU

-- cleanup the subscription
14. TRANSMIT SubscribeCOV-Request,
    'Subscriber Process Identifier' = PID1,
    'Monitored Object Identifier' = X
15. RECEIVE BACnet-SimpleACK-PDU
```

[Add new **Clause 8.2.X18**, p. 196]

## 8.2.X18 Change of Value Notification from a Staging Object Status\_Flags Property

Purpose: To verify that the IUT can initiate ConfirmedCOVNotification service requests conveying a change of the Status\_Flags property of Staging objects.

Test Concept: A COV subscription is established, using a Lifetime of L. L shall be set to a value less than 24 hours and large enough to complete the test. 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 implementations where it is not possible to write Out\_Of\_Service or change the Status\_Flags by any other means, this test shall be skipped.

Configuration Requirements: Select a Staging object where Present\_Value is not expected to change outside the tester's control.

Test Steps:

```
1. VERIFY Out_Of_Service = FALSE
2. READ PV1 = Present_Value
3. READ SF1 = Status_Flags
4. READ PS1 = Present_Stage

-- subscribe for COV and receive initial notification
5.  TRANSMIT SubscribeCOV-Request,
    'Subscriber Process Identifier' = (PID1: any value > 0 chosen by the TD),
    'Monitored Object Identifier' = X,
    'Issue Confirmed Notifications' = TRUE,
    'Lifetime' = L
6.  RECEIVE BACnet-SimpleACK-PDU
7.  BEFORE Notification Fail Time
    RECEIVE ConfirmedCOVNotification-Request,
        'Subscriber Process Identifier' = PID1,
        'Initiating Device Identifier' = IUT,
        'Monitored Object Identifier' = X,
        'Time Remaining' = (any value appropriate for the Lifetime selected),
```

```

        'List of Values' = (PV1, SF1, PS1)
8. TRANSMIT BACnet-SimpleACK-PDU

-- change Status_Flags and receive notification
9. IF Out_Of_Service is writable THEN
    WRITE X, Out_Of_Service = TRUE
    SF2 = (SF1 with the Out_Of_Service bit changed to 1)
ELSE
    MAKE (Status_Flags = SF2, any value other than SF1)
10. BEFORE Notification Fail Time
    RECEIVE ConfirmedCOVNotification-Request,
        'Subscriber Process Identifier' = PID1,
        'Initiating Device Identifier' = IUT,
        'Monitored Object Identifier' = X,
        'Time Remaining' = (any value appropriate for the Lifetime selected),
        'List of Values' = (PV1, SF2, PS1)
11. TRANSMIT BACnet-SimpleACK-PDU

-- cleanup the subscription and Out_Of_Service
12. TRANSMIT SubscribeCOV-Request,
    'Subscriber Process Identifier' = PID1,
    'Monitored Object Identifier' = X
13. RECEIVE BACnet-SimpleACK-PDU
14. IF (Out_Of_Service was changed via writing) THEN
    WRITE X, Out_Of_Service = FALSE

```

[Add new **Clause 8.2.X19**, p. 196]

### 8.2.X19 Change of Value Notification from a Staging Object Present\_Stage Property

Purpose: To verify that the IUT can initiate ConfirmedCOVNotification service requests conveying a change of the Present\_Stage property of Staging objects.

Test Concept: A COV subscription is established, using a Lifetime of L. L shall be set to a value less than 24 hours and large enough to complete the test. The Present\_Stage property of the monitored object is then changed, and a notification shall be received.

Configuration Requirements: Select a Staging object, O1, where Present\_Value is not expected to change outside the tester's control. The object shall be configured with Present\_Value having a value, PV1, which is less than COV\_Increment away from a value, PV2, which will change the current stage to a new stage, PS2. If no Staging object can be configured with a COV\_increment larger than the resolution of Present\_Value, this test shall be skipped.

Test Steps:

```

1. VERIFY Present_Value = PV1
2. VERIFY Present_Stage = PS2
3. READ SF1 = Status_Flags
4. CHECK( The difference between PV1 and PV2 is less than COV_Increment)

-- subscribe for COV and receive initial notification
5. TRANSMIT SubscribeCOV-Request,
    'Subscriber Process Identifier' = (PID1: any value > 0 chosen by the TD),
    'Monitored Object Identifier' = O1,
    'Issue Confirmed Notifications' = TRUE,
    'Lifetime' = L
6. RECEIVE BACnet-SimpleACK-PDU
7. BEFORE Notification Fail Time
    RECEIVE ConfirmedCOVNotification-Request,
        'Subscriber Process Identifier' = PID1,

```

'Initiating Device Identifier' = IUT,  
 'Monitored Object Identifier' = O1,  
 'Time Remaining' = (any value appropriate for the Lifetime selected),  
 'List of Values' = (PV1, SF1, PS1)

#### 8. TRANSMIT BACnet-SimpleACK-PDU

-- change Present\_Value and receive notification

#### 9. WRITE X, Present\_Value = PV2

#### 10. BEFORE Notification Fail Time

RECEIVE ConfirmedCOVNotification-Request,

'Subscriber Process Identifier' = PID1,  
 'Initiating Device Identifier' = IUT,  
 'Monitored Object Identifier' = O1,  
 'Time Remaining' = (any value appropriate for the Lifetime selected),  
 'List of Values' = (PV2, SF1, PS2)

#### 11. TRANSMIT BACnet-SimpleACK-PDU

-- cleanup the subscription

#### 12. TRANSMIT SubscribeCOV-Request,

'Subscriber Process Identifier' = PID1,  
 'Monitored Object Identifier' = O1

#### 13. RECEIVE BACnet-SimpleACK-PDU

### 8.3 UnconfirmedCOVNotification Service Initiation Tests

[Change **Clause 8.3.1**, p. 196]

[Reviewer Note: Modified test for covering more primitive value objects and the Lighting Output Object. Added clarification to test that the last COVNotification shall reflect the correct values.]

#### 8.3.1 Change of Value Notification ~~from an Analog Input, Analog Output, and Analog Value Object's~~ for changes to Present\_Value Property in Objects with a COV\_Increment

Purpose: To verify that the IUT can initiate UnconfirmedCOVNotification service requests conveying a change of the Present\_Value property ~~of Analog Input, Analog Output, and Analog Value objects.~~

Test Steps: The steps for this test case are identical to the test steps in 8.2.1 except that the SubscribeCOV service request in step 1 shall have a value of FALSE for the 'Issue Confirmed Notifications' parameter, all of the ConfirmedCOVNotification requests shall be UnconfirmedCOVNotification requests, and there is no acknowledgment of the unconfirmed services. The MAC address used for the notification message shall be such that the TD is one of the recipients.

*Notes to Tester: The IUT may initiate additional COVNotifications. The final COVNotification shall accurately reflect Present\_Value and Status\_Flags.*

[Change **Clause 8.3.2**, p. 197]

[Reviewer Note: Modified test for covering more primitive value objects.]

#### 8.3.2 Change of Value Notification ~~from an Analog Input, Analog Output, and Analog Value Object's~~ for Changes to Status\_Flags Property

Purpose: To verify that the IUT can initiate UnconfirmedCOVNotification service requests conveying a change of the Status\_Flags property ~~of Analog Input, Analog Output, and Analog Value objects.~~

Test Steps: The steps for this test case are identical to the test steps in 8.2.2 except that the SubscribeCOV service request in step 1 shall have a value of FALSE for the 'Issue Confirmed Notifications' parameter, all of the ConfirmedCOVNotification requests shall be UnconfirmedCOVNotification requests, and there is no acknowledgment of the unconfirmed services. The MAC address used for the notification message shall be such that the TD is one of the recipients.

[Change **Clause 8.3.3**, p. 197]

[Reviewer Note: Modified test for covering more primitive value objects. Added clarification to test that the last COVNotification shall reflect the correct values.]

### **8.3.3 Change of Value Notification from a Binary Input, Binary Output, and Binary Value Object's for Changes to Present\_Value Property in Objects without a COV\_Increment**

Purpose: To verify that the IUT can initiate UnconfirmedCOVNotification service requests conveying a change of the Present\_Value property ~~of Binary Input, Binary Output, and Binary Value objects of objects that do not support COV\_Increment.~~

Test Steps: The steps for this test case are identical to the test steps in 8.2.3 except that the SubscribeCOV service request in step 1 shall have a value of FALSE for the 'Issue Confirmed Notifications' parameter, all of the ConfirmedCOVNotification requests shall be UnconfirmedCOVNotification requests, and there is no acknowledgment of the unconfirmed services. The MAC address used for the notification message shall be such that the TD is one of the recipients.

*Notes to Tester: The IUT may initiate additional COVNotifications. The final COVNotification shall accurately reflect Present\_Value and Status\_Flags.*

[Remove **Clause 8.3.4**, p. 197]

### **8.3.4 ~~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 UnconfirmedCOVNotification service requests conveying a change of the Status\_Flags property ~~of Binary Input, Binary Output, and Binary Value objects.~~

Test Steps: The steps for this test case are identical to the test steps in 8.2.4 except that the SubscribeCOV service request in step 1 shall have a value of FALSE for the 'Issue Confirmed Notifications' parameter, all of the ConfirmedCOVNotification requests shall be UnconfirmedCOVNotification requests, and there is no acknowledgment of the unconfirmed services. The MAC address used for the notification message shall be such that the TD is one of the recipients.

[Remove **Clause 8.3.5**, p. 197]

### **8.3.5 ~~Change of Value Notification from a Multi-state Input, Multi-state Output, Multi-state Value, Life Safety Point, and Life Safety Zone Object Present\_Value Property~~**

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

Test Steps: The steps for this test case are identical to the test steps in 8.2.5 except that the SubscribeCOV service request in step 1 shall have a value of FALSE for the 'Issue Confirmed Notifications' parameter, all of the ConfirmedCOVNotification requests shall be UnconfirmedCOVNotification requests, and there is no acknowledgment of the unconfirmed services. The MAC address used for the notification message shall be such that the TD is one of the recipients.

[Remove **Clause 8.3.6**, p. 197]



### **~~8.3.6 — 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 UnconfirmedCOVNotification 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 Steps: The steps for this test case are identical to the test steps in 8.2.6 except that the SubscribeCOV service request in step 1 shall have a value of FALSE for the 'Issue Confirmed Notifications' parameter, all of the ConfirmedCOVNotification requests shall be UnconfirmedCOVNotification requests, and there is no acknowledgment of the unconfirmed services. The MAC address used for the notification message shall be such that the TD is one of the recipients.~~

[Change and renumber **Clause 8.3.7**, p. 198]

[Reviewer Note: Add clarification to test that the last COVNotification shall reflect the correct values.]

### **8.3.7.3.4 Change of Value Notification from Loop Object Present\_Value Property**

Purpose: To verify that the IUT can initiate UnconfirmedCOVNotification service requests conveying a change of the Present\_Value property of a Loop object.

Test Steps: The steps for this test case are identical to the test steps in 8.2.7 except that the SubscribeCOV service request in step 1 shall have a value of FALSE for the 'Issue Confirmed Notifications' parameter, all of the ConfirmedCOVNotification requests shall be UnconfirmedCOVNotification requests, and there is no acknowledgment of the unconfirmed services. The MAC address used for the notification message shall be such that the TD is one of the recipients.

*Notes to Tester: The IUT may initiate additional COVNotifications. The final COVNotification shall accurately reflect Present\_Value and Status\_Flags.*

[Remove **Clause 8.3.8**, p. 198]

### **~~8.3.8 — Change of Value Notification from a Loop Object Status\_Flags Property~~**

~~Purpose: To verify that the IUT can initiate UnconfirmedCOVNotification service requests conveying a change of the Status\_Flags property of a Loop object.~~

~~Test Steps: The steps for this test case are identical to the test steps in 8.2.8 except that the SubscribeCOV service request in step 1 shall have a value of FALSE for the 'Issue Confirmed Notifications' parameter, all of the ConfirmedCOVNotification requests shall be UnconfirmedCOVNotification requests, and there is no acknowledgment of the unconfirmed services. The MAC address used for the notification message shall be such that the TD is one of the recipients.~~

[Change and renumber **Clause 8.3.9**, p. 198]

[Reviewer Note: Added Process ID Requirement and Abort Conditionality to test.]

### **~~8.3.9~~ 8.3.5 Unsubscribed Change of Value Notifications**

Unsubscribed COV notifications differ from subscribed COV notifications that use the UnconfirmedCOVNotification service in two respects. First, no subscription is required. Second, the 'Subscriber Process Identifier' parameter usually has a value of zero.

~~BACnet Reference Clauses: 13.7.~~

Purpose: To verify that the IUT can initiate UnconfirmedCOVNotification service requests when no subscription for the COV notification has been made.

Test Concept: The IUT is configured to send unsubscribed COV notifications. The TD then waits for the notification. Given that there is no defined trigger, the vendor shall inform the tester how to make the IUT generate the notifications if they are not sent periodically.

Test Steps:

1. MAKE (the IUT send an unsubscribed COV notification)

2. *BEFORE Notification Fail Time*

RECEIVE UnconfirmedCOVNotification-Request,

'Subscriber Process Identifier' = 0,

'Initiating Device Identifier' = IUT,

'Monitored Object Identifier' = (any valid object identifier),

'Time Remaining' = 0,

'List of Values' = (any valid properties and values from the monitored object)

[Change and renumber **Clause 8.3.10**, p. 198]

[Reviewer Note: BTL-CR-0437 pointed out that test 8.3.10 does not work for devices that don't have a Local\_Time property and use a sequence number in Restart Notifications.]

### **8.3.10.3.6 Device Restart Notifications**

Purpose: To verify that the IUT initiates UnconfirmedCOVNotification service requests to each entry in its Restart\_Notification\_Recipients property when it resets.

Test Concept: The IUT is configured to send restart notifications and is then reset. The TD checks for the restart notifications.

Device restart notifications differ from subscribed COV notifications that use the UnconfirmedCOVNotification service in two respects. First, subscription is made through the Restart\_Notification\_Recipients property instead of SubscribeCOV. Second, the 'Subscriber Process Identifier' parameter always has a value of zero.

Configuration Requirements: For each Recipient of the Restart\_Notification\_Recipients property in the IUT which is of the device form, there shall be a device on the network that will answer Who-Is requests so that the IUT can determine addressing information before sending the restart notification.

Notes to tester: Not all IUTs can accurately differentiate between the types of restart reasons and thus no requirements are placed on the value returned in the restart notification(s) ~~step 4~~. The test shall pass regardless of the order in which the restart notifications are sent to the recipients. ~~IUT generates the UnconfirmedCOVNotification Requests in step 4. The value of T2 shall be the same for each notification sent out in step 4. If the Restart\_Notification\_Recipients list has multiple recipients, then the Time\_Of\_Device\_Restart value is expected to be the same in all notifications resulting from the same restart.~~

*Notes to Tester: Not all IUTs can accurately differentiate between the types of restart reasons, and thus, no requirements are placed on the value returned in the restart notification(s). The test shall pass regardless of the order in which the restart notifications are sent to the recipients. If the Restart\_Notification\_Recipients list has multiple recipients, then the Time\_Of\_Device\_Restart value is expected to be the same in all notifications resulting from the same restart.*

Test Steps:

```

1. IF (Restart_Notification_Recipients is writable) THEN
    WRITE(Restart_Notification_Recipients = any non-empty list of Recipients)
ELSE
    MAKE (Restart_Notification_Recipients contain any non-empty list of Recipients)
2. READ T1 = Local_Time
3.2. MAKE(the IUT reset)
4.3. REPEAT X = (each entry in the Restart_Notification_Recipients) DO {
    BEFORE Notification Fail Time
    RECEIVE UnconfirmedCOVNotification-Request,
        DESTINATION = X,
        'Subscriber Process Identifier' = 0,
        'Initiating Device Identifier' = IUT,
        'Monitored Object Identifier' = (the IUT Device Identifier),
        'Time Remaining' = 0,
        'List of Values' = (System_Status=OPERATIONAL,
            Time_Of_Device_Restart = (T2),
            Last_Restart_Reason=(any valid restart reason, R))
    }
5.4. VERIFY Time_Of_Device_Restart = T2
6. CHECK (T1 ~ T2)
7.5. VERIFY Last_Restart_Reason = R
6. IF (T2 is not a sequence number) THEN
    VERIFY Local_Time ~ T2
ELSE
    MAKE(the IUT reset)
    REPEAT X = (each entry in the Restart_Notification_Recipients) DO {
        BEFORE Notification Fail Time
        RECEIVE UnconfirmedCOVNotification-Request,
            DESTINATION = X,
            'Subscriber Process Identifier' = 0,
            'Initiating Device Identifier' = IUT,
            'Monitored Object Identifier' = (the IUT Device Identifier),
            'Time Remaining' = 0,
            'List of Values' = (System_Status=OPERATIONAL,
                Time_Of_Device_Restart = (T3),
                Last_Restart_Reason=(any valid restart reason, R))

        CHECK (T3 > T2)
    }

```

~~Note to tester: Not all IUTs can accurately differentiate between the types of restart reasons and thus no requirements are placed on the value returned in step 4. The test shall pass regardless of the order in which the IUT generates the UnconfirmedCOVNotification-Requests in step 4. The value of T2 shall be the same for each notification sent out in step 4.~~

[Add new **Clause 8.3.X1**, p. 199]

### 8.3.X1 COVU\_Recipients Notifications

Purpose: To verify that the IUT initiates UnconfirmedCOVNotification service requests to each entry in its COVU\_Recipients property based on COVU\_Period.

Test Concept: The IUT contains a Global Group object, O1, that is configured to periodically send UnconfirmedCOVNotification using COVU\_Period and COVU\_Recipients. The TD checks for these notifications.

Configuration Requirements: COVU\_Recipients property shall be non-empty and contain at least one device and one address based recipient. The COVU\_Period shall be non-zero.

Notes to Tester: The test shall pass regardless of the order in which the IUT generates the UnconfirmedCOVNotification-Requests in each step.

Test Steps:

```

1. REPEAT X = (each entry in the COVU_Recipients) DO {
    BEFORE COVU_Period + Notification Fail Time
    RECEIVE UnconfirmedCOVNotification-Request,
        DESTINATION = X,
        'Subscriber Process Identifier' = 0,
        'Initiating Device Identifier' = IUT,
        'Monitored Object Identifier' = O1,
        'Time Remaining' = 0,
        'List of Values' = (Member_Status_Flags,
                           Elements of Present_Value)
    IF (X is the first entry in the COVU_Recipients) THEN
        READ T1 = Local_Time
    }
2. READ T1 = Local_Time
3. REPEAT X = (each entry in the COVU_Recipients) DO {
    BEFORE COVU_Period + Notification Fail Time
    RECEIVE UnconfirmedCOVNotification-Request,
        DESTINATION = X,
        'Subscriber Process Identifier' = 0,
        'Initiating Device Identifier' = IUT,
        'Monitored Object Identifier' = O1,
        'Time Remaining' = 0,
        'List of Values' = (Member_Status_Flags,
                           Elements of Present_Value)
    IF (X is the first entry in the COVU_Recipients) THEN
        READ T2 = Local_Time
    }
4. CHECK (T2 - T1 == COVU_Period)

```

[Add new **Clause 8.3.X12**, p. 199]

### 8.3.X12 UnconfirmedCOVNotification Pulse Converter changing Present\_Value

Purpose: To verify the correct operation of COV in the Pulse Converter object. The Pulse Converter initiates periodic COV Notifications every COV\_Period, even when there are no changes in the object, in addition to the COV notifications that this object type generates due to changes in the Present\_Value property.

Test Concept: This test is the same as 8.2.X9 except that the SubscribeCOV service request in step 1 shall have a value of FALSE for the 'Issue Confirmed Notifications' parameter, all of the ConfirmedCOVNotification requests shall be UnconfirmedCOVNotification requests, and there is no BACnet-SimpleACK-PDU returned in acknowledgment of the unconfirmed services.

Notes to Tester: The IUT may initiate additional COVNotifications. The final COVNotification shall accurately reflect Present\_Value and Status\_Flags.

[Add new **Clause 8.3.X13**, p. 199]

### 8.3.X13 UnconfirmedCOVNotification Pulse Converter changing Status\_Flags

Purpose: To verify the correct operation of COV in the Pulse Converter object. The Pulse Converter initiates periodic COV Notifications every COV\_Period, even when there are no changes in the object, in addition to the COV notifications that this object type generates due to changes in the Status\_Flags property.

Test Concept: This test is the same as 8.2.X10 except that the SubscribeCOV service request in step 1 shall have a value of FALSE for the 'Issue Confirmed Notifications' parameter, all of the ConfirmedCOVNotification requests shall be UnconfirmedCOVNotification requests, and there is no BACnet-SimpleACK-PDU returned in acknowledgment of the unconfirmed services.

[Add new **Clause 8.3.X14**, p. 199]

### **8.3.X14 Change of Value Notification from an Access Door object Present\_Value, Status\_Flags and Door\_Alarm\_State property**

Purpose: To verify that the IUT can initiate UnconfirmedCOVNotification service requests conveying a change of the Present\_Value, Status\_Flag and Door\_Alarm\_State property of Access Door objects.

Test Steps: The steps for this test case are identical to the test steps in 8.2.X11 except that the SubscribeCOV service request in step 1 shall have a value of FALSE for the 'Issue Confirmed Notifications' parameter, all the ConfirmedCOVNotification requests shall be UnconfirmedCOVNotification requests, and there is no acknowledgment of the unconfirmed services.

[Add new **Clause 8.3.X15**, p. 199]

### **8.3.X15 Change of Value Notification from an Access Point object**

Purpose: To verify that the IUT can initiate UnconfirmedCOVNotification service requests conveying a change of the Status\_Flag and Access\_Event\_Time properties of Access Point objects.

Test Steps: The steps for this test case are identical to the test steps in 8.2.X12 except that the SubscribeCOV service request in step 1 shall have a value of FALSE for the 'Issue Confirmed Notifications' parameter, all the ConfirmedCOVNotification requests shall be UnconfirmedCOVNotification requests, and there is no acknowledgment of the unconfirmed services.

[Add new **Clause 8.3.X16**, p. 199]

### **8.3.X16 Change of Value Notification from a Credential Data Input object**

Purpose: To verify that the IUT can initiate UnconfirmedCOVNotification service requests conveying a change of the Status\_Flags and Update\_Time properties of Credential Data Input objects.

Test Steps: The steps for this test case are identical to the test steps in 8.2.X13 except that the SubscribeCOV service request in step 1 shall have a value of FALSE for the 'Issue Confirmed Notifications' parameter, all the ConfirmedCOVNotification requests shall be UnconfirmedCOVNotification requests, and there is no acknowledgment of the unconfirmed services.

[Add new **Clause 8.3.X17**, p. 199]

### **8.3.X17 Change of Value Notification of Staging Object Present\_Value Property**

Purpose: To verify that the IUT can initiate UnconfirmedCOVNotification service requests conveying a change of the Present\_Value property of Staging objects.

Test Steps: The steps for this test case are identical to the test steps in 8.2.X17 except that the SubscribeCOV service request in step 1 shall have a value of FALSE for the 'Issue Confirmed Notifications' parameter, all of the ConfirmedCOVNotification requests shall be UnconfirmedCOVNotification requests, and there is no acknowledgment of the unconfirmed services. The MAC address used for the notification message shall be such that the TD is one of the recipients.

[Add new **Clause 8.3.X18**, p. 199]

### **8.3.X18 Change of Value Notification of Staging Object Status\_Flags Property**

Purpose: To verify that the IUT can initiate UnconfirmedCOVNotification service requests conveying a change of the Status\_Flags property of Staging objects.

Test Steps: The steps for this test case are identical to the test steps in 8.2.X18 except that the SubscribeCOV service request in step 1 shall have a value of FALSE for the 'Issue Confirmed Notifications' parameter, all of the ConfirmedCOVNotification requests shall be UnconfirmedCOVNotification requests, and there is no acknowledgment of the unconfirmed services. The MAC address used for the notification message shall be such that the TD is one of the recipients.

[Add new **Clause 8.3.X19**, p. 199]

### **8.3.X19 Change of Value Notification of Staging Object Present\_Stage Property**

Purpose: To verify that the IUT can initiate UnconfirmedCOVNotification service requests conveying a change of the Present\_Stage property of Staging objects.

Test Steps: The steps for this test case are identical to the test steps in 8.2.X18 except that the SubscribeCOV service request in step 1 shall have a value of FALSE for the 'Issue Confirmed Notifications' parameter, all of the ConfirmedCOVNotification requests shall be UnconfirmedCOVNotification requests, and there is no acknowledgment of the unconfirmed services. The MAC address used for the notification message shall be such that the TD is one of the recipients.

## **8.4 ConfirmedEventNotification Service Initiation Tests**

[Change **Clause 8.4.4**, p. 205]

[Reviewer Note: Corrected the Status\_Flags values expected.]

### **8.4.4 COMMAND\_FAILURE Tests (ConfirmedEventNotification)**

Purpose: To verify the correct operation of the COMMAND\_FAILURE algorithm.

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

Configuration Requirements: The IUT shall be configured such that the Event\_Enable property has a value of TRUE for the TO-OFFNORMAL and TO-NORMAL transitions. The Issue\_Confirmed\_Notifications property shall have a value of TRUE. The event-generating object shall be in a NORMAL state at the start of the test. pFeedbackValue shall be decoupled from the input signal that is normally used to verify the output so that it can be independently manipulated.

*Notes to Tester: 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.*

Test Steps:

1. VERIFY pCurrentState = NORMAL
2. IF (Protocol\_Revision is present AND Protocol\_Revision >= 13) THEN  
    VERIFY Status\_Flags = (~~FALSE, FALSE, FALSE, FALSE~~)(FALSE, FALSE, ?, ?)
3. IF (pMonitoredValue is writable) THEN  
    WRITE Present\_Value = (a different value)  
ELSE  
    MAKE (Present\_Value take on a different value)
4. WAIT (pTimeDelay)
5. BEFORE **Notification Fail Time**  
    RECEIVE ConfirmedEventNotification-Request,  
    'Process Identifier' = (any valid process ID),  
    'Initiating Device Identifier' = IUT,

- 'Event Object Identifier' = (the object being tested),  
 'Time Stamp' = (any valid time stamp),  
 'Notification Class' = (the configured notification class),  
 'Priority' = (the value configured to correspond to a TO-OFFNORMAL transition),  
 'Event Type' = COMMAND\_FAILURE,  
 'Message Text' = (optional, any valid message text),  
 'Notify Type' = EVENT | ALARM,  
 'AckRequired' = TRUE | FALSE,  
 'From State' = NORMAL,  
 'To State' = OFFNORMAL,  
 'Event Values' = pMonitoredValue, pStatusFlags, pFeedbackValue
6. TRANSMIT BACnet-SimpleACK-PDU
  7. IF (Protocol\_Revision is present AND Protocol\_Revision >= 13) THEN  
     VERIFY pStatusFlags = (TRUE, FALSE, ?, ?)
  8. VERIFY pCurrentState = OFFNORMAL
  9. IF (Protocol\_Revision is present and Protocol\_Revision ≥ 1) THEN  
     VERIFY Event\_Time\_Stamps = (the timestamp in step 5, \*, \*)
  10. IF (pFeedbackValue is writable) THEN  
     WRITE pFeedbackValue = (a value consistent with pMonitoredValue)  
   ELSE  
     MAKE (pFeedbackValue take on a value consistent with pMonitoredValue)
  11. WAIT (pTimeDelay)
  12. **BEFORE Notification Fail Time**  
     RECEIVE ConfirmedEventNotification-Request,  
     'Process Identifier' = (any valid process ID),  
     'Initiating Device Identifier' = IUT,  
     'Event Object Identifier' = (the object being tested),  
     'Time Stamp' = (any valid time stamp),  
     'Notification Class' = (the configured notification class),  
     'Priority' = (the value configured to correspond to a TO-NORMAL transition),  
     'Event Type' = COMMAND\_FAILURE,  
     'Message Text' = (optional, any valid message text),  
     'Notify Type' = EVENT | ALARM,  
     'AckRequired' = TRUE | FALSE,  
     'From State' = OFFNORMAL,  
     'To State' = NORMAL,  
     'Event Values' = pMonitoredValue, pStatusFlags, pFeedbackValue
  13. TRANSMIT BACnet-SimpleACK-PDU
  14. IF (Protocol\_Revision is present AND Protocol\_Revision >= 13) THEN  
     VERIFY pStatusFlags = (FALSE, FALSE, ?, ?)
  15. VERIFY pCurrentState = NORMAL
  16. IF (Protocol\_Revision is present and Protocol\_Revision ≥ 1) THEN  
     VERIFY Event\_Time\_Stamps = (the timestamp in step 5, \*, the timestamp in step 12)

~~Notes to Tester: The 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.~~

#### 8.4.8 CHANGE\_OF\_LIFE\_SAFETY Tests (ConfirmedEventNotification)

[Change Clause 8.4.8.7, p. 220]

[Reviewer Note: Modify the test case as per BTL-CR-0477, when IUT does not support Mode changes which maintain the current Event\_State.]

##### 8.4.8.7 Mode Transition Tests when Event State is Maintained

Purpose: To verify the correct operation of the CHANGE\_OF\_LIFE\_SAFETY event algorithm for objects transitioning between the NORMAL, OFFNORMAL and LIFE\_SAFETY\_ALARM event states when a mode change occurs. Tests are conducted when a mode change occurs, but the event state does not change. Tests are also conducted when a mode change

occurs simultaneously with an event state change. In this latter case, the test verifies that the notification is immediate rather than waiting for the time delay.

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

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

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

Test Steps:

1. VERIFY Event\_Detection\_Enable = TRUE
2. CHECK (pCurrentState = NORMAL)
3. MAKE (pMonitoredValue have a value that corresponds to a NORMAL state)
4. *IF (IUT supports another pMode value which maintains the NORMAL state) THEN {*
- 4—MAKE (pMode = different value that maintains pCurrentState as NORMAL)
- 5—WAIT (pTimeDelayNormal)
- 6—**BEFORE Notification Fail Time**  
RECEIVE ConfirmedEventNotification-Request,  
'Process Identifier' = (any valid process ID),  
'Initiating Device Identifier' = IUT,  
'Event Object Identifier' = (the intrinsic reporting object being tested or the Event Enrollment object being tested),  
'Time Stamp' = (T1: any valid time stamp),  
'Notification Class' = (the configured notification class),  
'Priority' = (the value configured to correspond to a TO-NORMAL transition),  
'Event Type' = CHANGE\_OF\_LIFE\_SAFETY,  
'Message Text' = (S1: optional, any valid message text),  
'Notify Type' = EVENT | ALARM,  
'AckRequired' = TRUE | FALSE,  
'From State' = NORMAL,  
'To State' = NORMAL,  
'Event Values' = pMonitoredValue, pMode, pStatusFlags, pOperationExpected
- 7—TRANSMIT BACnet-SimpleACK-PDU
- 8 IF (Protocol\_Revision is present AND Protocol\_Revision ≥ 13) THEN  
VERIFY pStatusFlags = (FALSE, FALSE, ?, ?)
- 9 VERIFY pCurrentState = NORMAL
- 10 IF (Protocol\_Revision is present AND Protocol\_Revision ≥ 1) THEN  
VERIFY Event\_Time\_Stamps = (\*, \*, T1)
- 11 IF (Event\_Message\_Texts property exists) THEN  
VERIFY Event\_Message\_Texts = (\*, \*, S1)
- }*
5. ~~12~~MAKE (pMonitoredValue have a value that corresponds to an OFFNORMAL state)
6. VERIFY pCurrentState = OFFNORMAL
7. *IF (IUT supports another pMode value which maintains the OFFNORMAL state) THEN {*
- 13 MAKE (pMode = different value that maintains pCurrentState as OFFNORMAL)
- 14 WAIT (pTimeDelay)
- 15 **BEFORE Notification Fail Time**  
RECEIVE ConfirmedEventNotification-Request,  
'Process Identifier' = (any valid process ID),  
'Initiating Device Identifier' = IUT,  
'Event Object Identifier' = (the intrinsic reporting object being tested or the Event Enrollment object being tested),



```

    'Time Stamp' = (T2: any valid time stamp),
    'Notification Class' = (the configured notification class),
    'Priority' = (the value configured to correspond to a TO-OFFNORMAL transition),
    'Event Type' = CHANGE_OF_LIFE_SAFETY,
    'Message Text' = (S2: optional, any valid message text),
    'Notify Type' = EVENT | ALARM,
    'AckRequired' = TRUE | FALSE,
    'From State' = OFFNORMAL,
    'To State' = OFFNORMAL,
    'Event Values' = pMonitoredValue, pMode, pStatusFlags, pOperationExpected
16    TRANSMIT BACnet-SimpleACK-PDU
17    IF (Protocol_Revision is present AND Protocol_Revision ≥ 13) THEN
    VERIFY pStatusFlags = (TRUE, FALSE, ?, ?)
18-  VERIFY pCurrentState = OFFNORMAL
19-  IF (Protocol_Revision is present AND Protocol_Revision ≥ 1) THEN
    VERIFY Event_Time_Stamps = (T2, *, *)
20-  IF (Event_Message_Texts property exists) THEN
    VERIFY Event_Message_Texts = (S2, *, *)
}
8. 24-MAKE (pMonitoredValue have a value that corresponds to a LIFE_SAFETY_ALARM state)
9.  IF (IUT supports another pMode value which maintains the LIFE_SAFETY_ALARM state) THEN {
22-  MAKE (pMode = different value that maintains pCurrentState = LIFE_SAFETY_ALARM)
23-  WAIT (pTimeDelay)
24-  BEFORE Notification Fail Time
    RECEIVE ConfirmedEventNotification-Request,
    'Process Identifier' = (any valid process ID),
    'Initiating Device Identifier' = IUT,
    'Event Object Identifier' = (the intrinsic reporting object being tested or the Event Enrollment object being
tested),

    'Time Stamp' = (T3: any valid time stamp),
    'Notification Class' = (the configured notification class),
    'Priority' = (the value configured to correspond to a TO-NORMAL transition),
    'Event Type' = CHANGE_OF_LIFE_SAFETY,
    'Message Text' = (S3: optional, any valid message text),
    'Notify Type' = EVENT | ALARM,
    'AckRequired' = TRUE | FALSE,
    'From State' = OFFNORMAL,
    'To State' = OFFNORMAL,
    'Event Values' = pMonitoredValue, pMode, pStatusFlags, pOperationExpected
25-  TRANSMIT BACnet-SimpleACK-PDU
26-  IF (Protocol_Revision is present AND Protocol_Revision ≥ 13) THEN
    VERIFY pStatusFlags = (TRUE, FALSE, ?, ?)
27-  VERIFY pCurrentState = OFFNORMAL
28-  IF (Protocol_Revision is present AND Protocol_Revision ≥ 1) THEN
    VERIFY Event_Time_Stamps = (T3, *, *)
29-  IF (Event_Message_Texts property exists) THEN
    VERIFY Event_Message_Texts = (S3, *, *)
}

```

[Change **Clause 8.4.9**, p. 229]

[Reviewer Note: Added clarifying statement in Test Concept. Added Notes to Tester. Added step numbers back for clarity.]

#### 8.4.9 EXTENDED Test (ConfirmedEventNotification)

Purpose: To verify the correct generation EXTENDED event notifications.

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

Configuration Requirements: The IUT shall be configured such that the Event\_Enable property has a value of TRUE for whichever transition shall be used for the test. The 'Issue\_Confirmed\_Notifications' parameter shall have a value of TRUE. D1 and D2 are vendor specific delays (either of them or both may be zero).

Notes to Tester: The time stamps indicated by "\*" can have any valid value.

Test Steps:

1. IF (the object generates TO-OFFNORMAL transitions) THEN {
2.     READ CS1 = pCurrentState
3.     MAKE (an OFFNORMAL condition exist)
4.     WAIT D1
5.     **BEFORE Notification Fail Time**  
       RECEIVE ConfirmedEventNotification-Request,  
         'Process Identifier' =         (any valid process ID),  
         'Initiating Device Identifier' = IUT,  
         'Event Object Identifier' =    (the event generating object),  
         'Time Stamp' =                (TS1: the current local time),  
         'Notification Class' =        (the configured notification class),  
         'Priority' =                    (the value configured for TO\_OFFNORMAL),  
         'Event Type' =                EXTENDED,  
         'Message Text' =              (optional, any valid message text),  
         'Notify Type' =                EVENT | ALARM,  
         'AckRequired' =                TRUE | FALSE,  
         'From State' =                CS1,  
         'To State' =                   (CS2: any offnormal valid event state),  
         'Event Values' =              ( ( pVendorId: any valid vendor id),  
                                       (pEventType: any valid event-type),  
                                       (a list of 0 or more valid parameters as defined by the Vendor)  
                                       )  
       )
6.     TRANSMIT BACnet-SimpleACK-PDU
7.     IF (Protocol\_Revision is present AND Protocol\_Revision >= 13) THEN  
        VERIFY pStatusFlags = (TRUE, FALSE,?,?)
8.     VERIFY pCurrentState = CS2
9.     IF (Protocol\_Revision is present AND Protocol\_Revision >= 1) THEN  
        VERIFY Event\_Time\_Stamps = (TS1, \*, \*)
10. }  
 10. IF (the object generates TO\_NORMAL transitions) THEN
11.     READ CS1 = pCurrentState
12.     MAKE (a NORMAL condition exist)  
       WAIT D2
13.     **BEFORE Notification Fail Time**  
       RECEIVE ConfirmedEventNotification-Request,  
         'Process Identifier' =         (any valid process ID),  
         'Initiating Device Identifier' = IUT,  
         'Event Object Identifier' =    (the intrinsic reporting object being tested),  
         'Time Stamp' =                (TS2: the current local time),  
         'Notification Class' =        (the configured notification class),  
         'Priority' =                    (the value configured TO-NORMAL),  
         'Event Type' =                EXTENDED,  
         'Notify Type' =                EVENT | ALARM,  
         'Message Text' =              (optional, any valid message text),  
         'AckRequired' =                TRUE | FALSE,  
         'From State' =                CS1,  
         'To State' =                   NORMAL,  
         'Event Values' =              ( ( pVendorId: any valid vendor id),

```

                                (pEventType: any valid event-type),
                                (a list of 0 or more valid parameters as defined by the Vendor)
                            )
14.    TRANSMIT BACnet-SimpleACK-PDU
15. IF (Protocol_Revision is present AND Protocol_Revision >= 13) THEN
    VERIFY pStatusFlags = (FALSE, FALSE, ?, ?)
16. VERIFY pCurrentState = NORMAL
17. IF (Protocol_Revision is present AND Protocol_Revision >= 1) THEN
    VERIFY Event_Time_Stamps = (*, *, TS2)
}

```

#### 8.4.17 CHANGE\_OF\_RELIABILITY (ConfirmedEventNotification Tests)

[Change **Clause 8.4.17.1**, p. 238]

[Reviewer Note: Test name changed.]

##### 8.4.17.1 CHANGE\_OF\_RELIABILITY with *No Fault* the ~~NONE-fault~~ Algorithm (ConfirmedEventNotifications)

Purpose: To verify the correct operation of the None fault algorithm

Test Concept: The Test concept corresponds to 8.5.17.1

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

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

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

[Remove **Clause 8.4.17.7**, p. 239]

[Reviewer Note: Test 8.4.17.7 was replaced by tests 8.4.17.7.1, 8.4.17.7.2 and 8.4.17.7.3.]

##### ~~8.4.17.7 CHANGE\_OF\_RELIABILITY for Event Enrollment Fault Condition Precedence (ConfirmedEventNotifications)~~

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

~~Test Concept: The test concept corresponds to 8.5.17.7~~

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

~~Test Steps: The test steps for this test case are identical to the test steps in 8.5.17.7, except that the UnconfirmedEventNotification requests are ConfirmedEventNotification requests and the TD acknowledges receiving the notifications.~~

~~Notes to Tester: The passing results for this test case are identical to the ones in 8.5.17.7, except that the event notifications shall be conveyed using a ConfirmedEventNotification service request.~~

[Add new **Clause 8.4.17.7**, p. 239]

[Reviewer Note: This new Clause 8.4.17.7 and its subclauses 8.4.17.7.1, 8.4.17.7.2, and 8.4.17.7.3 replace the original Clause 8.4.17.7.]

#### **8.4.17.7 Event Enrollment Fault Condition Precedence Tests**

##### **8.4.17.7.1 Internal Faults Take Precedence Over Monitored Object Faults**

Purpose: To verify that the Event Enrollment object's fault detection gives precedence to internal unreliable operational faults over faults in the monitored object.

Test Concept: The test concept corresponds to 8.5.17.7.1

Configuration Requirements: The configuration requirements are identical to those in 8.5.17.7.1, except that the 'Issue Confirmed Notifications' parameter shall have a value of TRUE.

Test Steps: The test steps for this test case are identical to the test steps in 8.5.17.7.1, except that the UnconfirmedEventNotification requests are ConfirmedEventNotification requests, and the TD acknowledges receiving the notifications.

Notes to Tester: The passing results for this test case are identical to the ones in 8.5.17.7.1, except that the event notifications shall be conveyed using a ConfirmedEventNotification service request.

##### **8.4.17.7.2 Monitored Object Faults Take Precedence Over Fault Algorithms**

Purpose: To verify that the Event Enrollment object's fault detection gives precedence to faults in the monitored object over faults detected by fault algorithm.

Test Concept: The test concept corresponds to 8.5.17.7.2

Configuration Requirements: The configuration requirements are identical to those in 8.5.17.7.2, except that the 'Issue Confirmed Notifications' parameter shall have a value of TRUE.

Test Steps: The test steps for this test case are identical to the test steps in 8.5.17.7.2, except that the UnconfirmedEventNotification requests are ConfirmedEventNotification requests, and the TD acknowledges receiving the notifications.

Notes to Tester: The passing results for this test case are identical to the ones in 8.5.17.7.2, except that the event notifications shall be conveyed using a ConfirmedEventNotification service request.

##### **8.4.17.7.3 Internal Faults Take Precedence Over Fault Algorithms**

Purpose: To verify that the Event Enrollment object's fault detection gives precedence to internal unreliable operational faults over faults detected by fault algorithm.

Test Concept: The test concept corresponds to 8.5.17.7.3

Configuration Requirements: The configuration requirements are identical to those in 8.5.17.7.3, except that the 'Issue Confirmed Notifications' parameter shall have a value of TRUE.

Test Steps: The test steps for this test case are identical to the test steps in 8.5.17.7.3, except that the UnconfirmedEventNotification requests are ConfirmedEventNotification requests, and the TD acknowledges receiving the notifications.

Notes to Tester: The passing results for this test case are identical to the ones in 8.5.17.7.3, except that the event notifications shall be conveyed using a ConfirmedEventNotification service request.

[Add new **Clause 8.4.17.X1**, p. 240]

#### **8.4.17.X1 CHANGE\_OF\_RELIABILITY - FAULT\_LISTED Tests (ConfirmedEventNotification)**

##### **8.4.17.X1.1 NORMAL to FAULT Transition (ConfirmedEventNotification)**

Purpose: This test case verifies the correct operation of the FAULT\_LISTED event algorithm for objects transitioning from NORMAL to FAULT event states.

Test Concept: The test concept corresponds to 8.5.17.X1.1.

Configuration Requirements: The configuration requirements are identical to those in 8.5.17.X1.1, except that the 'Issue Confirmed Notifications' parameter shall have a value of TRUE.

Test Steps: The test steps for this test case are identical to the test steps in 8.5.17.X1.1, except that the UnconfirmedEventNotification requests are ConfirmedEventNotification requests, and the TD acknowledges receiving the notifications.

Notes to Tester: The passing results for this test case are identical to the ones in 8.5.17.X1.1, except that the event notifications shall be conveyed using a ConfirmedEventNotification service request.

##### **8.4.17.X1.2 FAULT-to-FAULT transition (ConfirmedEventNotification)**

Purpose: This test case verifies the correct operation of the FAULT\_LISTED event algorithm for objects transitioning from FAULT to FAULT event states.

Test Concept: The test concept corresponds to 8.5.17.X1.2.

Configuration Requirements: The configuration requirements are identical to those in 8.5.17.X1.2, except that the 'Issue Confirmed Notifications' parameter shall have a value of TRUE.

Test Steps: The test steps for this test case are identical to the test steps in 8.5.17.X1.2, except that the UnconfirmedEventNotification requests are ConfirmedEventNotification requests, and the TD acknowledges receiving the notifications.

Notes to Tester: The passing results for this test case are identical to the ones in 8.5.17.X1.2, except that the event notifications shall be conveyed using a ConfirmedEventNotification service request.

##### **8.4.17.X1.3 CHANGE\_OF\_RELIABILITY with the FAULT\_OUT\_OF\_RANGE Algorithm (ConfirmedEventNotification)**

Test Concept: Select a fault detecting object O1 which is configured to use the FAULT\_OUT\_OF\_RANGE algorithm. Ensure that no other fault conditions exist in the object. Set pMonitoredValue to outside the range of values considered to be normal for the object. Verify the correct transition is generated. The fault condition is then removed. It is verified that O1 generates the correct notifications.

Test Configuration: O1 is configured to detect and report faults, to have no fault conditions present. 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 pCurrentReliability = NO\_FAULT\_DETECTED
2. VERIFY pCurrentState = NORMAL
3. IF (pMonitoredValue is writable) THEN  
     WRITE pMonitoredValue = (a value less than pMinimumNormalValue | a value greater than pMaximumNormalValue)  
   ELSE  
     MAKE (pMonitoredValue = a value less than pMinimumNormalValue | a value greater than pMaximumNormalValue)
4. BEFORE **Notification Fail Time**,  
   RECEIVE ConfirmedEventNotification-Request,  
     'Process Identifier' = (any valid process identifier),  
     'Initiating Device Identifier' = IUT,  
     'Event Object Identifier' = O1,  
     'Time Stamp' = (any valid time stamp),  
     'Notification Class' = (the class corresponding to the object O1 being tested),  
     'Priority' = (the value configured to correspond to a TO\_FAULT transition),  
     'Event Type' = CHANGE\_OF\_RELIABILITY,  
     'Message Text' = (optional, any valid message text),  
     'Notify Type' = EVENT | ALARM,  
     'AckRequired' = TRUE | FALSE,  
     'From State' = NORMAL,  
     'To State' = FAULT,  
     'Event Values' = (pCurrentReliability, pStatusFlags, (A list of valid values for properties required to be reported for O1, and 0 or more other properties of O1))
5. TRANSMIT BACnet-SimpleACK-PDU
6. VERIFY pCurrentReliability = UNDER\_RANGE | OVER\_RANGE
7. VERIFY pCurrentState = FAULT
8. VERIFY pStatusFlags = (TRUE, TRUE,?,?)
9. IF (pMonitoredValue is writable) THEN  
     WRITE pMonitoredValue = (a value greater than or equal to pMinimumNormalValue, and pMonitoredValue is less than or equal to pMaximumNormalValue)  
   ELSE  
     MAKE (pMonitoredValue = a value, greater than or equal to pMinimumNormalValue, and pMonitoredValue is less than or equal to pMaximumNormalValue)
10. BEFORE **Notification Fail Time**,  
   RECEIVE ConfirmedEventNotification-Request,  
     'Process Identifier' = (any valid process identifier),  
     'Initiating Device Identifier' = IUT,  
     'Event Object Identifier' = O1,  
     'Time Stamp' = (any valid time stamp),  
     'Notification Class' = (the class corresponding to the object O1 being tested),  
     'Priority' = (the value configured to correspond to a TO\_FAULT transition),  
     'Event Type' = CHANGE\_OF\_RELIABILITY,  
     'Message Text' = (optional, any valid message text),  
     'Notify Type' = EVENT | ALARM  
     'AckRequired' = TRUE | FALSE,  
     'From State' = FAULT,  
     'To State' = NORMAL,  
     'Event Values' = (pCurrentReliability, pStatusFlags, (A list of valid values for properties required to be reported for O1, and 0 or more other properties of O1))
11. TRANSMIT BACnet-SimpleACK-PDU
12. VERIFY pCurrentReliability = NO\_FAULT\_DETECTED
13. VERIFY pCurrentState = NORMAL
14. VERIFY pStatusFlags = (FALSE, FALSE,?, ?)

[Add new **Clause 8.4.X10**, p. 241]

[Reviewer Note: Added testing for CHANGE\_OF\_DISCRETE\_VALUE algorithm.]

#### 8.4.X10 CHANGE\_OF\_DISCRETE\_VALUE Test (ConfirmedEventNotification)

Purpose: To verify correct operation of the CHANGE\_OF\_DISCRETE\_VALUE event algorithm. This test applies to Event Enrollment objects with an Event\_Type of CHANGE\_OF\_DISCRETE\_VALUE.

Test Concept: pMonitoredValue is changed to a value different from the initial value. After pTimeDelay (pTimeDelay is the value for Time\_Delay specified in the EventParameters), a TO-NORMAL transition occurs and a ConfirmedEventNotification is generated by the IUT.

Configuration Requirements: An Event\_Enrollment, EE1 is configured with an Object\_Property\_Reference, (O1, P1), such that P1 is of one of the following datatypes: BOOLEAN, Unsigned, Integer, Enumerated, CharacterString, Octet String, Date, Time, BACnetObjectIdentifier, or BACnetDateTime. Event\_Enable is configured with a value of (T,T,T) and Event\_Algorithm\_Inhibit = FALSE. The Event\_Parameters are configured with an Event Algorithm of CHANGE\_OF\_DISCRETE\_VALUE and a value for Time\_Delay that is within the allowable range for the IUT. The configured notification class is configured to send confirmed notifications to the TD. EE1 shall have an Event\_State of NORMAL at the start of the test.

Test Steps:

1. VERIFY pCurrentState = Normal
2. MAKE (the referenced property has a value x: x differs from the initial value)
3. WAIT (pTimeDelay)
4. BEFORE Notification Fail Time  
    RECEIVE ConfirmedEventNotification  
        'Process Identifier' = (any valid process ID),  
        'Initiating Device Identifier' = IUT,  
        'Event Object Identifier' = EE1,  
        'Time Stamp' = (the current local datetime or time or sequence number),  
        'Notification Class' = (the notification class configured for EE1),  
        'Priority' = (the value configured to correspond to TO-NORMAL),  
        'Event Type' = CHANGE\_OF\_DISCRETE\_VALUE,  
        'Message Text' = (optional, any valid message text),  
        'Notify Type' = ALARM | EVENT,  
        'Ack Required' = TRUE | FALSE,  
        'From State' = NORMAL,  
        'To State' = NORMAL,  
        'Event Values' = (x, Status\_Flags of O1)
5. TRANSMIT BACnet-SimpleAck-PDU
6. VERIFY pCurrentState = Normal

[Add new **Clause 8.4.X11**, p. 241]

[Reviewer Note: Added testing for ACCESS\_EVENT algorithm.]

#### 8.4.X11 ACCESS\_EVENT Test (ConfirmedEventNotification)

Purpose: To verify the correct operation of the ACCESS\_EVENT event algorithm.

Test Concept: The object, O1, begins the test in a NORMAL state. An access event, of a type listed in pAccessEvents is made to occur. It is verified that the IUT sends a confirmed notification of type ACCESS\_EVENT. A second access event, of a type not listed in pAccessEvents, is made to occur, if such is supported by the IUT. It is verified that no notification is generated. A third access event, of a type listed in pAccessEvents is made to occur. It is verified that the IUT sends a confirmed notification of type ACCESS\_EVENT.

Configuration Requirements: The IUT shall be configured such that the Event\_Enable property has a value of TRUE for TO-NORMAL transitions. The Issue\_Confirmed\_Notifications property shall have a value of TRUE. The event-generating object

ANSI/ASHRAE Addendum s to ANSI/ASHRAE Standard 135.1-2019 373

shall be in a NORMAL state at the start of the test. pAccessEvents shall be configured with at least 1 access event type that can be made to occur. If possible, at least access event type that can be made to occur shall not be included in pAccessEvents.

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 "\*" can have a value that indicates an unspecified time or a time that precedes the timestamp of the first received notification.

#### Test Steps:

-- Cause an access-event to occur that should be reported

1. VERIFY O1, Event\_State = NORMAL
2. MAKE(an access event occur which is listed in pAccessEvents)
3. BEFORE **Notification Fail Time**  
RECEIVE ConfirmedEventNotification-Request,  
'Process Identifier' = (the configured process ID),  
'Initiating Device Identifier' = IUT,  
'Event Object Identifier' = O1,  
'Time Stamp' = (Tnormal1: the current local time),  
'Notification Class' = (the configured notification class),  
'Priority' = (the value configured for a TO-NORMAL transition),  
'Event Type' = ACCESS\_EVENT,  
'Notify Type' = EVENT | ALARM,  
'AckRequired' = TRUE | FALSE,  
'From State' = NORMAL,  
'To State' = NORMAL,  
'Event Values' = { pMonitoredValue, pStatusFlags, pAccessEventTag,  
pAccessEventType, pAccessCredential  
-- and optionally pAuthenticationFactor  
}  
4. TRANSMIT BACnet-SimpleACK-PDU  
5. VERIFY O1, Status\_Flags = (FALSE, FALSE,?,?)  
6. VERIFY O1, Event\_State = NORMAL  
7. VERIFY O1, Event\_Time\_Stamps = (\*, \*, Tnormal1)

-- Cause an access-event to occur that should not be reported

8. IF (the IUT can detect access events which are not in pAccessEvents) THEN {  
MAKE(an access event occur which is not listed in pAccessEvents)  
CHECK(no notification is generated)  
}  
9. VERIFY O1, Status\_Flags = (FALSE, FALSE,?,?)  
10. VERIFY O1, Event\_State = NORMAL  
11. VERIFY O1, Event\_Time\_Stamps = (\*, \*, Tnormal1)

-- Cause an access-event to occur that should be reported

12. MAKE(an access event occur which is listed in pAccessEvents, and if possible different from the first access event )
13. BEFORE **Notification Fail Time**  
RECEIVE ConfirmedEventNotification-Request,  
'Process Identifier' = (the configured process ID),  
'Initiating Device Identifier' = IUT,  
'Event Object Identifier' = O1,  
'Time Stamp' = (Tnormal2: the current local time),  
'Notification Class' = (the configured notification class),  
'Priority' = (the value configured for a TO-NORMAL transition),  
'Event Type' = ACCESS\_EVENT,  
'Notify Type' = EVENT | ALARM,  
'AckRequired' = TRUE | FALSE,  
'From State' = NORMAL,  
'To State' = NORMAL,  
'Event Values' = { pMonitoredValue, pStatusFlags, pAccessEventTag,



```
pAccessEventType, pAccessCredential
-- and optionally pAuthenticationFactor
}
```

14. TRANSMIT BACnet-SimpleACK-PDU
15. VERIFY O1, Status\_Flags = (FALSE, FALSE,?,?)
16. VERIFY O1, Event\_State = NORMAL
17. VERIFY O1, Event\_Time\_Stamps = (\*, \*, Tnormal2)

[Add new **Clause 8.4.X18**, p. 241]

[Reviewer Note: Added testing for CHANGE\_OF\_TIMER algorithm.]

## 8.4.X18 CHANGE\_OF\_TIMER ConfirmedNotification Tests

### 8.4.X18.X1 CHANGE\_OF\_TIMER ConfirmedEventNotification Test

Purpose: To verify the correct operation of the CHANGE\_OF\_TIMER event algorithm. This test applies to objects that support an Event\_Type of CHANGE\_OF\_TIMER.

Test Concept: The object O1 begins the test in a NORMAL state. The pMonitoredValue is changed, typically by putting the Timer into operation where it conducts its usual operations, such that pMonitoredValue becomes equal to any of the values contained in pAlarmValues. After pTimeDelay time later in that same value, the object shall enter the OFFNORMAL state and transmit an event notification message. The pMonitoredValue is then changed such it is no longer equal to any of the values contained in pAlarmValues. After pTimeDelayNormal time later in that same value, the object shall enter the NORMAL state and transmit an event notification message.

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

Test Steps:

1. VERIFY Event\_State = NORMAL
2. MAKE (pMonitoredValue equal to any of the values contained in pAlarmValues)
3. WAIT (pTimeDelay)
4. BEFORE **Notification Fail Time**
  - RECEIVE ConfirmedEventNotification-Request,
  - 'Process Identifier' = (any valid process ID),
  - 'Initiating Device Identifier' = IUT,
  - 'Event Object Identifier' = O1,
  - 'Time Stamp' = (the current local datetime or time or sequence number),
  - 'Notification Class' = (the notification class configured for O1),
  - 'Priority' = (the value configured in transition),
  - 'Event Type' = CHANGE\_OF\_TIMER,
  - 'Message Text' = (optional, any valid message text),
  - 'Notify Type' = EVENT | ALARM,
  - 'AckRequired' = TRUE | FALSE,
  - 'From State' = NORMAL,
  - 'To State' = OFFNORMAL,
  - 'Event Values' = pMonitoredValue, pStatusFlags, pUpdateTime, (there are also three potential optional notification parameters: pLastStateChange, if one is available, pInitialTimeout, if one is available, pExpirationTime, if one is available)
5. TRANSMIT BACnet-SimpleACK-PDU
6. VERIFY Status\_Flags = {TRUE, FALSE, ?, ?}
7. VERIFY Event\_State = OFFNORMAL
8. MAKE (pMonitoredValue <> any of the values contained in pAlarmValues)
9. WAIT (pTimeDelayNormal)

## 10. BEFORE Notification Fail Time

RECEIVE ConfirmedEventNotification-Request,

'Process Identifier' = (any valid process ID),

'Initiating Device Identifier' = IUT,

'Event Object Identifier' = O1

'Time Stamp' = (the current local datetime or time or sequence number),

'Notification Class' = (the notification class configured for O1),

'Priority' = (the value configured in transition),

'Event Type' = CHANGE\_OF\_TIMER,

'Message Text' = (optional, any valid message text),

'Notify Type' = EVENT | ALARM,

'AckRequired' = TRUE | FALSE,

'From State' = OFFNORMAL,

'To State' = NORMAL,

'Event Values' = pMonitoredValue, pStatusFlags, pUpdateTime, (there are also three potential optional notification parameters: pLastStateChange, if one is available, pInitialTimeout, if one is available, pExpirationTime, if one is available)

11. TRANSMIT BACnet-SimpleACK-PDU

12. VERIFY Status\_Flags = {FALSE, FALSE, ?, ?}

13. VERIFY Event\_State = NORMAL

### 8.4.X18.X2 CHANGE\_OF\_TIMER Offnormal-to-Offnormal ConfirmedEventNotification

Purpose: To verify the correct operation of the CHANGE\_OF\_TIMER event algorithm when pUpdateTime changes while the object is already OFFNORMAL, and the new state is also one of the pAlarmValues. This test applies to objects that support an Event\_Type of CHANGE\_OF\_TIMER.

Test Concept: The object O1 begins the test in a NORMAL state. The pMonitoredValue is changed multiple times, typically by putting the Timer into operation where it conducts its usual operations, such that at multiple points the pMonitoredValue equal to any of the values contained in pAlarmValues. After the object enters the OFFNORMAL state and transmits a first event notification message, when pMonitoredValue changes but is again equal to any of the values contained in pAlarmValues, it is observed to transmit another event notification message. Finally, the pMonitoredValue becomes a value that is not equal to any of the values contained in pAlarmValues. After pTimeDelayNormal time holding at that value, the object shall enter the NORMAL state and transmit an event notification message.

Configuration Requirements: The O1 shall be configured such that the Event\_Enable property has a value of TRUE for the TO-OFFNORMAL and TO-NORMAL transitions. The 'Issue Confirmed Notifications' parameter in the Recipient\_List of the configured Notification Class shall have a value of TRUE. The Recipient\_List of the configured Notification Class shall contain recipients. The event-generating object shall be in a NORMAL state at the start of the test. If the pAlarmValues cannot be configured with two different values to which pMonitored will become equal, then this test shall be skipped.

Test Steps:

1. VERIFY Event\_State = NORMAL

2. MAKE (pMonitoredValue equal to any of the values contained in pAlarmValues)

3. WAIT (pTimeDelay)

4. BEFORE Notification Fail Time

RECEIVE ConfirmedEventNotification-Request,

'Process Identifier' = (any valid process ID),

'Initiating Device Identifier' = IUT,

'Event Object Identifier' = O1,

'Time Stamp' = (the current local datetime or time or sequence number),

'Notification Class' = (the notification class configured for O1),

'Priority' = (the value configured in transition),

'Event Type' = CHANGE\_OF\_TIMER,

'Message Text' = (optional, any valid message text),

'Notify Type' = EVENT | ALARM,

'AckRequired' = TRUE | FALSE,

'From State' = NORMAL,

'To State' = OFFNORMAL,

'Event Values' = pMonitoredValue, pStatusFlags, pUpdateTime, (there are also three potential optional notification parameters: pLastStateChange, if one is available, pInitialTimeout, if one is available, pExpirationTime, if one is available)

5. TRANSMIT BACnet-SimpleACK-PDU

6. VERIFY Status\_Flags = {TRUE, FALSE, ?, ?}

7. VERIFY Event\_State = OFFNORMAL

8. MAKE (pUpdateTime change, usually by pMonitoredValue becoming a different value, but ensure it is again equal to any of the values contained in pAlarmValues)

9. BEFORE **Notification Fail Time** -- Note: no pTimeDelay here

RECEIVE ConfirmedEventNotification-Request,

'Process Identifier' = (any valid process ID),

'Initiating Device Identifier' = IUT,

'Event Object Identifier' = O1,

'Time Stamp' = (the current local datetime or time or sequence number),

'Notification Class' = (the notification class configured for O1),

'Priority' = (the value configured in transition),

'Event Type' = CHANGE\_OF\_TIMER,

'Message Text' = (optional, any valid message text),

'Notify Type' = EVENT | ALARM,

'AckRequired' = TRUE | FALSE,

'From State' = OFFNORMAL,

'To State' = OFFNORMAL,

'Event Values' = pMonitoredValue, pStatusFlags, pUpdateTime, (there are also three potential optional notification parameters: pLastStateChange, if one is available, pInitialTimeout, if one is available, pExpirationTime, if one is available)

10. TRANSMIT BACnet-SimpleACK-PDU

11. VERIFY Status\_Flags = {TRUE, FALSE, ?, ?}

12. VERIFY Event\_State = OFFNORMAL

13. MAKE (pMonitoredValue <> any of the values contained in pAlarmValues)

14. WAIT (pTimeDelayNormal)

15. BEFORE **Notification Fail Time**

RECEIVE ConfirmedEventNotification-Request,

'Process Identifier' = (any valid process ID),

'Initiating Device Identifier' = IUT,

'Event Object Identifier' = O1

'Time Stamp' = (the current local datetime or time or sequence number),

'Notification Class' = (the notification class configured for O1),

'Priority' = (the value configured in transition),

'Event Type' = CHANGE\_OF\_TIMER,

'Message Text' = (optional, any valid message text),

'Notify Type' = EVENT | ALARM,

'AckRequired' = TRUE | FALSE,

'From State' = OFFNORMAL,

'To State' = NORMAL,

'Event Values' = pMonitoredValue, pStatusFlags, pUpdateTime, (there are also three potential optional notification parameters: pLastStateChange, if one is available, pInitialTimeout, if one is available, pExpirationTime, if one is available)

16. TRANSMIT BACnet-SimpleACK-PDU

17. VERIFY Status\_Flags = {FALSE, FALSE, ?, ?}

18. VERIFY Event\_State = NORMAL

## 8.5 UnconfirmedEventNotification Service Initiation Tests

[Change **Clause 8.5.17**, p. 247]

### 8.5.17 CHANGE\_OF\_RELIABILITY Tests (*UnconfirmedEventNotification Tests*)

[Change **Clause 8.5.17.1**, p. 247]

[Reviewer Note: Changed title to use the No Fault vs NON algorithm. Fixed Status\_Flags values in steps 4 and 8.]

#### 8.5.17.1 CHANGE\_OF\_RELIABILITY with ~~No NONE-fault~~ **Fault Algorithm (UnconfirmedEventNotifications)**

Purpose: ~~To verify the correct operation of the NONE-fault algorithm.~~ *To verify the correct operation of an object that supports first stage reliability evaluation and does not apply a standardized fault algorithm.*

Test Concept: Select an object, O1 that supports first stage reliability evaluation and does not apply a standardized fault algorithm. Ensure that no other fault conditions exist for the object. Create a fault condition. Verify the transition to fault is generated with Reliability set to R1. Remove the fault condition and verify the object transitions out of fault.

Test Configuration: O1 is configured to detect and report faults using unconfirmed event notifications. O1 is configured to have no fault conditions present and the Event\_State is NORMAL.

*Notes to Tester: The mechanism to enter the no fault state is a local matter.*

Test Steps:

1. VERIFY pCurrentReliability = NO\_FAULT\_DETECTED
2. VERIFY Event\_State = NORMAL
3. MAKE(O1 enter a fault condition)
4. BEFORE **Notification Fail Time**
  - RECEIVE UnconfirmedEventNotification-Request
    - 'Process Identifier' = (any valid process identifier),
    - 'Initiating Device Identifier' = IUT,
    - 'Event Object Identifier' = O1,
    - 'Time Stamp' = (the current local time or sequence number),
    - 'Notification Class' = (the notification class configured for O1),
    - 'Priority' = (the value configured for the transition),
    - 'Event Type' = CHANGE\_OF\_RELIABILITY,
    - 'Message Text' = (optional, any valid message text),
    - 'Notify Type' = ALARM | EVENT,
    - 'AckRequired' = TRUE | FALSE,
    - 'From State' = NORMAL,
    - 'To State' = FAULT,
    - 'Event Values' = (R1 any valid BACnetReliability, (?T, T, ?, ?),  
(A list of valid values for properties required to be reported for O1, and 0 or more other properties of O1))
5. VERIFY pCurrentReliability = R1
6. VERIFY Event\_State = FAULT
7. MAKE(O1 clear the fault condition)
8. BEFORE **Notification Fail Time**
  - RECEIVE UnconfirmedEventNotification-Request
    - 'Process Identifier' = (any valid process identifier),
    - 'Initiating Device Identifier' = IUT,
    - 'Event Object Identifier' = O1,
    - 'Time Stamp' = (the current local time or sequence number),
    - 'Notification Class' = (the notification class configured for O1),
    - 'Priority' = (the value configured for the transition),
    - 'Event Type' = CHANGE\_OF\_RELIABILITY,
    - 'Message Text' = (optional, any valid message text),
    - 'Notify Type' = ALARM | EVENT,
    - 'AckRequired' = TRUE | FALSE,

'From State' = FAULT,  
 'To State' = NORMAL,  
 'Event Values' = ( NO\_FAULT\_DETECTED,  
 (2F, F, ?, ?),  
 (A list of valid values for properties required to be reported  
 for O1, and 0 or more other properties of O1)  
 )

9. VERIFY pCurrentReliability = NO\_FAULT\_DETECTED

10. VERIFY Event\_State = NORMAL

~~Notes to Tester: The mechanism to enter the NONE fault algorithm is a local matter.~~

[Change **Clause 8.5.17.2**, p. 248]

[Reviewer Note: Added steps to verify the pCurrentState after each transition change.]

### **8.5.17.2 CHANGE\_OF\_RELIABILITY with the FAULT\_CHARACTERSTRING Algorithm (UnconfirmedEventNotifications)**

Purpose: To verify the correct operation of the FAULT\_CHARACTERSTRING fault algorithm.

Test Concept: Select a fault detecting object O1 which is configured to use the FAULT\_CHARACTERSTRING algorithm, and no other fault conditions exist for the object. pMonitoredValue is changed to a fault string and back to a non-fault string. It is verified that O1 generates the correct transitions.

Test Configuration: O1 is configured to detect and report faults, to have no fault conditions present, and to be in the NORMAL state. FVSET is the set of character strings defined as fault values for O1. ONVSET is the set of character strings defined as offnormal values for O1. FV1 contain a substring that exists in FVSET. If the empty string is included in the FVSET, then FV1 should be the empty string. NFV1 is a string value that does not contain substrings from FVSET or ONVSET. The 'Issue Confirmed Notifications' parameter shall have a value of FALSE.

*Notes to Tester: Note that a string is considered a substring of itself. Values required and allowed for O1 are described in Standard 135 as "Properties Reported in CHANGE\_OF\_RELIABILITY Notifications" (Table 13-5) along with supporting paragraphs.*

Test Steps:

1. VERIFY pCurrentReliability = NO\_FAULT\_DETECTED
2. VERIFY pCurrentState = NORMAL
3. IF (pMonitoredValue is writable) THEN  
     WRITE pMonitoredValue = FV1  
 ELSE  
     MAKE (pMonitoredValue = FV1)
4. BEFORE **Notification Fail Time**  
     RECEIVE UnconfirmedEventNotification-Request  
     'Process Identifier' = (any valid process identifier),  
     'Initiating Device Identifier' = IUT,  
     'Event Object Identifier' = O1,  
     'Time Stamp' = (any valid time stamp),  
     'Notification Class' = (the notification class configured for O1),  
     'Priority' = (the value configured for the transition),  
     'Event Type' = CHANGE\_OF\_RELIABILITY,  
     'Message Text' = (optional, any valid message text),  
     'Notify Type' = ALARM | EVENT,  
     'AckRequired' = TRUE | FALSE,  
     'From State' = NORMAL,  
     'To State' = FAULT,  
     'Event Values' = ( MULTI\_STATE\_FAULT,

- (T, T, ?, ?),  
(A list of valid values for properties required to be reported  
for O1, and 0 or more other properties of O1))
5. VERIFY pCurrentReliability = MULTI\_STATE\_FAULT
  6. *VERIFY pCurrentState = FAULT*
  7. IF (pMonitoredValue is writable) THEN  
    WRITE pMonitoredValue = NFV1  
ELSE  
    MAKE (pMonitoredValue = NFV1)
  8. BEFORE **Notification Fail Time**  
    RECEIVE UnconfirmedEventNotification-Request  
        'Process Identifier' = (any valid process identifier),  
        'Initiating Device Identifier' = IUT,  
        'Event Object Identifier' = O1,  
        'Time Stamp' = (any valid time stamp),  
        'Notification Class' = (the notification class configured for O1),  
        'Priority' = (the value configured for the transition),  
        'Event Type' = CHANGE\_OF\_RELIABILITY,  
        'Message Text' = (optional, any valid message text),  
        'Notify Type' = ALARM | EVENT,  
        'AckRequired' = TRUE | FALSE,  
        'From State' = FAULT,  
        'To State' = NORMAL,  
        'Event Values' = ( NO\_FAULT\_DETECTED,  
                          (F, F, ?, ?),  
                          (A list of valid values for properties required to be reported  
                          for O1, and 0 or more other properties of O1))
  9. VERIFY pCurrentReliability = NO\_FAULT\_DETECTED
  10. *VERIFY pCurrentState = NORMAL*

~~Notes to Tester: Note that a string is considered a substring of itself. Values required and allowed for O1 are described in standard 135 as "Properties Reported in CHANGE\_OF\_RELIABILITY Notifications" (Table 13-5 in 135-2016) along with supporting paragraphs.~~

[Change **Clause 8.5.17.3**, p. 249]

[Reviewer Note: Added missing verification of pCurrentState after each transition and fixed Status\_Flags value in step 8.]

### 8.5.17.3 CHANGE\_OF\_RELIABILITY with the FAULT\_EXTENDED Algorithm (UnconfirmedEventNotifications)

Purpose: To verify the correct operation of the FAULT\_EXTENDED fault algorithm.

Test Concept: Select a fault detecting object O1 which is configured to use the FAULT\_EXTENDED algorithm, and either pMonitoredValue is configured. Ensure that no other fault conditions exist for the object. In object O1, a condition is created that is detected as a fault by the FAULT\_EXTENDED algorithm configured. The fault condition is then removed. It is verified that O1 generates the correct notifications.

Test Configuration: O1 is configured to detect and report faults. O1 is configured to have no fault conditions present and has an Event\_State of NORMAL. The 'Issue Confirmed Notifications' parameter shall have a value of FALSE.

Test Steps:

1. VERIFY pCurrentReliability = NO\_FAULT\_DETECTED
2. VERIFY pCurrentState = NORMAL
3. MAKE (a fault condition exist)
4. BEFORE **Notification Fail Time**  
    RECEIVE UnconfirmedEventNotification-Request  
        'Process Identifier' = (any valid process identifier),  
        'Initiating Device Identifier' = IUT,

'Event Object Identifier' = O1,  
 'Time Stamp' = (any valid time stamp),  
 'Notification Class' = (the notification class configured for O1),  
 'Priority' = (the value configured for the transition),  
 'Event Type' = CHANGE\_OF\_RELIABILITY,  
 'Message Text' = (optional, any valid message text),  
 'Notify Type' = ALARM | EVENT,  
 'AckRequired' = TRUE | FALSE,  
 'From State' = NORMAL,  
 'To State' = FAULT,  
 'Event Values' = ( (R1: any valid reliability value),  
 (T, T, ?, ?),  
 (a vendor specified set of values))

5. VERIFY pCurrentReliability = R1
6. *VERIFY pCurrentState = FAULT*
7. MAKE (remove the fault condition)
8. BEFORE **Notification Fail Time**

RECEIVE UnconfirmedEventNotification-Request

'Process Identifier' = (any valid process identifier),  
 'Initiating Device Identifier' = IUT,  
 'Event Object Identifier' = O1,  
 'Time Stamp' = (any valid time stamp),  
 'Notification Class' = (the notification class configured for O1),  
 'Priority' = (the value configured for the transition),  
 'Event Type' = CHANGE\_OF\_RELIABILITY,  
 'Message Text' = (optional, any valid message text),  
 'Notify Type' = ALARM | EVENT,  
 'AckRequired' = TRUE | FALSE,  
 'From State' = FAULT,  
 'To State' = NORMAL,  
 'Event Values' = ( NO\_FAULT\_DETECTED,  
 (?F, F, ?, ?),  
 (a vendor specified set of values))

9. VERIFY pCurrentReliability = NO\_FAULT\_DETECTED
10. *VERIFY pCurrentState = NORMAL*

[Change **Clause 8.5.17.4**, p. 250]

[Reviewer Note: Added verification of pCurrentState after each transition.]

#### **8.5.17.4 CHANGE\_OF\_RELIABILITY with the FAULT\_LIFE\_SAFETY Algorithm (UnconfirmedEventNotifications)**

Purpose: To verify the correct operation of the FAULT\_LIFE\_SAFETY fault algorithm.

Test Concept: Select a fault detecting object O1 which is configured to use the FAULT\_LIFE\_SAFETY algorithm. Ensure that no other fault conditions exist in the object. Set pMonitoredValue to FV1, a value which indicates a FAULT\_LIFE\_SAFETY fault condition. Verify the correct transition is generated. The fault condition is removed by setting pMonitoredValue to NV1, a value which indicates NO\_FAULT\_DETECTED and verify the correct transition is generated.

Test Configuration: O1 is configured to detect faults and to report those using unconfirmed event notifications. O1 is initially configured to have no fault conditions present and has an Event\_State of NORMAL. FV1 is a value for pMonitoredValue which indicates a fault condition, and NV1 is a value for pMonitoredValue which does not indicate a fault condition. The 'Issue Confirmed Notification' parameter shall have a value of FALSE.

Test Steps:

1. VERIFY pCurrentReliability = NO\_FAULT\_DETECTED
2. VERIFY pCurrentState = NORMAL

3. IF (pMonitoredValue is writable) THEN  
     WRITE pMonitoredValue = FV1  
   ELSE  
     MAKE (pMonitoredValue = FV1)
4. BEFORE **Notification Fail Time**  
   RECEIVE UnconfirmedEventNotification-Request  
     'Process Identifier' = (any valid process identifier),  
     'Initiating Device Identifier' = IUT,  
     'Event Object Identifier' = O1,  
     'Time Stamp' = (any valid time stamp),  
     'Notification Class' = (the notification class configured for O1),  
     'Priority' = (the value configured for the transition),  
     'Event Type' = CHANGE\_OF\_RELIABILITY,  
     'Message Text' = (optional, any valid message text),  
     'Notify Type' = ALARM | EVENT,  
     'AckRequired' = TRUE | FALSE,  
     'From State' = NORMAL,  
     'To State' = FAULT,  
     'Event Values' = ( MULTI\_STATE\_FAULT,  
                           (T, T, ?, ?),  
                           (A list of valid values for properties required to be reported  
                           for O1, and 0 or more other properties of O1))
5. VERIFY pCurrentReliability = MULTI\_STATE\_FAULT
6. *VERIFY pCurrentState = FAULT*
7. IF (pMonitoredValue is writable) THEN  
     WRITE pMonitoredValue = NV1  
   ELSE  
     MAKE (pMonitoredValue = NV1)
8. BEFORE **Notification Fail Time**  
   RECEIVE UnconfirmedEventNotification-Request  
     'Process Identifier' = (any valid process identifier),  
     'Initiating Device Identifier' = IUT,  
     'Event Object Identifier' = O1,  
     'Time Stamp' = (any valid time stamp),  
     'Notification Class' = (the notification class configured for O1),  
     'Priority' = (the value configured for the transition),  
     'Event Type' = CHANGE\_OF\_RELIABILITY,  
     'Message Text' = (optional, any valid message text),  
     'Notify Type' = ALARM | EVENT,  
     'AckRequired' = TRUE | FALSE,  
     'From State' = FAULT,  
     'To State' = NORMAL,  
     'Event Values' = ( NO\_FAULT\_DETECTED,  
                           (F, F, ?, ?),  
                           (A list of valid values for properties required to be reported  
                           for O1, and 0 or more other properties of O1))
9. VERIFY pCurrentReliability = NO\_FAULT\_DETECTED
10. *VERIFY pCurrentState = NORMAL*

[Change **Clause 8.5.17.5**, p. 251]

[Reviewer Note: Added verification of pCurrentState after each transition.]

#### **8.5.17.5 CHANGE\_OF\_RELIABILITY with the FAULT\_STATE Algorithm (UnconfirmedEventNotifications)**

Purpose: To verify the correct operation of the FAULT\_STATE fault algorithm.

Test Concept: Select a fault detecting object O1 which is configured to use the FAULT\_STATE algorithm. Ensure that no other fault conditions exist in the object. Set pMonitoredValue to FV1, a value which indicates a FAULT\_STATE fault



condition. Verify the correct transition is generated. The fault condition is removed by setting pMonitoredValue to NV1, a value which indicates NO\_FAULT\_DETECTED and verify the correct transition is generated.

Test Configuration: O1 is configured to detect faults and to report those using unconfirmed event notifications. O1 is initially configured to have no fault conditions present, and an Event\_State of NORMAL. FV1 is a value for pMonitoredValue which indicates a fault condition, and NV1 is a value for pMonitoredValue which does not indicate a fault condition. The 'Issue Confirmed Notifications' parameter shall have a value of FALSE.

Test Steps:

1. VERIFY pCurrentReliability = NO\_FAULT\_DETECTED
2. VERIFY pCurrentState = NORMAL
3. IF (pMonitoredValue is writable) THEN  
    WRITE pMonitoredValue = FV1  
ELSE  
    MAKE (pMonitoredValue = FV1)
4. BEFORE **Notification Fail Time**  
    RECEIVE UnconfirmedEventNotification-Request  
    'Process Identifier' = (any valid process identifier),  
    'Initiating Device Identifier' = IUT,  
    'Event Object Identifier' = O1,  
    'Time Stamp' = (any valid time stamp),  
    'Notification Class' = (the notification class configured for O1),  
    'Priority' = (the value configured for the transition),  
    'Event Type' = CHANGE\_OF\_RELIABILITY,  
    'Message Text' = (optional, any valid message text),  
    'Notify Type' = ALARM | EVENT,  
    'AckRequired' = TRUE | FALSE,  
    'From State' = NORMAL,  
    'To State' = FAULT,  
    'Event Values' = ( MULTI\_STATE\_FAULT,  
                      (T, T, ?, ?),  
                      (A list of valid values for properties required to be reported  
                      for O1, and 0 or more other properties of O1))
5. VERIFY pCurrentReliability = MULTI\_STATE\_FAULT
6. VERIFY pCurrentState = FAULT
7. IF (pMonitoredValue is writable) THEN  
    WRITE pMonitoredValue = NV1  
ELSE  
    MAKE (pMonitoredValue = NV1)
8. BEFORE **Notification Fail Time**  
    RECEIVE UnconfirmedEventNotification-Request  
    'Process Identifier' = (any valid process identifier),  
    'Initiating Device Identifier' = IUT,  
    'Event Object Identifier' = O1,  
    'Time Stamp' = (any valid time stamp),  
    'Notification Class' = (the notification class configured for O1),  
    'Priority' = (the value configured for the transition),  
    'Event Type' = CHANGE\_OF\_RELIABILITY,  
    'Message Text' = (optional, any valid message text),  
    'Notify Type' = ALARM | EVENT,  
    'AckRequired' = TRUE | FALSE,  
    'From State' = FAULT,  
    'To State' = NORMAL,  
    'Event Values' = ( NO\_FAULT\_DETECTED,  
                      (F, F, ?, ?),  
                      (A list of valid values for properties required to be reported  
                      for O1, and 0 or more other properties of O1))
9. VERIFY pCurrentReliability = NO\_FAULT\_DETECTED
10. VERIFY pCurrentState = NORMAL

[Change **Clause 8.5.17.6**, p. 253]

[Reviewer Note: Added verification of pCurrentState after each transition.]

#### 8.5.17.6 CHANGE\_OF\_RELIABILITY with the FAULT\_STATUS\_FLAGS Algorithm (UnconfirmedEventNotifications)

Purpose: To verify the correct operation of the FAULT\_STATUS\_FLAGS fault algorithm.

Test Concept: Select a fault detecting object O1 which is configured to use the FAULT\_STATUS\_FLAGS algorithm. Ensure that no other fault conditions exist for the object. Set pMonitoredValue to FV1, a value which indicates a FAULT\_STATUS\_FLAGS fault condition. Verify the correct transition is generated. The fault condition is removed by setting pMonitoredValue to NV1, a value which indicates NO\_FAULT\_DETECTED and verify the correct transition is generated.

Test Configuration: O1 is configured to detect faults and to report those using unconfirmed event notifications. O1 is initially configured to have no fault conditions present, and Event\_State is NORMAL. FV1 is a value for pMonitoredValue which indicates a fault condition, and NV1 is a value for pMonitoredValue which does not indicate a fault condition. The 'Issue Confirmed Notifications' parameter shall have a value of FALSE.

Test Steps:

1. VERIFY pCurrentReliability = NO\_FAULT\_DETECTED
2. VERIFY pCurrentState = NORMAL
3. IF (pMonitoredValue is writable) THEN  
    WRITE pMonitoredValue = FV1  
ELSE  
    MAKE (pMonitoredValue = FV1)
4. BEFORE **Notification Fail Time**  
    RECEIVE UnconfirmedEventNotification-Request  
    'Process Identifier' = (any valid process identifier),  
    'Initiating Device Identifier' = IUT,  
    'Event Object Identifier' = O1,  
    'Time Stamp' = (any valid time stamp),  
    'Notification Class' = (the notification class configured for O1),  
    'Priority' = (the value configured for the transition),  
    'Event Type' = CHANGE\_OF\_RELIABILITY,  
    'Message Text' = (optional, any valid message text),  
    'Notify Type' = ALARM | EVENT,  
    'AckRequired' = TRUE | FALSE,  
    'From State' = NORMAL,  
    'To State' = FAULT,  
    'Event Values' = ( MEMBER\_FAULT,  
                      (T, T, ?, ?),  
                      (A list of valid values for properties required to be reported  
                      for O1, and 0 or more other properties of O1))
5. VERIFY pCurrentReliability = MEMBER\_FAULT
6. VERIFY pCurrentState = FAULT
7. IF (pMonitoredValue is writable) THEN  
    WRITE pMonitoredValue = NV1  
ELSE  
    MAKE (pMonitoredValue = NV1)
8. BEFORE **Notification Fail Time**  
    RECEIVE UnconfirmedEventNotification-Request  
    'Process Identifier' = (any valid process identifier),  
    'Initiating Device Identifier' = IUT,  
    'Event Object Identifier' = O1,  
    'Time Stamp' = (any valid time stamp),  
    'Notification Class' = (the notification class configured for O1),  
    'Priority' = (the value configured for the transition),

'Event Type' = CHANGE\_OF\_RELIABILITY,  
 'Message Text' = (optional, any valid message text),  
 'Notify Type' = ALARM | EVENT,  
 'AckRequired' = TRUE | FALSE,  
 'From State' = FAULT,  
 'To State' = NORMAL,  
 'Event Values' = ( NO\_FAULT\_DETECTED,  
 (F, F̄, ?, ?),  
 (A list of valid values for properties required to be reported  
 for O1, and 0 or more other properties of O1))

9. VERIFY pCurrentReliability = NO\_FAULT\_DETECTED

10. VERIFY pCurrentState = NORMAL

[Remove **Clause 8.5.17.7**, p. 254]

[Reviewer Note: Test 8.5.17.7 was replaced by tests 8.5.17.7.1, 8.5.17.7.2 and 8.5.17.7.3.]

### **~~8.5.17.7 CHANGE\_OF\_RELIABILITY for Event Enrollment Fault Condition Precedence (UnconfirmedEventNotifications)~~**

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

**Test Concept:** Select an Event Enrollment object, EE1, which monitors an object, O2, which can transition into FAULT. EE1 should be able to be put into a state where it has an internal fault (internal to the Event Enrollment object and unrelated to the Reliability of the monitored object). Starting with both objects in a NORMAL state, causes a condition, which results in a fault in O2. Verify that EE1 reports the fault. Make a condition exist that results in EE1 entering an internal fault. Verify that EE1 reports the new fault condition. Verify that a fault detectable by the fault algorithm does not generate an event. Clear EE1's the internal fault condition and verify that EE1 reports O2's fault. Clear the condition causing O2's fault and verify that EE1 reports fault algorithm event. Clear the condition causing the fault algorithm and verify the return to NORMAL event occurs.

**Configuration Requirements:** EE1 is configured to detect and report faults and contains a fault algorithm. EE1 and O2 are each initially configured to have no fault conditions present, and Event\_State is NORMAL. The 'Issue Confirmed Notifications' parameter shall have a value of FALSE.

**Test Steps:**

1. ~~VERIFY pCurrentState = NORMAL~~
2. ~~VERIFY pCurrentReliability = NO\_FAULT\_DETECTED~~
3. ~~MAKE (a condition exist which will cause O2 to detect a fault)~~
4. ~~BEFORE Notification Fail Time~~
  - ~~RECEIVE UnconfirmedEventNotification Request~~
  - ~~'Process Identifier' = (any valid process identifier),~~
  - ~~'Initiating Device Identifier' = IUT,~~
  - ~~'Event Object Identifier' = EE1,~~
  - ~~'Time Stamp' = (any valid time stamp),~~
  - ~~'Notification Class' = (the notification class configured for O1),~~
  - ~~'Priority' = (the value configured for the transition),~~
  - ~~'Event Type' = CHANGE\_OF\_RELIABILITY,~~
  - ~~'Message Text' = (optional, any valid message text),~~
  - ~~'Notify Type' = ALARM | EVENT,~~
  - ~~'AckRequired' = FALSE,~~
  - ~~'From State' = NORMAL,~~
  - ~~'To State' = FAULT,~~
  - ~~'Event Values' = (MONITORED\_OBJECT\_FAULT,~~
    - ~~(T, T, ?, ?),~~
    - ~~O2,~~
    - ~~(optional, the value of the monitored property),~~
    - ~~(optional, Reliability of O2),~~

~~\_\_\_\_\_ (optional, Status\_Flags of O2);~~  
~~\_\_\_\_\_ (0 or more other properties of O2))~~  
 5. VERIFY pCurrentReliability = MONITORED\_OBJECT\_FAULT  
 6. MAKE (a condition exist which will cause EE1 to transition into internal fault)  
 7. BEFORE **Notification Fail Time**  
     RECEIVE UnconfirmedEventNotification Request  
     'Process Identifier' = \_\_\_\_\_ (any valid process identifier);  
     'Initiating Device Identifier' = IUT;  
     'Event Object Identifier' = EE1;  
     'Time Stamp' = \_\_\_\_\_ (any valid time stamp);  
     'Notification Class' = \_\_\_\_\_ (the notification class configured for O1);  
     'Priority' = \_\_\_\_\_ (the value configured for the transition);  
     'Event Type' = CHANGE\_OF\_RELIABILITY;  
     'Message Text' = \_\_\_\_\_ (optional, any valid message text);  
     'Notify Type' = ALARM|EVENT;  
     'AckRequired' = FALSE;  
     'From State' = FAULT;  
     'To State' = FAULT;  
     'Event Values' = \_\_\_\_\_ ((R1: any value other than  
     \_\_\_\_\_ MONITORED\_OBJECT\_FAULT  
     \_\_\_\_\_ and NO\_FAULT\_DETECTED);  
     \_\_\_\_\_ (T, T, ?, ?);  
     \_\_\_\_\_ O2;  
     \_\_\_\_\_ (optional, the value of the monitored property);  
     \_\_\_\_\_ (optional, Reliability of O2);  
     \_\_\_\_\_ (optional, Status\_Flags of O2);  
     \_\_\_\_\_ (0 or more other properties of O2))  
 8. VERIFY pCurrentReliability = R1  
 9. MAKE (a condition that results in a fault detectable by the configured fault algorithm with a reliability value, R2,  
     different from R1)  
 10. CHECK (that the IUT does not send any notifications)  
 11. VERIFY pCurrentReliability = R1  
 12. MAKE (clear the condition that caused EE1 to enter into an internal fault)  
 11. BEFORE **Notification Fail Time**  
     RECEIVE UnconfirmedEventNotification Request  
     'Process Identifier' = \_\_\_\_\_ (any valid process identifier);  
     'Initiating Device Identifier' = IUT;  
     'Event Object Identifier' = EE1;  
     'Time Stamp' = \_\_\_\_\_ (any valid time stamp);  
     'Notification Class' = \_\_\_\_\_ (the notification class configured for O1);  
     'Priority' = \_\_\_\_\_ (the value configured for the transition);  
     'Event Type' = CHANGE\_OF\_RELIABILITY;  
     'Message Text' = \_\_\_\_\_ (optional, any valid message text);  
     'Notify Type' = ALARM|EVENT;  
     'AckRequired' = FALSE;  
     'From State' = FAULT;  
     'To State' = FAULT;  
     'Event Values' = \_\_\_\_\_ (MONITORED\_OBJECT\_FAULT;  
     \_\_\_\_\_ (T, T, ?, ?);  
     \_\_\_\_\_ O2;  
     \_\_\_\_\_ (optional, the value of the monitored property);  
     \_\_\_\_\_ (optional, Reliability of O2);  
     \_\_\_\_\_ (optional, Status\_Flags of O2)  
     \_\_\_\_\_ (0 or more other properties of O2))  
 12. VERIFY pCurrentReliability = MONITORED\_OBJECT\_FAULT  
 13. MAKE (clear the condition that caused O2 to enter into fault)  
 14. BEFORE **Notification Fail Time**  
     RECEIVE UnconfirmedEventNotification Request  
     'Process Identifier' = \_\_\_\_\_ (any valid process identifier);  
     'Initiating Device Identifier' = IUT;

~~'Event Object Identifier' = EE1,~~  
~~'Time Stamp' = (any valid time stamp),~~  
~~'Notification Class' = (the notification class configured for O1),~~  
~~'Priority' = (the value configured for the transition),~~  
~~'Event Type' = CHANGE\_OF\_RELIABILITY,~~  
~~'Message Text' = (optional, any valid message text),~~  
~~'Notify Type' = ALARM|EVENT,~~  
~~'AckRequired' = FALSE,~~  
~~'From State' = FAULT,~~  
~~'To State' = FAULT,~~  
~~'Event Values' = (R2,~~  
~~(T, T, ?, ?),~~  
~~O2,~~  
~~(optional, the value of the monitored property),~~  
~~NO\_FAULT\_DETECTED,~~  
~~(optional, Status\_Flags of O2),~~  
~~(0 or more other properties of O2))~~

~~15. VERIFY pCurrentReliability = R2~~

~~16. MAKE (clear the condition for the fault algorithm)~~

~~17. BEFORE Notification Fail Time~~

~~RECEIVE UnconfirmedEventNotification Request~~  
~~'Process Identifier' = (any valid process identifier),~~  
~~'Initiating Device Identifier' = IUT,~~  
~~'Event Object Identifier' = EE1,~~  
~~'Time Stamp' = (any valid time stamp),~~  
~~'Notification Class' = (the notification class configured for O1),~~  
~~'Priority' = (the value configured for the transition),~~  
~~'Event Type' = CHANGE\_OF\_RELIABILITY,~~  
~~'Message Text' = (optional, any valid message text),~~  
~~'Notify Type' = ALARM|EVENT,~~  
~~'AckRequired' = FALSE,~~  
~~'From State' = FAULT,~~  
~~'To State' = NORMAL,~~  
~~'Event Values' = (NO\_FAULT\_DETECTED,~~  
~~(F, F, ?, ?),~~  
~~O2,~~  
~~(the value of the monitored property),~~  
~~NO\_FAULT\_DETECTED,~~  
~~(optional, Status\_Flags of O2),~~  
~~(0 or more other properties of O2))~~

~~19. VERIFY pCurrentReliability = NO\_FAULT\_DETECTED~~

~~20. VERIFY pCurrentState = NORMAL~~

~~Notes to Tester: If O2 is located in the IUT, then the IUT shall know and report the property values of O2 in the CHANGE\_OF\_RELIABILITY notifications. If O2 is not located in the IUT, then more time must be allowed between making or clearing a fault condition in O2 and the IUT detecting the change in O2's Reliability (the Notification Fail Time allowance does not start until after the IUT has acquired the information from O2).~~

[Add new **Clause 8.5.17.7**, p. 254]

[Reviewer Note: This new Clause 8.5.17.7 and its subclauses 8.5.17.7.1, 8.5.17.7.2, and 8.5.17.7.3 replace the original Clause 8.5.17.7.]

### **8.5.17.7 Event Enrollment Fault Condition Precedence Tests**

#### **8.5.17.7.1 Internal Faults Take Precedence Over Monitored Object Faults**

Purpose: To verify that the Event Enrollment object's fault detection gives precedence to internal unreliable operational faults over faults in the monitored object.

Test Concept: Select an Event Enrollment object, EE1, which can detect internal faults and which monitors an object O1 that can detect faults. Test that an internal unreliable operational fault takes precedence over a monitored object fault.

Test Steps:

1. VERIFY pCurrentReliability = NO\_FAULT\_DETECTED
2. VERIFY pCurrentState = NORMAL
3. MAKE(a condition exist which will cause O1 to transition into fault)
4. VERIFY pCurrentReliability = MONITORED\_OBJECT\_FAULT
5. VERIFY pCurrentState = FAULT
6. MAKE(a condition exist which will cause EE1 to transition into internal fault R1)
7. VERIFY pCurrentReliability = R1
8. MAKE(clear the condition that caused EE1 to enter into an internal fault)
9. VERIFY pCurrentReliability = MONITORED\_OBJECT\_FAULT
10. MAKE(clear the condition that caused O1 to transition into fault)
11. VERIFY pCurrentReliability = NO\_FAULT\_DETECTED
12. VERIFY pCurrentState = NORMAL

#### **8.5.17.7.2 Monitored Object Faults Take Precedence Over Fault Algorithms**

Purpose: To verify that the Event Enrollment object's fault detection gives precedence to faults in the monitored object over faults detected by fault algorithm.

Test Concept: Select an Event Enrollment object, EE1, which applies a fault algorithm and which monitors an object O1 that can detect faults. Test that a monitored object fault takes precedence over a standard fault algorithm fault.

Test Steps:

1. VERIFY pCurrentReliability = NO\_FAULT\_DETECTED
2. VERIFY pCurrentState = NORMAL
3. MAKE(a condition that results in a fault due to a configured fault algorithm with a reliability value, R1)
4. VERIFY pCurrentReliability = R1
5. VERIFY pCurrentState = FAULT
6. MAKE(a condition exist which will cause O1 to transition into fault)
7. VERIFY pCurrentReliability = MONITORED\_OBJECT\_FAULT
8. MAKE(clear the condition that caused O1 to transition into fault)
9. VERIFY pCurrentReliability = R1
10. MAKE(clear the condition for the fault algorithm)
11. VERIFY pCurrentReliability = NO\_FAULT\_DETECTED
12. VERIFY pCurrentState = NORMAL

#### **8.5.17.7.3 Internal Faults Take Precedence Over Fault Algorithms**

Purpose: To verify that the Event Enrollment object's fault detection gives precedence to internal unreliable operational faults over faults detected by fault algorithm.

Test Concept: Select an Event Enrollment object, EE1, which can detect internal faults and which applies a fault algorithm. Test that an internal unreliable operational fault takes precedence over a standard fault algorithm fault.

#### Test Steps:

1. VERIFY pCurrentReliability = NO\_FAULT\_DETECTED
2. VERIFY pCurrentState = NORMAL
3. MAKE(a condition that results in a fault due to a configured fault algorithm with a reliability value, R1)
4. VERIFY pCurrentReliability = R1
5. VERIFY pCurrentState = FAULT
6. MAKE(a condition exist which will cause EE1 to transition into internal fault R2, different from R1)
7. VERIFY pCurrentReliability = R2
8. MAKE(clear the condition that caused EE1 to enter into an internal fault)
9. VERIFY pCurrentReliability = R1
10. MAKE(clear the condition for the fault algorithm)
11. VERIFY pCurrentReliability = NO\_FAULT\_DETECTED
12. VERIFY pCurrentState = NORMAL

[Change **Clause 8.5.17.8**, p. 256]

[Reviewer Note: Added verify of pCurrentState after each transition.]

#### **8.5.17.8 CHANGE\_OF\_RELIABILITY of Event Enrollment Object, Monitored Object Fault (UnconfirmedEventNotifications)**

Purpose: To verify the proper operation of the Event Enrollment object's fault detection when the monitored object enters the fault state.

Test Concept: Select an Event Enrollment object EE1 that monitors an object M1 that can transition into FAULT. Starting with both objects in a NORMAL state, cause a condition which results in a fault in M1. Verify EE1 reports the fault. Clear the condition and verify EE1 reports the return to NORMAL.

Test Configuration: EE1 is configured to process faults in M1 and to report those using unconfirmed event notifications. EE1 and M1 are each initially configured to have no fault conditions present, and Event\_State is NORMAL. The 'Issue Confirmed Notifications' parameter shall have a value of FALSE.

#### Test Steps:

1. VERIFY pCurrentReliability = NO\_FAULT\_DETECTED
2. VERIFY pCurrentState = NORMAL
3. MAKE (M1 enter any fault state)
4. BEFORE **Notification Fail Time**
  - RECEIVE UnconfirmedEventNotification-Request
 

'Process Identifier' =	(any valid process identifier),
'Initiating Device Identifier' =	IUT,
'Event Object Identifier' =	EE1,
'Time Stamp' =	(any valid time stamp),
'Notification Class' =	(the notification class configured for EE1),
'Priority' =	(the value configured for the transition),
'Event Type' =	CHANGE_OF_RELIABILITY,
'Message Text' =	(optional, any valid message text),
'Notify Type' =	ALARM   EVENT,
'AckRequired' =	TRUE   FALSE,
'From State' =	NORMAL,
'To State' =	FAULT,
'Event Values' =	( MONITORED_OBJECT_FAULT,
	(T, T, ?, ?),
	M1,
	(optional, property value of M1),
	(optional, M1 Status_Flags, (?, T, ?, ?)),
	(0 or more other properties of M1))

5. VERIFY pCurrentReliability = MONITORED\_OBJECT\_FAULT
6. VERIFY pCurrentState = FAULT
7. MAKE (M1 clear fault state)
8. BEFORE **Notification Fail Time**
  - RECEIVE UnconfirmedEventNotification-Request
    - 'Process Identifier' = (any valid process identifier),
    - 'Initiating Device Identifier' = IUT,
    - 'Event Object Identifier' = EE1,
    - 'Time Stamp' = (any valid time stamp),
    - 'Notification Class' = (the notification class configured for EE1),
    - 'Priority' = (the value configured for the transition),
    - 'Event Type' = CHANGE\_OF\_RELIABILITY,
    - 'Message Text' = (optional, any valid message text),
    - 'Notify Type' = ALARM | EVENT,
    - 'AckRequired' = TRUE | FALSE,
    - 'From State' = FAULT,
    - 'To State' = NORMAL,
    - 'Event Values' = ( NO\_FAULT\_DETECTED,
      - (F, F, ?, ?),
      - M1,
      - (optional, property value of M1),
      - (optional, M1 Status\_Flags, (? , F, ?, ?)),
      - (0 or more other properties of M1))
9. VERIFY pCurrentReliability = NO\_FAULT\_DETECTED
10. VERIFY pCurrentState = NORMAL

[Change **Clause 8.5.17.9**, p. 258]

[Reviewer Note: Added verify of pCurrentState after each transition.]

#### 8.5.17.9 CHANGE\_OF\_RELIABILITY of Event Enrollment Object Fault (UnconfirmedEventNotifications)

Purpose: To verify the Event Enrollment object generates a fault event when the object enters into fault due to an internal unreliable operation.

Test Concept: Select an Event Enrollment object EE1 that can be made to enter into fault due to an internal unreliable operation. Starting EE1 in a NORMAL state, cause a condition which results in an internal fault. Verify that EE1 reports the fault. Clear the condition and verify that EE1 reports the return to NORMAL.

Configuration Requirements: EE1 is configured to be able to enter a fault state and to report. EE1 is initially configured to have no fault conditions present, and Event\_State is NORMAL. The 'Issue Confirmed Notifications' parameter shall have a value of FALSE.

Test Steps:

1. VERIFY pCurrentReliability = NO\_FAULT\_DETECTED
2. VERIFY pCurrentState = NORMAL
3. MAKE (EE1 enter any internal fault state)
4. BEFORE **Notification Fail Time**
  - RECEIVE UnconfirmedEventNotification-Request
    - 'Process Identifier' = (any valid process identifier),
    - 'Initiating Device Identifier' = IUT,
    - 'Event Object Identifier' = EE1,
    - 'Time Stamp' = (any valid time stamp),
    - 'Notification Class' = (the notification class configured for EE1),
    - 'Priority' = (the value configured for the transition),
    - 'Event Type' = CHANGE\_OF\_RELIABILITY,
    - 'Message Text' = (optional, any valid message text),
    - 'Notify Type' = ALARM | EVENT,



'AckRequired' = TRUE | FALSE,  
 'From State' = NORMAL,  
 'To State' = FAULT,  
 'Event Values' = ( (R1: any value other than  
 MONITORED\_OBJECT\_FAULT  
 and NO\_FAULT\_DETECTED),  
 (T, T, ?, ?),  
 (M1, any valid monitored object),  
 (optional, property value of M1),  
 (optional, M1 Status\_Flags, (?, F, ?, ?)),  
 (0 or more other properties of M1)

5. VERIFY pCurrentReliability = R1

6. VERIFY pCurrentState = FAULT

7. MAKE (EE1 clear fault state)

8. BEFORE **Notification Fail Time**

RECEIVE UnconfirmedEventNotification-Request

'Process Identifier' = (any valid process identifier),  
 'Initiating Device Identifier' = IUT,  
 'Event Object Identifier' = EE1,  
 'Time Stamp' = (any valid time stamp),  
 'Notification Class' = (the notification class configured for EE1),  
 'Priority' = (the value configured for the transition),  
 'Event Type' = CHANGE\_OF\_RELIABILITY,  
 'Message Text' = (optional, any valid message text),  
 'Notify Type' = ALARM | EVENT,  
 'AckRequired' = TRUE | FALSE,  
 'From State' = FAULT,  
 'To State' = NORMAL,  
 'Event Values' = ( NO\_FAULT\_DETECTED,  
 (F, F, ?, ?),  
 M1,  
 (optional, property value of M1),  
 (optional, M1 Status\_Flags, (?, F, ?, ?)),  
 (0 or more other properties of M1))

9. VERIFY pCurrentReliability = NO\_FAULT\_DETECTED

10. VERIFY pCurrentState = NORMAL

[Change **Clause 8.5.17.10**, p. 259]

[Reviewer Note: Fixed the Status\_Flags in step 7. Added verify of pCurrentState after each transition.]

#### **8.5.17.10 After FAULT-to-NORMAL, Re-Notification of OFFNORMAL (UnconfirmedEventNotifications)**

Purpose: To verify that objects go to the NORMAL state after leaving the FAULT state, then transition to OFFNORMAL if the condition still exists.

Test Concept: Select a fault detecting object O1 which is able to detect OFFNORMAL conditions. Make O1 transition to an OFFNORMAL state and then transition to FAULT. Remove the condition causing the FAULT and verify O1 transitions from FAULT to NORMAL, then verify that the object transitions from NORMAL to the original OFFNORMAL state.

Configuration Requirements: O1 is configured to detect and report unconfirmed events and faults. O1 is configured to have no fault conditions present, and Event\_State is NORMAL. The 'Issue Confirmed Notifications' parameter shall have a value of FALSE.

Test Steps:

1. VERIFY pCurrentReliability = NO\_FAULT\_DETECTED
2. VERIFY pCurrentState = NORMAL

3. MAKE(O1 transition to an off normal state)
4. **BEFORE Notification Fail Time**
  - RECEIVE UnconfirmedEventNotification-Request
    - 'Process Identifier' = (any valid process identifier),
    - 'Initiating Device Identifier' = IUT,
    - 'Event Object Identifier' = O1,
    - 'Time Stamp' = (any valid time stamp),
    - 'Notification Class' = (the notification class configured for O1),
    - 'Priority' = (the value configured for the transition),
    - 'Event Type' = (ET1, any valid off normal event type),
    - 'Message Text' = (optional, any valid message text),
    - 'Notify Type' = ALARM | EVENT,
    - 'AckRequired' = TRUE | FALSE,
    - 'From State' = NORMAL,
    - 'To State' = OFFNORMAL,
    - 'Event Values' = (property-values appropriate for O1)
5. VERIFY pCurrentState = OFFNORMAL
6. MAKE(O1 enter a fault state)
7. **BEFORE Notification Fail Time**
  - RECEIVE UnconfirmedEventNotification-Request
    - 'Process Identifier' = (any valid process identifier),
    - 'Initiating Device Identifier' = IUT,
    - 'Event Object Identifier' = O1,
    - 'Time Stamp' = (the current local time or sequence number),
    - 'Notification Class' = (the notification class configured for O1),
    - 'Priority' = (the value configured for the transition),
    - 'Event Type' = CHANGE\_OF\_RELIABILITY,
    - 'Message Text' = (optional, any valid message text),
    - 'Notify Type' = ALARM | EVENT,
    - 'AckRequired' = TRUE | FALSE,
    - 'From State' = OFFNORMAL,
    - 'To State' = FAULT,
    - 'Event Values' = ( (R1 any valid BACnetReliability),  
(?T, T, ?, ?),  
(A list of valid values for properties required to be reported for O1, and 0 or more other properties of O1))
8. MAKE(O1 clear the fault condition)
9. **BEFORE Notification Fail Time**
  - RECEIVE UnconfirmedEventNotification-Request
    - 'Process Identifier' = (any valid process identifier),
    - 'Initiating Device Identifier' = IUT,
    - 'Event Object Identifier' = O1,
    - 'Time Stamp' = (any valid time stamp),
    - 'Notification Class' = (the notification class configured for O1),
    - 'Priority' = (the value configured for the transition),
    - 'Event Type' = CHANGE\_OF\_RELIABILITY,
    - 'Message Text' = (optional, any valid message text),
    - 'Notify Type' = ALARM | EVENT,
    - 'AckRequired' = TRUE | FALSE,
    - 'From State' = FAULT,
    - 'To State' = NORMAL,
    - 'Event Values' = ( NO\_FAULT\_DETECTED,  
(F, F, ?, ?),  
(A list of valid values for properties required to be reported for O1, and 0 or more other properties of O1))
10. VERIFY pCurrentReliability = NO\_FAULT\_DETECTED
11. *VERIFY pCurrentState = NORMAL*
12. **BEFORE Notification Fail Time**
  - RECEIVE UnconfirmedEventNotification-Request
    - 'Process Identifier' = (any valid process identifier),

'Initiating Device Identifier' = IUT,  
 'Event Object Identifier' = O1,  
 'Time Stamp' = (the current local time or sequence number),  
 'Notification Class' = (the notification class configured for O1),  
 'Priority' = (the value configured for the transition),  
 'Event Type' = ET1,  
 'Message Text' = (optional, any valid message text),  
 'Notify Type' = ALARM | EVENT,  
 'AckRequired' = TRUE | FALSE,  
 'From State' = NORMAL,  
 'To State' = OFFNORMAL,  
 'Event Values' = (property-values appropriate for O1)

11. *VERIFY pCurrentReliability = NO\_FAULT\_DETECTED*

12. *VERIFY pCurrentState = OFFNORMAL*

[Add new **Clause 8.5.17.X1**, p. 261]

## **8.5.17.X1 CHANGE\_OF\_RELIABILITY - FAULT\_LISTED Tests (UnconfirmedEventNotification)**

### **8.5.17.X1.1 NORMAL to FAULT Transition (UnconfirmedEventNotification)**

Purpose: This test case verifies the correct operation of the FAULT\_LISTED event algorithm for objects transitioning from NORMAL to FAULT event states.

Test Concept: Select a fault detecting object, O1, which is configured to use the FAULT\_LISTED algorithm. Ensure that no fault conditions exist in the object. Set pMonitoredList to FV1, a non-empty list of supported faults. Verify the correct transition is generated. The fault condition is removed by setting pMonitoredList to an empty list. Verify the correct transition is generated.

Configuration Requirements: O1 is configured to detect faults and to report those using unconfirmed event notifications. O1 is initially configured to have no fault conditions present and has an Event\_State of NORMAL.

Test Steps:

1. *VERIFY pCurrentReliability = NO\_FAULT\_DETECTED*
2. *VERIFY pCurrentState = NORMAL*
3. IF (pMonitoredList is writable) THEN  
     WRITE pMonitoredList = FV1  
 ELSE  
     MAKE (pMonitoredList = FV1)
4. BEFORE **Notification Fail Time**  
     RECEIVE UnconfirmedEventNotification-Request,  
     'Process Identifier' = (any valid process Identifier),  
     'Initiating Device Identifier' = IUT  
     'Event Object Identifier' = O1  
     'Time Stamp' = (any valid time stamp),  
     'Notification Class' = (the notification class configured for O1),  
     'Priority' = (the value configured for the transition),  
     'Event Type' = CHANGE\_OF\_RELIABILITY,  
     'Message Text' = (optional, any valid message text),  
     'Notify Type' = ALARM | EVENT,  
     'AckRequired' = TRUE | FALSE,  
     'From State' = NORMAL,  
     'To State' = FAULT,  
     'Event Values' = (FAULT\_LISTED,  
                             (T, T, ? ?),

- (A list of valid values for properties required to be reported  
for O1, and 0 or more other properties of O1))
5. VERIFY pCurrentReliability = FAULTS\_LISTED
  6. VERIFY pCurrentState = FAULT
  7. IF (pMonitoredList is writable) THEN  
    WRITE pMonitoredList = (an empty list)  
ELSE  
    MAKE (pMonitoredList = (an empty list))
  8. BEFORE **Notification Fail Time**  
    RECEIVE UnconfirmedEventNotification-Request,  
    'Process Identifier' = (any valid process Identifier),  
    'Initiating Device Identifier' = IUT  
    'Event Object Identifier' = O1  
    'Time Stamp' = (any valid time stamp),  
    'Notification Class' = (the notification class configured for O1),  
    'Priority' = (the value configured for the transition),  
    'Event Type' = CHANGE\_OF\_RELIABILITY,  
    'Message Text' = (optional, any valid message text),  
    'Notify Type' = ALARM | EVENT,  
    'AckRequired' = TRUE | FALSE,  
    'From State' = FAULT,  
    'To State' = NORMAL,  
    'Event Values' = (NO\_FAULT\_DETECTED,  
                      (F, F, ? ?),  
                      (A list of valid values for properties required to be reported  
                      for O1, and 0 or more other properties of O1))
  9. pCurrentReliability = NO\_FAULT\_DETECTED
  10. VERIFY pCurrentState = NORMAL

#### 8.5.17.X1.2 FAULT-to-FAULT transition (UnconfirmedEventNotification)

Purpose: This test case verifies the correct operation of the FAULT\_LISTED event algorithm for objects transitioning from FAULT to FAULT event states.

Test Concept: Select a fault detecting object, O1, which is configured to use the FAULT\_LISTED algorithm. Ensure that a fault condition, FV1, exists in the object. Set pMonitoredList to FV2, a non-empty list different from FV1. Verify the correct transition is generated.

Configuration Requirements: O1 is configured to detect faults and to report those using unconfirmed event notifications. O1 is initially configured to have a fault by having pMonitoredList contain a non-empty list, FV1, and has an Event\_State of FAULT.

Test Steps:

1. VERIFY pCurrentReliability = FAULT\_LISTED
2. VERIFY pCurrentState = FAULT
3. IF (pMonitoredList is writable) THEN  
    WRITE pMonitoredList = FV2  
ELSE  
    MAKE (pMonitoredList = FV2)
4. BEFORE **Notification Fail Time**  
    RECEIVE UnconfirmedEventNotification-Request,  
    'Process Identifier' = (any valid process Identifier),  
    'Initiating Device Identifier' = IUT  
    'Event Object Identifier' = O1  
    'Time Stamp' = (any valid time stamp),  
    'Notification Class' = (the notification class configured for O1),  
    'Priority' = (the value configured for the transition),  
    'Event Type' = CHANGE\_OF\_RELIABILITY,  
    'Message Text' = (optional, any valid message text),  
    'Notify Type' = ALARM | EVENT,

'AckRequired' = TRUE | FALSE,  
 'From State' = FAULT,  
 'To State' = FAULT,  
 'Event Values' = (FAULT\_LISTED,  
 (T, T, ? ?),  
 (A list of valid values for properties required to be reported  
 for O1, and 0 or more other properties of O1))

5. VERIFY pCurrentReliability = FAULTS\_LISTED
6. VERIFY pCurrentState = FAULT

### 8.5.17.X9.11 CHANGE\_OF\_RELIABILITY with First Stage Object Fault (UnconfirmedEventNotifications)

Purpose: To verify that fault conditions due to first stage faults are detected and reported.

Test Concept: An object in the IUT, O1, which can detect at least one first stage fault is selected. One of O1's detectable first stage faults, R1, is selected for the test. O1 begins the test in the NORMAL state with pCurrentReliability equal to NO\_FAULT\_DETECTED. The first stage fault condition, R1, is made to exist and it is verified that the pCurrentReliability changes to R1. It is verified that O1 generates the appropriate event notification. The fault condition is removed, and it is verified that the pCurrentReliability returns to NO\_FAULT\_DETECTED and the appropriate event notification message is generated.

Configuration Requirements: O1 is configured to detect faults and to report. O1 is initially configured to have no fault conditions present, and Event\_State is NORMAL. The 'Issue Confirmed Notifications' parameter shall have a value of FALSE.

Test Steps:

1. VERIFY pCurrentReliability = NO\_FAULT\_DETECTED
2. VERIFY pCurrentState = NORMAL
3. MAKE (pCurrentReliability = R1)
4. BEFORE **Notification Fail Time**  
 RECEIVE UnconfirmedEventNotification-Request,  
 'Process Identifier' = (any valid process ID),  
 'Initiating Device Identifier' = IUT,  
 'Event Object Identifier' = O1,  
 'Time Stamp' = (any valid time stamp),  
 'Notification Class' = (the notification class configured for O1),  
 'Priority' = (the value configured for the transition),  
 'Event Type' = CHANGE\_OF\_RELIABILITY,  
 'Message Text' = (optional, any valid message text),  
 'Notify Type' = EVENT | ALARM,  
 'AckRequired' = TRUE | FALSE,  
 'From State' = NORMAL,  
 'To State' = FAULT,  
 'Event Values' = ( R1,  
 (T, T, ?, ?),  
 (A list of valid values for properties required to be reported  
 for O1, and 0 or more other properties of O1))
5. VERIFY pCurrentReliability = R1
6. VERIFY pCurrentState = FAULT
7. MAKE (pCurrentReliability = NO\_FAULT\_DETECTED)
8. BEFORE **Notification Fail Time**  
 RECEIVE UnconfirmedEventNotification-Request,  
 'Process Identifier' = (any valid process ID),  
 'Initiating Device Identifier' = IUT,  
 'Event Object Identifier' = O1,  
 'Time Stamp' = (any current time stamp),  
 'Notification Class' = (the notification class configured for O1),  
 'Priority' = (the value configured for the transition),  
 'Event Type' = CHANGE\_OF\_RELIABILITY,

'Message Text' = (optional, any valid message text),  
 'Notify Type' = EVENT | ALARM,  
 'AckRequired' = TRUE | FALSE,  
 'From State' = FAULT,  
 'To State' = NORMAL,  
 'Event Values' = ( NO\_FAULT\_DETECTED,  
 (F, F̄, ?, ?),  
 (A list of valid values for properties required to be reported  
 for O1, and 0 or more other properties of O1))

9. VERIFY pCurrentReliability = NO\_FAULT\_DETECTED

10. VERIFY pCurrentState = NORMAL

### 8.5.17.X1.3 CHANGE\_OF\_RELIABILITY with the FAULT\_OUT\_OF\_RANGE Algorithm

#### (UnconfirmedEventNotification)

Purpose: To verify the correct operation of the FAULT\_OUT\_OF\_RANGE event algorithm.

Test Concept: Select a fault detecting object, O1, which is configured to use the FAULT\_OUT\_OF\_RANGE algorithm. Ensure that no other fault conditions exist in the object. Set pMonitoredValue to outside the range of values considered to be normal for the object. Verify the correct transition is generated. The fault condition is then removed. It is verified that O1 generates the correct notifications.

Test Configuration: O1 is configured to detect and report faults, to have no fault conditions present. 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: The test steps for this test case are identical to the test steps in 8.4.17.X1.3, except that the ConfirmedEventNotification requests are UnconfirmedEventNotification requests, and the TD does not acknowledge receiving the notifications.

Notes to Tester: The passing results for this test case are identical to the ones in 8.4.17.X1.3, except that the event notifications shall be conveyed using an UnconfirmedEventNotification service request. The MAC address used for these messages shall be either a broadcast that reaches the local network of the TD or the MAC address of the TD.

### 8.5.X10 CHANGE\_OF\_DISCRETE\_VALUE Test (UnconfirmedEventNotification)

Purpose: To verify correct operation of the CHANGE\_OF\_DISCRETE\_VALUE event algorithm. This test applies to Event Enrollment objects with an Event\_Type of CHANGE\_OF\_DISCRETE\_VALUE.

Test Concept: pMonitoredValue is changed to a value different from the initial value. After pTimeDelayNormal (pTimeDelay is the value for Time\_Delay specified in the EventParameters), a TO-NORMAL transition occurs and a UnconfirmedEventNotification is generated by the IUT.

Configuration Requirements: An Event\_Enrollment, EE1 is configured with an Object\_Property\_Reference, (O1, P1), such that P1 is of one of the following datatypes: BOOLEAN, Unsigned, Integer, Enumerated, CharacterString, Octet String, Date, Time, BACnetObjectIdentifier, or BACnetDateTime. Event\_Enable is configured with a value of (T,T,T) and Event\_Algorithm\_Inhibit = FALSE. The Event\_Parameters are configured with an Event Algorithm of CHANGE\_OF\_DISCRETE\_VALUE and a value for Time\_Delay that is within the allowable range for the IUT. The configured notification class is configured to send unconfirmed notifications to the TD. EE1 shall have an Event\_State of NORMAL at the start of the test.

Test Steps:

1. VERIFY pCurrentState = NORMAL
2. MAKE (the referenced property has a value x: x differs from the initial value)
3. WAIT (pTimeDelay)
4. BEFORE Notification Fail Time

#### RECEIVE UnconfirmedEventNotification

'Process Identifier' = (any valid process ID),  
 'Initiating Device Identifier' = IUT,  
 'Event Object Identifier' = EE1,  
 'Time Stamp' = (any valid time stamp),  
 'Notification Class' = (the notification class configured for EE1),  
 'Priority' = (the value configured to correspond to TO-NORMAL),  
 'Event Type' = CHANGE\_OF\_DISCRETE\_VALUE,  
 'Message Text' = (optional, any valid message text),  
 'Notify Type' = ALARM | EVENT,  
 'Ack Required' = TRUE | FALSE,  
 'From State' = NORMAL,  
 'To State' = NORMAL,  
 'Event Values' = (x, Status\_Flags of O1)

5. VERIFY pCurrentState = NORMAL

[Add new **Clause 8.5.X11**, p. 261]

[Reviewer Note: Added testing for ACCESS\_EVENT algorithm.]

#### **8.5.X11 ACCESS\_EVENT Test (UnconfirmedEventNotification)**

Purpose: To verify the correct operation of the ACCESS\_EVENT event algorithm.

Test Concept: The object, O1, begins the test in a NORMAL state. An access event, of a type listed in pAccessEvents is made to occur. It is verified that the IUT sends a confirmed notification of type ACCESS\_EVENT. A second access event, of a type not listed in pAccessEvents, is made to occur, if such is supported by the IUT. It is verified that no notification is generated. A third access event, of a type listed in pAccessEvents is made to occur. It is verified that the IUT sends a confirmed notification of type ACCESS\_EVENT.

Configuration Requirements: The IUT shall be configured such that the Event\_Enable property has a value of TRUE for 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. pAccessEvents shall be configured with at least 1 access event type that can be made to occur. If possible, at least access event type that can be made to occur shall not be included in pAccessEvents.

Notes to Tester: The passing results for this test case are identical to the ones in 8.4.X11 except that the event notifications shall be conveyed using an UnconfirmedEventNotification service request.

Test Steps: The test steps for this test case are identical to the test steps in 8.4.X11 except that the event notification requests are UnconfirmedEventNotification requests, and the TD does not acknowledge receiving the notifications.

[Add new **Clause 8.5.X18**, p. 261]

[Reviewer Note: Add testing for CHANGE\_OF\_TIMER algorithm.]

#### **8.5.X18 CHANGE\_OF\_TIMER Tests**

##### **8.5.X18.X1 CHANGE\_OF\_TIMER UnconfirmedEventNotification Test**

Purpose: To verify the correct operation of the CHANGE\_OF\_TIMER event algorithm. This test applies to objects that support an Event\_Type of CHANGE\_OF\_TIMER.

Configuration Requirements: The O1 shall be configured such that the Event\_Enable property has a value of TRUE for the TO-OFFNORMAL and TO-NORMAL transitions. The 'Issue Confirmed Notifications' parameter in the Recipient\_List of the

configured Notification Class shall have a value of FALSE. The event-generating object shall be in a NORMAL state at the start of the test.

**Test Steps:** The test steps for this test case are identical to the test steps in 8.4.10.X11, except that the ConfirmedEventNotification requests are UnconfirmedEventNotification requests, and the TD does not acknowledge receiving the notifications.

**Notes to Tester:** The passing results for this test case are identical to the ones in 8.4.10.X11, except that the event notifications shall be conveyed using an UnconfirmedEventNotification service request. The MAC address used for these messages shall be either a broadcast that reaches the local network of the TD or the MAC address of the TD.

### **8.5.X18.X2 CHANGE\_OF\_TIMER Offnormal-to-Offnormal UnconfirmedEventNotification Test**

**Purpose:** To verify the correct operation of the CHANGE\_OF\_TIMER event algorithm. This test applies to objects that support an Event\_Type of CHANGE\_OF\_TIMER.

**Test Concept:** The object O1 begins the test in a NORMAL state. The pMonitoredValue is changed multiple times, typically by putting the Timer into operation where it conducts its usual operations, such that at multiple points the pMonitoredValue is equal to any of the values contained in pAlarmValues. After the object enters the OFFNORMAL state and transmits a first event notification message, when pMonitoredValue changes but is again equal to any of the values contained in pAlarmValues, it is observed to transmit another event notification message. Finally, the pMonitoredValue becomes a value that is not equal to any of the values contained in pAlarmValues. After pTimeDelayNormal time holding at that value, the object shall enter the NORMAL state and transmit an event notification message.

**Configuration Requirements:** The O1 shall be configured such that the Event\_Enable property has a value of TRUE for the TO-OFFNORMAL and TO-NORMAL transitions. The 'Issue Confirmed Notifications' parameter in the Recipient\_List of the configured Notification Class shall have a value of FALSE. The Recipient\_List of the configured Notification Class shall contain recipients. The event-generating object shall be in a NORMAL state at the start of the test. If the pAlarmValues cannot be configured with two different values to which pMonitored will become equal, then this test shall be skipped.

**Test Steps:** The test steps for this test case are identical to the test steps in 8.4.10.X2 except that the ConfirmedEventNotification requests are UnconfirmedEventNotification requests, and the TD does not acknowledge receiving the notifications.

**Notes to Tester:** The passing results for this test case are identical to the ones in 8.4.10.X2, except that the event notifications shall be conveyed using an UnconfirmedEventNotification service request. The MAC address used for these messages shall be either a broadcast that reaches the local network of the TD or the MAC address of the TD.

## **8.11 SubscribeCOVProperty Service Initiation Tests**

[Change **Clause 8.11.1**, p. 267]

[Reviewer Note: Added test concept and variables to simplify test.]

### **8.11.1 Confirmed Notifications Subscription**

**Purpose:** To verify that the IUT can initiate a SubscribeCOVProperty service request for confirmed notifications *to any valid object, X*.

**Test Concept:** *A subscription for COV notifications is established, using a Lifetime of L. L shall be set to a value less than 24 hours and large enough to complete the test.*

**Test Steps:**

1. MAKE (the IUT send a SubscribeCOVProperty-Request),
2. RECEIVE SubscribeCOVProperty-Request  
'Subscriber Process Identifier' = (any valid process identifier),



'Monitored Object Identifier' = ~~(any valid object identifier)~~X  
 'Issue Confirmed Notifications' = TRUE,  
 'Lifetime' = ~~(any non-zero value)~~L,  
 'Monitored Property Identifier' = ~~(any valid property identifier)~~(the property Y to be monitored),  
 'COV Increment' = ~~(any valid value)~~any REAL value -- optional)

3. TRANSMIT BACnet-SimpleACK-PDU

[Change **Clause 8.11.2**, p. 267]

[Reviewer Note: Added test concept and variables to simplify test.]

### 8.11.2 Unconfirmed Notifications Subscription

Purpose: To verify that the IUT can initiate a SubscribeCOVProperty service request for unconfirmed notifications *to any valid object, X*.

*Test Concept: A subscription for COV notifications is established, using a Lifetime of L. L shall be set to a value less than 24 hours and large enough to complete the test.*

Test Steps:

1. MAKE (the IUT send a SubscribeCOVProperty-Request),
2. RECEIVE SubscribeCOVProperty-Request
 

'Subscriber Process Identifier' = (any valid process identifier),  
 'Monitored Object Identifier' = ~~(any valid object identifier)~~X  
 'Issue Confirmed Notifications' = FALSE,  
 'Lifetime' = ~~(any non-zero value)~~L,  
 'Monitored Property Identifier' = (any valid property identifier)(the property Y to be monitored),  
 'COV Increment' = ~~(any valid value)~~any REAL value -- optional)
3. TRANSMIT BACnet-SimpleACK-PDU

[Change **Clause 8.11.3**, p. 268]

[Reviewer Note: Added test concept and variables to simplify test.]

### 8.11.3 Canceling a Subscription

Purpose: To verify that the IUT can initiate a SubscribeCOVProperty service request to cancel a subscription *to any valid object, X*.

Test Steps:

1. MAKE (the IUT send a SubscribeCOVProperty-Request),
2. RECEIVE SubscribeCOVProperty-Request,
 

'Subscriber Process Identifier' = (any valid process identifier),  
 'Monitored Object Identifier' = ~~(any valid object identifier)~~X  
 'Monitored Property Identifier' = ~~(any valid property identifier)~~(the property Y to be monitored),  
 'COV Increment' = ~~(any valid value)~~any REAL value -- optional)
3. TRANSMIT BACnet-SimpleACK-PDU

[Add new **Clause 8.11.X1**, p. 268]

[Reviewer Note: Added new tests to support DS-COVP-A testing.]

## 8.11.X1 Change of Value Notification Tests

### 8.11.X1.1 Change of Value Notification

Purpose: To verify that the IUT can execute COVNotification requests from object types that provides a Property and Status\_Flags properties in COV notifications.

Test Concept: A subscription for COV notifications is established, using a Lifetime of L. L shall be set to a value less than 24 hours and large enough to complete the test.

Test Steps:

1. MAKE (the IUT send a SubscribeCOVProperty-Request),
2. RECEIVE SubscribeCOVProperty-Request
  - 'Subscriber Process Identifier' = (any valid process identifier),
  - 'Monitored Object Identifier' = X
  - 'Issue Confirmed Notifications' = TRUE | FALSE,
  - 'Lifetime' = L,
  - 'Monitored Property Identifier' = (the property Y to be monitored),
  - 'COV Increment' = (Any REAL value -- optional)
3. TRANSMIT BACnet-SimpleACK-PDU
4. BEFORE Notification Fail Time
  - IF (the subscription was for confirmed notifications) THEN
    - TRANSMIT ConfirmedCOVNotification-Request,
      - 'Subscriber Process Identifier' = (the process identifier used in step 1),
      - 'Initiating Device Identifier' = TD,
      - 'Monitored Object Identifier' = X
      - 'Time Remaining' = (any value appropriate for the Lifetime selected),
      - 'List of Values' = (values appropriate to the property Y subscribed to, and any other properties the IUT provides with it, such as Status\_Flags)
    - RECEIVE BACnet-SimpleACK-PDU
  - ELSE
    - TRANSMIT UnconfirmedCOVNotification-Request,
      - 'Subscriber Process Identifier' = (the process identifier used in step 1),
      - 'Initiating Device Identifier' = TD,
      - 'Monitored Object Identifier' = X
      - 'Time Remaining' = (any value appropriate for the Lifetime selected),
      - 'List of Values' = (values appropriate to the property Y subscribed to, and any other properties the IUT provides with it, such as Status\_Flags)
5. CHECK (to ensure that any appropriate functions defined by the manufacturer, such as displaying information on a workstation screen are carried out)

### 8.11.X1.2 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 Concept: A subscription for COV notifications is established, using a Lifetime of L. L shall be set to a value less than 24 hours and large enough to complete the test.

Test Steps:

1. MAKE (the IUT send a SubscribeCOVProperty-Request),
2. RECEIVE SubscribeCOVProperty-Request
  - 'Subscriber Process Identifier' = (any valid process identifier),
  - 'Monitored Object Identifier' = X
  - 'Issue Confirmed Notifications' = TRUE,
  - 'Lifetime' = L,
  - 'Monitored Property Identifier' = (the property Y to be monitored),

- 'COV Increment' = (Any REAL value -- optional)
3. TRANSMIT BACnet-SimpleACK-PDU
  4. TRANSMIT ConfirmedCOVNotification-Request,
    - 'Subscriber Process Identifier' = (a process identifier different from the one used in step 1),
    - 'Initiating Device Identifier' = TD,
    - 'Monitored Object Identifier' = X
    - 'Time Remaining' = (any value appropriate for the Lifetime selected),
    - 'List of Values' = (values appropriate to the property Y subscribed to, and any other properties the IUT provides with it, such as Status\_Flags)
  5. IF (Protocol\_Revision is present and Protocol\_Revision  $\geq$  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)

### 8.11.X1.3 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 Concept: A subscription for COV notifications is established and then cancelled or allowed to expire. A ConfirmedCOVNotification is then sent to the IUT to verify it returns the appropriate error or a Simple-Ack.

Test Steps:

1. MAKE (the IUT send a SubscribeCOVProperty-Request),
2. RECEIVE SubscribeCOVProperty-Request
  - 'Subscriber Process Identifier' = (any valid process identifier),
  - 'Monitored Object Identifier' = X
  - 'Issue Confirmed Notifications' = TRUE,
  - 'Lifetime' = L,
  - 'Monitored Property Identifier' = (the property Y to be monitored),
  - 'COV Increment' = (Any REAL value -- optional)
3. TRANSMIT BACnet-SimpleACK-PDU
4. BEFORE Notification Fail Time
  - TRANSMIT ConfirmedCOVNotification-Request,
    - 'Subscriber Process Identifier' = (the process identifier used in step 1),
    - 'Initiating Device Identifier' = TD,
    - 'Monitored Object Identifier' = X
    - 'Time Remaining' = (any value appropriate for the Lifetime selected),
    - 'List of Values' = (values appropriate to the property Y subscribed to, and any other properties the IUT provides with it, such as Status\_Flags)
  - RECEIVE BACnet-SimpleACK-PDU
5. MAKE (the IUT stop resubscribing, if it resubscribes automatically)
6. IF (the IUT can cancel the subscription) THEN
  - RECEIVE SubscribeCOVProperty – Request,
    - 'Subscriber Process Identifier' = (the process identifier used in step 1),
    - 'Monitored Object Identifier' = X
    - 'Monitored Property Identifier' = Y
    - 'COV Increment' = (Any REAL value –optional)

```

ELSE
    WAIT (a value two times Lifetime)
7. TRANSMIT ConfirmedCOVNotification-Request,
    'Subscriber Process Identifier' = (the process identifier used in step 1),
    'Initiating Device Identifier' = TD,
    'Monitored Object Identifier' = X
    'Time Remaining' = (any value appropriate for the Lifetime selected),
    'List of Values' = (values appropriate to the property Y subscribed to, and any other
                        properties the IUT provides with it, such as Status_Flags)
8. 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)

```

#### 8.11.X1.4 Change of Value Notifications with Invalid Monitored Object Identifier

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

Test Concept: A subscription for COV notifications is established, using a Lifetime of L. L shall be set to a value less than 24 hours and large enough to complete the test.

Test Steps:

```

1. MAKE (the IUT send a SubscribeCOVProperty-Request),
2. RECEIVE SubscribeCOVProperty-Request
    'Subscriber Process Identifier' = (any valid process identifier),
    'Monitored Object Identifier' = X
    'Issue Confirmed Notifications' = TRUE,
    'Lifetime' = L,
    'Monitored Property Identifier' = (the property Y to be monitored),
    'COV Increment' = (Any REAL value -- optional)
3. TRANSMIT BACnet-SimpleACK-PDU
4. TRANSMIT ConfirmedCOVNotification-Request,
    'Subscriber Process Identifier' = (the process identifier used in step 1),
    'Initiating Device Identifier' = TD,
    'Monitored Object Identifier' = (any object Y supporting COV notification except X),
    'Time Remaining' = (any value appropriate for the Lifetime selected),
    'List of Values' = (any value)
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)

```

### 8.11.X1.5 Change of Value Notifications with Invalid Monitored property

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

Test Concept: A subscription for COV notifications is established, using a Lifetime of L. L shall be set to a value less than 24 hours and large enough to complete the test.

Test Steps:

1. MAKE (the IUT send a SubscribeCOVProperty-Request),
2. RECEIVE SubscribeCOVProperty-Request
  - 'Subscriber Process Identifier' = (any valid process identifier),
  - 'Monitored Object Identifier' = X
  - 'Issue Confirmed Notifications' = TRUE,
  - 'Lifetime' = L,
  - 'Monitored Property Identifier' = (the property Y to be monitored),
  - 'COV Increment' = (Any REAL value -- optional)
3. TRANSMIT BACnet-SimpleACK-PDU
4. TRANSMIT ConfirmedCOVNotification-Request,
  - 'Subscriber Process Identifier' = (the process identifier used in step 1),
  - 'Initiating Device Identifier' = TD,
  - 'Monitored Object Identifier' = X
  - 'Time Remaining' = (any value appropriate for the Lifetime selected),
  - 'List of Values' = (any property supporting COV notification except Y),
5. IF (Protocol\_Revision is present and Protocol\_Revision  $\geq$  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)

### 8.11.X4 Requests 8 Hour Lifetimes

Purpose: To verify that the IUT correctly generates subscription requests with lifetimes less than or equal to 8 hours. Either confirmed or unconfirmed notifications may be used, but at least one of these options shall be supported by the IUT.

Test Steps:

1. MAKE (the IUT send a SubscribeCOVProperty-Request),
2. RECEIVE SubscribeCOVProperty-Request,
  - 'Subscriber Process Identifier' = (any valid process identifier),
  - 'Monitored Object Identifier' = X
  - 'Issue Confirmed Notifications' = TRUE | FALSE,
  - 'Lifetime' = (any valid lifetime between 1 and 28800),
  - 'Monitored Property Identifier' = (the property Y to be monitored),
  - 'COV Increment' = (Any REAL value -- optional)
3. TRANSMIT BACnet-SimpleACK-PDU

## 8.18 ReadProperty Service Initiation Tests

[Add new **Clause 8.18.X1**, p. 273]

[Reviewer Note: There is no appropriate test for reading and presenting large list property values as required by DM-SP-VM-A.]

### 8.18.X1 Reading and Presenting Large List Properties

Purpose: This test case verifies that the IUT is capable of reading and presenting large list properties using ReadRange. It is a generic test used to test data presentation requirements.

Configuration: For this test, the tester shall choose a list property, P1, from an object, O1. The TD shall be configured to not support segmentation. The value P1 shall be too large to read via ReadProperty or ReadPropertyMultiple.

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 value cannot be read using ReadProperty or ReadPropertyMultiple, the IUT may initiate a ReadProperty or ReadPropertyMultiple. If this occurs, the IUT shall pass the test only if it automatically falls back to using ReadRange upon receipt of the correct BACnetReject-PDU from the TD, indicating that the response is too large.

Test Steps:

1. MAKE (the IUT read P1)
2. WHILE (the complete list has not been read)
  - RECEIVE ReadRange-Request,
    - 'Object Identifier' = O1,
    - 'Property Identifier' = P1,
    - 'Range' = (any valid value for P1)
  - TRANSMIT BACnet-ComplexACK-PDU,
    - 'Object Identifier' = O1,
    - 'Property Identifier' = P1,
    - 'Result Flags' = (values consistent with the request),
    - 'Item Count' = (values consistent with the request),
    - 'Item Data' = (values consistent with the request)
3. CHECK (that the IUT presents a list of values that is consistent with the values received in step 2)

## 8.20 ReadPropertyMultiple Service Initiation Tests

### 8.20.5 Cases In Which ReadProperty Shall Be Used After ReadPropertyMultiple Fails

[Change **Clause 8.20.5.1**, p. 275]

[Reviewer Note: Modified test to allow multiple objects in addition to single objects. Notes to Tester added to permit a behavior that allows the IUT to omit RP requests for properties which are not essential to the operation being performed.]

#### 8.20.5.1 The IUT Determines the TD does not Support the ReadPropertyMultiple Service

Purpose: Verifies the IUT's ability to automatically change its service choice from ReadPropertyMultiple to ReadProperty when the IUT determines the TD does not support the ReadPropertyMultiple service.

Test Concept: The IUT is configured in a manner that would normally cause it to access one or more properties in the TD via the ReadPropertyMultiple service. Prior to sending a ReadPropertyMultiple request, however, the IUT determines that the TD

does not support the ReadPropertyMultiple service. The IUT instead attempts to access the TD's property values via the ReadProperty service (it is assumed that the IUT will make this determination by reading the TD's Protocol\_Services\_Supported property, but this test specifically does not attempt to verify this behavior).

Configuration Requirements: The TD is configured so that it does not support the ReadPropertyMultiple service. The IUT is configured such that it is ~~capable of~~ accessing one or more properties of a single *or multiple* objects in the TD via the ReadProperty and ReadPropertyMultiple services. ~~If the IUT cannot be configured in this way, then this test shall be omitted.~~

*Notes to Tester: In step 3, observing a ReadProperty of the properties from Step 1 is what is expected, but it is acceptable if fewer properties are read with ReadProperty, if those which are omitted are not essential to the operation being performed.*

Test Steps:

1. MAKE (a condition in the IUT that would normally cause it to send a ReadPropertyMultiple request to the TD to access one or more properties ~~ies values of a single object~~)
2. WAIT (a time interval specified by the vendor as sufficient for the IUT to determine that the TD does not support the ReadPropertyMultiple service)
3. REPEAT X = (the properties that the IUT is to read) DO {
  - RECEIVE ReadProperty-Request,
  - 'Object Identifier' = (object identifier referenced by X),
  - 'Property Identifier' = (property identifier referenced by X)
  - TRANSMIT ReadProperty-Ack,
  - 'Object Identifier' = (object identifier referenced by X),
  - 'Property Identifier' = (property identifier referenced by X),
  - 'Property Value' = (any valid value)

[Change **Clause 8.20.5.2**, p. 276]

[Reviewer Note: Modified test to allow multiple objects in addition to single objects. Notes to Tester added to permit a behavior that allows the IUT to omit RP requests for properties which are not essential to the operation being performed.]

### 8.20.5.2 Fallback to ReadProperty on Reject - UNRECOGNIZED\_SERVICE Response

~~BACnet Reference Clauses: 15.5 and 15.7~~

Purpose: Verifies the IUT's ability to automatically change its service choice from ReadPropertyMultiple to ReadProperty when the TD returns a Reject-PDU and a Reject Reason of UNRECOGNIZED\_SERVICE.

Test Concept: The IUT is configured to send the TD a ReadPropertyMultiple request to access one or more properties of one or more objects. The TD responds with a Reject-PDU and a Reject Reason of UNRECOGNIZED\_SERVICE. With no additional configuration, the IUT sends one or more ReadProperty requests to the TD, where each ReadProperty request specifies an individual property from the original ReadPropertyMultiple request covering all the properties from the original ReadPropertyMultiple request.

Configuration Requirements: The TD is configured so that it does not support the ReadPropertyMultiple service. The IUT is configured such that it attempts to acquire values from the TD using the ReadPropertyMultiple service without first interrogating the TD's Protocol\_Services\_Supported property. If the IUT cannot be configured in this way then this test shall be omitted.

*Notes to Tester: In step 4, observing a ReadProperty of the properties from Step 1 is what is expected, but it is acceptable if fewer properties are read with ReadProperty, if those which are omitted are not essential to the operation being performed.*

Test Steps:

1. MAKE (the IUT send a ReadPropertyMultiple-Request for a list of properties *L*)
2. RECEIVE ReadPropertyMultiple-Request,
  - the following is repeated for each object *O* referenced in *L*
  - 'Object Identifier' = (Object identifier of the specified object),
  - 'List of Property References' = (one or more properties of *O* as specified in *L* ~~the specified object~~)

3. TRANSMIT BACnet-Reject-PDU,  
    'Reject Reason' = UNRECOGNIZED\_SERVICE
4. REPEAT X = (the properties from *L* in the order the IUT chooses ~~Step 4~~) DO {  
    RECEIVE ReadProperty-Request,  
        'Object Identifier' = (object identifier referenced by X),  
        'Property Identifier' = (property identifier referenced by X)  
    TRANSMIT ReadProperty-Ack,  
        'Object Identifier' = (object identifier referenced by X),  
        'Property Identifier' = (property identifier referenced by X),  
        'Property Value' = (any valid value)  
    }

## 8.21 ReadRange Service Initiation Tests

[Change **Clause 8.21.1**, p. 276]

[Reviewer Note: Modified to comply with Addendum 135-2008u-3.]

### 8.21.1 Reading Values with no Specified Range

Purpose: To verify that the IUT can correctly initiate a ReadRange service request that does not specify any range of values to be returned.

Test Steps:

1. RECEIVE ReadRange-Request,  
    'Object Identifier' = (*O*, any ~~Trend Log~~ object),  
    'Property Identifier' = ~~Log Buffer~~(*P*, any list property the IUT can read)
2. TRANSMIT ReadRange-ACK  
    'Object Identifier' = *O*,  
    'Property Identifier' = *P*,  
    'Result Flags' = (*TRUE*, (*bLast*), (*NOT bLast*)),  
    'Item Count' = (*C*: any valid value)  
    'Item Data' = (*C* valid records for the requested property)
3. CHECK(that the IUT performs the vendor specified action)

[Change **Clause 8.21.3**, p. 277]

[Reviewer Note: Modified to comply with Addendum 135-2008u-3.]

### 8.21.3 Reading a Range of Values by Position

Purpose: To verify that the IUT can correctly initiate a ReadRange service request that specifies the range of values to be returned by position.

Test Steps:

1. RECEIVE ReadRange-Request,  
    'Object Identifier' = (*O*, any ~~Trend Log~~ object),  
    'Property Identifier' = ~~Log Buffer~~(*P*, any list property),  
    'Reference Index' = (any Unsigned value),  
    'Count' = (*CI*, any INTEGER value)
2. TRANSMIT ReadRange-ACK  
    'Object Identifier' = *O*,  
    'Property Identifier' = *P*,  
    'Result Flags' = ((*TRUE* if the first was requested, *FALSE* otherwise), *?*, *?*),



'Item Count' = (C2: any valid value <= |C|)

'Item Data' = (C2 valid records for the requested property)

3. CHECK(that the IUT performs the vendor specified action)

[Change **Clause 8.21.9**, p. 280]

[Reviewer Note: Modified the name of the test and improved the wording of the Purpose. Added 's' to Notes to Tester.]

### 8.21.9 Presents Log Records ~~Containing a Specific Datatype~~

Purpose: To verify that the IUT can initiate one or more ReadRange requests that access and present a tester-specified portion of log records ~~having a specific datatype, using any valid range~~. It is a generic test used to test data presentation requirements.

Test Concept: Run test in ~~Clause 135.1-2019~~ - 8.21.8 and verify that the data presentation meets the criteria specified by the BIBB being tested.

~~Note~~Notes to Tester: The values presented by the IUT may differ from the values transmitted on the wire due to rounding, truncation, formatting, language, conversion, etc.

~~Note~~Notes to Tester: The IUT is not required to display records containing log-status values.

## 8.22 WriteProperty Service Initiation Tests

[Add new **Clause 8.22.X1**, p. 282]

[Reviewer Note: There is no appropriate test for modifying large list property values as required by DM-SP-VM-A.]

### 8.22.X1 Accepting Input and Modifying Large List Properties

Purpose: This test case verifies that the IUT is capable of accepting user input and using it to modify large list properties where AddListElement and RemoveListElement will be required. It is a generic test used to test data-input requirements.

Configuration: For this test, the tester shall choose a list property, P1, from an object, O1. The TD shall be configured to not support segmentation. The list property shall be configured with a value such that it cannot be read or written without the use of ReadRange and AddListElement/RemoveListElement.

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 value cannot be transmitted using WriteProperty or WritePropertyMultiple, the IUT may initiate a WriteProperty or WritePropertyMultiple. If this occurs, the IUT shall pass the test only if it automatically falls back to using AddListElement and RemoveListElement upon receipt of the correct BACnetReject-PDU from the TD, indicating that the write request is too large.

Test Steps:

-- test adding elements into the list

1. MAKE (the IUT accept 1 or more new entries, {E1..En} for P1 from the user)
2. RECEIVE AddListElement-Request,
  - 'Object Identifier' = O1,
  - 'Property Identifier' = P1
  - 'List Of Elements' = (E1, ..., En)
3. TRANSMIT BACnet-SimpleACK-PDU

-- test removing elements from the list

4. MAKE (the IUT delete 1 or more entries, {E1..Ex} for P1 from the user)
5. RECEIVE RemoveListElement-Request,  
    'Object Identifier' = O1,  
    'Property Identifier' = P1  
    'List Of Elements' = (E1, .., Ex)
6. TRANSMIT BACnet-SimpleACK-PDU

[Add new **Clause 8.22.X4**, p. 282]

#### **8.22.X4 Writing Array Properties as a Whole Array**

Purpose: This test verifies that the IUT is writing the entire array to the TD without the use of the array index.

Configuration Requirements: For this test, the tester shall choose a property, P<sub>1</sub>, from an object, O<sub>1</sub>. The TD shall be configured to not support execution of WritePropertyMultiple.

The WriteProperty request initiated by IUT shall contain array of elements in P<sub>1</sub>, which shall fit in the APDU and segment limitations of the IUT.

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 WritePropertyMultiple, the IUT may initiate a WritePropertyMultiple. If this occurs, the IUT shall pass the test only if it automatically falls back to using WriteProperty upon receipt of the correct BACnetReject-PDU from the TD, indicating that WritePropertyMultiple is not supported.

Notes to Tester: Any WriteProperty request generated by the IUT may have a Priority parameter. If included, it shall be in the range 1-16, excluding 6.

Test Steps:

1. MAKE (the IUT accept a new value for P<sub>1</sub> including all elements of the array from the user)
2. RECEIVE WriteProperty-Request,  
    'Object Identifier' = O<sub>1</sub>,  
    'Property Identifier' = P<sub>1</sub>,  
    'Property Value' = (the value provided to the IUT for P<sub>1</sub>)
3. TRANSMIT BACnet-SimpleACK-PDU

#### **8.24 DeviceCommunicationControl Service Initiation Tests**

[Change **Clause 8.24.1**, p. 285]

[Reviewer Note: Modified to include the responding ACK.]

##### **8.24.1 Indefinite Duration, Disable, No Password**

Purpose: To verify that the IUT can initiate DeviceCommunicationControl service requests that indicate communication should cease for an indefinite time duration and do not convey a password.

Test Steps:

1. RECEIVE DeviceCommunicationControl-Request,  
    'Enable/Disable' = DISABLE,

## 2. TRANSMIT BACnet-SimpleACK-PDU

[Change **Clause 8.24.2**, p. 285]

[Reviewer Note: This test was modified to remove the requirement of a minimum password length of 5 but include the requirement of up to 20 characters.]

### 8.24.2 Indefinite Duration, Disable, Password

Purpose: To verify that the IUT can initiate DeviceCommunicationControl service requests that indicate communication should cease for an indefinite time duration and convey a password.

Test Steps:

1. RECEIVE DeviceCommunicationControl-Request,  
'Enable/Disable' = DISABLE,  
'Password' = ~~(a password of at least 5 characters)~~ *(a password of up to 20 characters)*
2. TRANSMIT BACnet-SimpleACK-PDU

[Change **Clause 8.24.3**, p. 285]

[Reviewer Note: This test was modified to remove the requirement of a minimum password length of 5 but include the requirement of up to 20 characters.]

### 8.24.3 Time Duration, Disable, Password

Purpose: To verify that the IUT can initiate DeviceCommunicationControl service requests that indicate communication should cease for a specific time duration and convey a password.

Test Steps:

1. RECEIVE DeviceCommunicationControl-Request,  
'Time Duration' = (any unsigned value > 0),  
'Enable/Disable' = DISABLE,  
'Password' = ~~(a password of at least 5 characters)~~ *(a password of up to 20 characters)*
2. TRANSMIT BACnet-SimpleACK-PDU

[Change **Clause 8.24.4**, p. 285]

[Reviewer Note: This test was modified to remove the requirement of a minimum password length of 5 but include the requirement of up to 20 characters.]

### 8.24.4 Enable, Password

Purpose: To verify that the IUT can initiate DeviceCommunicationControl service requests that indicate communication should resume and convey a password.

Test Steps:

1. RECEIVE DeviceCommunicationControl-Request,  
'Enable/Disable' = ENABLE,  
'Password' = ~~(a password of at least 5 characters)~~ *(a password of up to 20 characters)*
2. TRANSMIT BACnet-SimpleACK-PDU

[Change **Clause 8.24.5**, p. 285]

[Reviewer Note: This test was modified to include the responding ACK.]

#### **8.24.5 Enable, No Password**

Purpose: To verify that the IUT can initiate DeviceCommunicationControl service requests that indicate communication should resume and do not convey a password.

Test Steps:

1. RECEIVE DeviceCommunicationControl-Request,  
'Enable/Disable' = ENABLE,
2. TRANSMIT BACnet-SimpleACK-PDU

[Change **Clause 8.24.6**, p. 286]

[Reviewer Note: This test was modified to include the responding ACK.]

#### **8.24.6 Time Duration, Disable, No Password**

Purpose: To verify that the IUT can initiate DeviceCommunicationControl service requests that indicate communication should cease for a specific time duration and do not convey a password. If the IUT does not support the “no password” option, this test shall not be performed.

Test Steps:

1. RECEIVE DeviceCommunicationControl-Request,  
'Time Duration' = (any unsigned value > 0),  
'Enable/Disable' = DISABLE
2. TRANSMIT BACnet-SimpleACK-PDU

[Change **Clause 8.24.7**, p. 286]

[Reviewer Note: This test was modified to remove the requirement of a minimum password length of 5 but include the requirement of up to 20 characters.]

#### **8.24.7 Time Duration, Disable-Initiation, Password**

Purpose: To verify that the IUT can initiate DeviceCommunicationControl service requests that indicate communication should cease for a specific time duration and that convey a password.

Test Steps:

1. RECEIVE DeviceCommunicationControl-Request,  
'Time Duration' = (any unsigned value in the range from 1 to 65535),  
'Enable/Disable' = DISABLE  
'Password' = (a password of up to 20 characters)
2. TRANSMIT BACnet-SimpleACK-PDU

### **8.27 ReinitializeDevice Service Initiation Tests**

[Change **Clause 8.27.2**, p. 287]

[Reviewer Note: This test was modified to remove the requirement of a minimum password length of 5 but include the requirement of up to 20 characters.]

### **8.27.2 COLDSTART with a Password**

Purpose: To verify that the IUT can initiate ReinitializeDevice service requests that indicate a COLDSTART should be performed and convey a password.

Test Steps:

1. RECEIVE ReinitializeDevice-Request,  
'Reinitialized State of Device' = COLDSTART,  
'Password' = ~~(a password of at least 5 characters)~~ *(a password of up to 20 characters)*
2. TRANSMIT BACnet-SimpleACK-PDU

[Change **Clause 8.27.4**, p. 287]

[Reviewer Note: This test was modified to remove the requirement of a minimum password length of 5 but include the requirement of up to 20 characters.]

### **8.27.4 WARMSTART with a Password**

Purpose: To verify that the IUT can initiate ReinitializeDevice service requests that indicate a WARMSTART should be performed and convey a password.

Test Steps:

1. RECEIVE ReinitializeDevice-Request,  
'Reinitialized State of Device' = WARMSTART,  
'Password' = ~~(a password of at least 5 characters)~~ *(a password of up to 20 characters)*
2. TRANSMIT BACnet-SimpleACK-PDU

## **8.32 Who-Has Service Initiation Tests**

[Change **Clause 8.32.1**, p. 290]

[Reviewer Note: Modified to allow the IUT to send and receive a unicast response (per Addendum 135-2012ar-5, Protocol Revision 15). Added Notes to Tester.]

### **8.32.1 Object Identifier Selection with no Device Instance Range**

Purpose: To verify that the IUT can initiate Who-Has service requests using the object identifier form with no device instance range. If the IUT cannot be caused to issue a Who-Has request of this form, then this test shall be omitted.

*Notes to Tester: If there is no vendor-defined observable action, then test step 3 can be skipped.*

Test Steps:

1. RECEIVE  
DESTINATION = LOCAL BROADCAST | GLOBAL BROADCAST  
SOURCE = IUT,

Who-Has-Request,  
'Object Identifier' = *ObjectI* (~~any object identifier~~)

2. TRANSMIT  
    *DESTINATION* = IUT | LOCAL BROADCAST | GLOBAL BROADCAST  
    *SOURCE* = TD,  
    I-Have-Request,  
    'Device Identifier' = (the TD's Device object)  
    'Object Identifier' = *ObjectI*
- 3 CHECK (for any vendor-defined observable actions)

[Change **Clause 8.32.2**, p. 290]

[Reviewer Note: Modified to allow the IUT to send and receive a unicast response (per Addendum 135-2012ar-5, Protocol Revision 15). Added Notes to Tester.]

### 8.32.2 Object Name Selection with no Device Instance Range

Purpose: To verify that the IUT can initiate Who-Has service requests using the object name form with no device instance range. If the IUT cannot be caused to issue a Who-Has request of this form, then this test shall be omitted.

*Notes to Tester: If there is no vendor-defined observable action, then test step 3 can be skipped.*

Test Steps:

1. RECEIVE  
    *DESTINATION* = LOCAL BROADCAST | GLOBAL BROADCAST,  
    *SOURCE* = IUT,  
    Who-Has-Request,  
    'Object Name' = *VI* (~~any CharacterString~~)
2. TRANSMIT  
    *DESTINATION* = IUT | LOCAL BROADCAST | GLOBAL BROADCAST  
    *SOURCE* = TD,  
    I-Have-Request,  
    'Device Identifier' = (the TD's Device object)  
    'Object Name' = *VI*
3. CHECK (for any vendor-defined observable actions)

[Change **Clause 8.32.3**, p. 290]

[Reviewer Note: The allowed device instance range is from 0 - 4194303 and is specified in sections 16.9.1.1.1 and 16.10.1.1.1. The test incorrectly set the low limit to 1. The allowance for Unicast I-Have is added.]

### 8.32.3 Object Identifier Selection with a Device Instance Range

Purpose: To verify that the IUT can initiate Who-Has service requests using the object identifier form with a device instance range. If the IUT cannot be caused to issue a Who-Has request of this form, then this test shall be omitted.

*Notes to Tester: Device instance range should be selected to cover TD's device object identifier. If there is no vendor-defined observable action, then test step 3 can be skipped.*

Test Steps:

1. RECEIVE  
    *DESTINATION* = LOCAL BROADCAST | GLOBAL BROADCAST,  
    *SOURCE* = IUT,  
    Who-Has-Request,  
    'Device Instance Range Low Limit' = (any integer X:  $40 \leq X \leq$  'Device Instance Range High Limit'),  
    'Device Instance Range High Limit' = (any integer Y: 'Device Instance Range Low Limit'  $\leq Y \leq$  4,194,303),

'Object Identifier' = *Object1* (~~any object identifier~~)

2. TRANSMIT

*DESTINATION* = IUT | LOCAL BROADCAST | GLOBAL BROADCAST

*SOURCE* = TD,

*I-Have-Request*,

'Device Identifier' = (the TD's Device object)

'Object Identifier' = *Object1*

3. CHECK (for any vendor-defined observable actions)

[Change **Clause 8.32.4**, p. 291]

[Reviewer Note: The allowed device instance range is from 0 - 4194303 and is specified in sections 16.9.1.1.1 and 16.10.1.1.1. The test incorrectly set the low limit to 1. The allowance for Unicast I-Have is added.]

### 8.32.4 Object Name Selection with a Device Instance Range

Purpose: To verify that the IUT can initiate Who-Has service requests using the object name form with a device instance range. If the IUT cannot be caused to issue a Who-Has request of this form, then this test shall be omitted.

*Notes to Tester: Device instance range should be selected to cover TD's device object identifier. If there is no vendor-defined observable action, then test step 3 can be skipped.*

Test Steps:

1. RECEIVE

*DESTINATION* = LOCAL BROADCAST | GLOBAL BROADCAST,

*SOURCE* = IUT,

*Who-Has-Request*,

'Device Instance Range Low Limit' = (any integer X:  $40 \leq X \leq$  'Device Instance Range High Limit'),

'Device Instance Range High Limit' = (any integer Y: 'Device Instance Range Low Limit'  $\leq Y \leq$  4,194,303),

'Object Name' = *V1* (~~any CharacterString~~)

2. TRANSMIT

*DESTINATION* = IUT | LOCAL BROADCAST | GLOBAL BROADCAST

*SOURCE* = TD,

*I-Have-Request*,

'Device Identifier' = (the TD's Device object)

'Object Name' = *V1*

3. CHECK (for any vendor-defined observable actions)

### 8.34 Who-Is Service Initiation Tests

[Change **Clause 8.34.2**, p. 291]

[Reviewer Note: The allowed device instance range is from 0 - 4194303 and is specified in sections 16.9.1.1.1 and 16.10.1.1.1. The test incorrectly set the low limit to 1.]

#### 8.34.2 Who-Is Request with a Device Instance Range

Purpose: To verify that the IUT can initiate Who-Is service requests with a device instance range. If the IUT cannot be caused to issue a Who-Is request of this form, then this test shall be omitted.

Test Steps:

1. RECEIVE

*DESTINATION* = LOCAL BROADCAST | GLOBAL BROADCAST,

*SOURCE* = IUT,

Who-Is-Request,

'Device Instance Range Low Limit' = (any integer X:  $40 \leq X \leq$  'Device Instance Range High Limit'),

'Device Instance Range High Limit' = (any integer Y: 'Device Instance Range Low Limit'  $\leq Y \leq 4,194,303$ )

[Add new **Clause 8.X2**, p. 300]

## 8.X2 WriteGroup Service Initiation Tests

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

### 8.X2.1 Broadcasting to a Group of Channel Objects

Purpose: Verify that the IUT can initiate a WriteGroup request to an arbitrary group with an arbitrary channel number.

Test Concept: Make the IUT perform a WriteGroup request to a tester selected group and channel. Verify that the request is generated.

Test Steps:

1. MAKE(the IUT initiate a WriteGroup request to group G and Channel C)
2. RECEIVE WriteGroup-Request
  - DESTINATION = GLOBAL\_BROADCAST | LOCAL\_BROADCAST |
  - REMOTE\_BROADCAST | TD,
  - 'Group Number' = G,
  - 'Write Priority' = (any valid value),
  - 'Change List' = (a valid list of 1 or more changes which impact channel C),
  - 'Inhibit Delay' = (absent or TRUE or FALSE)

[Add new **Clause 8.X12**, p. 300]

## 8.X12 SubscribeCOVPropertyMultiple Service Initiation Tests

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

### 8.X12.1 Positive SubscribeCOVPropertyMultiple Service Initiation Tests

#### 8.X12.1.1 Confirmed Notifications Subscription

Purpose: To verify the client can subscribe to confirmed notifications using the SubscribeCOVPropertyMultiple service.

Test Concept: The IUT is made to subscribe for confirmed notifications.

Test Steps:

1. MAKE (the IUT send a SubscribeCOVPropertyMultiple-Request),
2. RECEIVE SubscribeCOVPropertyMultiple-Request
  - 'Subscriber Process Identifier' = (any valid process identifier),
  - 'Issue Confirmed Notifications' = TRUE,
  - 'Lifetime' = L,
  - 'Max Notification Delay' = (any valid notification delay),
  - 'List of COV Subscription Specifications' = (any valid list of subscriptions)
3. TRANSMIT BACnet-SimpleAck-PDU



#### 8.X12.1.2 Unconfirmed Notifications Subscription

Purpose: To verify the client can subscribe to unconfirmed notifications using the SubscribeCOVPropertyMultiple service.

Test Concept: The IUT is made to subscribe for unconfirmed notifications.

Test Steps:

1. MAKE (the IUT send a SubscribeCOVPropertyMultiple-Request),
2. RECEIVE SubscribeCOVPropertyMultiple-Request  
'Subscriber Process Identifier' = (any valid process identifier),  
'Issue Confirmed Notifications' = FALSE,  
'Lifetime' = L,  
'Max Notification Delay' = (any valid notification delay),  
'List of COV Subscription Specifications' = (any valid list of subscriptions)
3. TRANSMIT BACnet-SimpleAck-PDU

#### 8.X12.1.3 Requests 8 Hour Lifetimes

Purpose: To verify that the IUT is able to provide a lifetime which is less than or equal to 8 hours for any SubscribeCOVPropertyMultiple request it generates.

Test Concept: The tester selects any of the possible COVM subscriptions that the IUT is able to generate, and it is configured to use a lifetime less than or equal to 8 hours. The IUT is made to send the subscription, and the lifetime is verified to be less than or equal to 8 hours.

Test Steps:

1. MAKE (the IUT send the selected SubscribeCOVPropertyMultiple-Request),
2. RECEIVE SubscribeCOVPropertyMultiple-Request,  
'Subscriber Process Identifier' = (any valid process identifier),  
'Issue Confirmed Notifications' = TRUE | FALSE,  
'Lifetime' = (any value  $\leq$  28800),  
'Max Notification Delay' = (any valid delay between 1 and 3600),  
'List of COV Subscription Specifications' = (a valid list of COV Specifications)
3. TRANSMIT BACnet-SimpleACK-PDU

#### 8.X12.1.4 Subscribe to Timestamped Notifications

Purpose: To verify the client can subscribe to timestamped notifications using the SubscribeCOVPropertyMultiple service.

Test Concept: A subscription for timestamped COVM notifications is established with Lifetime L for property P1 of Object O1. L shall be less than 8 hours but large enough to complete the test.

Test Steps:

1. MAKE (the IUT send a SubscribeCOVPropertyMultiple-Request),
2. RECEIVE SubscribeCOVPropertyMultiple-Request  
'Subscriber Process Identifier' = (any valid process identifier),  
'Issue Confirmed Notifications' = TRUE | FALSE,  
'Lifetime' = L,  
'Max Notification Delay' = (any valid notification delay),  
'List of COV Subscription Specifications' = (any valid list with at least 1 entry where 'Timestamped' is TRUE)
3. TRANSMIT BACnet-SimpleAck-PDU

#### 8.X12.1.5 Subscribe to Two Properties in a Single Object

Purpose: To verify that the IUT can subscribe to 2 or more properties from a single object.

Test Concept: A subscription for COVM notifications is established for properties from a single object.

#### Test Steps:

1. MAKE (the IUT send a SubscribeCOVPropertyMultiple-Request),
2. RECEIVE SubscribeCOVPropertyMultiple-Request
  - 'Subscriber Process Identifier' = (any valid process identifier),
  - 'Issue Confirmed Notifications' = TRUE | FALSE,
  - 'Lifetime' = L,
  - 'Max Notification Delay' = (any valid notification delay),
  - 'List of COV Subscription Specifications' = (a valid list of 2 or more properties from a single object)
3. TRANSMIT BACnet-SimpleAck-PDU

### 8.X12.1.6 Subscribe to Properties in Multiple Objects Using a Single Request

Purpose: To verify the client can subscribe to properties from multiple objects.

Test Concept: A subscription for notifications is established for properties from 2 or more objects.

#### Test Steps:

1. MAKE (the IUT send a SubscribeCOVPropertyMultiple-Request),
2. RECEIVE SubscribeCOVPropertyMultiple-Request
  - 'Subscriber Process Identifier' = (PID: any valid process identifier),
  - 'Issue Confirmed Notifications' = TRUE | FALSE,
  - 'Lifetime' = L,
  - 'Max Notification Delay' = (any valid notification delay),
  - 'List of COV Subscription Specifications' = (PROPS: any valid list of properties from 2 or more objects)
4. TRANSMIT BACnet-SimpleAck-PDU

### 8.X12.1.7 Change of Value Multiple Notification

Purpose: To verify that the IUT accepts COVM notifications for properties which it subscribed to.

Test Concept: A subscription for COVM notifications is established, a notification is sent to the IUT, and the vendor defined actions are verified.

#### Test Steps:

1. MAKE (the IUT send a SubscribeCOVPropertyMultiple-Request),
2. RECEIVE SubscribeCOVPropertyMultiple-Request
  - 'Subscriber Process Identifier' = (ID1: any valid process identifier),
  - 'Issue Confirmed Notifications' = TRUE | FALSE,
  - 'Lifetime' = (L: any valid lifetime),
  - 'Max Notification Delay' = (any valid delay between 1 and 3600),
  - 'List of COV Subscription Specifications' = (PROPS: any valid list of subscriptions)
3. TRANSMIT BACnet-SimpleACK-PDU
4. IF (the subscription was for confirmed notifications) THEN
  - TRANSMIT ConfirmedCOVNotificationMultiple-Request,
    - 'Subscriber Process Identifier' = ID1,
    - 'Initiating Device Identifier' = TD,
    - 'Time Remaining' = (any value ~L),
    - 'Timestamp' = (any valid value, or absent if subscribed to non-timestamped notifications),
    - 'List of COV Notifications' = (values appropriate to each entry in PROPS)
  - RECEIVE BACnet-SimpleACK-PDU
- ELSE
  - TRANSMIT UnconfirmedCOVNotificationMultiple-Request,
    - 'Subscriber Process Identifier' = ID1,
    - 'Initiating Device Identifier' = TD,
    - 'Time Remaining' = (any value ~L),
    - 'Timestamp' = (any valid value, or absent if subscribed to non-timestamped notifications)

- notifications),
- 'List of COV Notifications' = (values appropriate to each entry in PROPS)
5. CHECK (verify that any appropriate functions defined by the manufacturer, such as displaying information on a workstation screen are carried out)

### 8.X12.1.8 Canceling a Subscription

Purpose: To verify the client can cancel a COVM subscription.

Test Concept: A subscription for COVM notifications is established with a lifetime L, which is long enough to complete the test. The client is made to cancel the subscription by sending a SubscribeCOVPropertyMultiple request with Lifetime, and Max Notification Delay absent.

Test Steps:

1. MAKE (the IUT send a SubscribeCOVPropertyMultiple-Request),
2. RECEIVE SubscribeCOVPropertyMultiple-Request
  - 'Subscriber Process Identifier' = (the process identifier used in step 1),
  - 'Issue Confirmed Notifications' = TRUE | FALSE,
  - 'Lifetime' = L,
  - 'Max Notification Delay' = (any valid delay between 1 and 3600),
  - 'List of COV Subscription Specifications' = (PROPS: a valid list of COV Subscription Specifications)
3. TRANSMIT BACnet-SimpleAck-PDU
4. IF confirmed notifications were subscribed for THEN
  - TRANSMIT ConfirmedCOVNotificationMultiple-Request
    - 'Subscriber Process Identifier' = PID,
    - 'Initiating Device Identifier' = (TD's device identifier),
    - 'Time Remaining' = (a value ~ L),
    - 'Timestamp' = (a valid value, or absent if Time Of Change was not requested in the subscription)
    - 'List of COV Notifications' = (a valid list containing an entry for each entry in PROPS)
  - RECEIVE BACnet-SimpleAck-PDU
- ELSE
  - TRANSMIT UnconfirmedCOVNotificationMultiple-Request
    - 'Subscriber Process Identifier' = PID,
    - 'Initiating Device Identifier' = (TD's device identifier),
    - 'Time Remaining' = (a value ~ L),
    - 'Timestamp' = (a valid value, or absent if Time Of Change was not requested in the subscription)
    - 'List of COV Notifications' = (a valid list containing an entry for each entry in PROPS)
5. MAKE (the IUT cancel the subscription)
6. RECEIVE SubscribeCOVPropertyMultiple-Request
  - 'Subscriber Process Identifier' = (the process identifier used in step 2),
  - 'Issue Confirmed Notifications' = (the same value used in step 2),
  - 'Lifetime' = (absent)
  - 'Max Notification Delay' = (absent)
  - 'List of COV Subscription Specifications' = (PROPS, or an empty list)
7. TRANSMIT BACnet-SimpleAck-PDU

### 8.X12.2 Negative SubscribeCOVPropertyMultiple Service Initiation Tests

#### 8.X12.2.1 Change of Value Multiple Notification Arrives After Subscription Has Expired

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

Test Concept: A subscription for COVM notifications is established and then cancelled or allowed to expire. A ConfirmedCOVNotificationMultiple is then sent to the IUT to verify it returns either the appropriate error or a SimpleACK.

#### Test Steps:

1. MAKE (the IUT send a SubscribeCOVPropertyMultiple-Request),
2. RECEIVE SubscribeCOVPropertyMultiple-Request
  - 'Subscriber Process Identifier' = (ID1: any valid process identifier),
  - 'Issue Confirmed Notifications' = TRUE,
  - 'Lifetime' = (L: any valid lifetime),
  - 'Max Notification Delay' = (any valid notification delay),
  - 'List of COV Subscription Specifications' = (PROPS: any valid list of COV subscriptions)
3. TRANSMIT BACnet-SimpleACK-PDU
4. TRANSMIT ConfirmedCOVNotificationMultiple-Request,
  - 'Subscriber Process Identifier' = ID1,
  - 'Initiating Device Identifier' = TD,
  - 'Time Remaining' = (a value  $\sim$  L),
  - 'Timestamp' = (any appropriate value or absent if it is not a timestamped subscription)
  - 'List of COV Notifications' = (values appropriate to the properties in PROPS)
5. RECEIVE BACnet-SimpleACK-PDU
6. IF (the IUT can cancel the subscription) THEN
  - MAKE (the IUT cancel the subscription),
  - RECEIVE SubscribeCOVPropertyMultiple-Request,
    - 'Subscriber Process Identifier' = ID1,
    - 'Issue Confirmed Notifications' = TRUE,
    - 'Lifetime' = (absent)
    - 'Max Notification Delay' = (absent)
    - 'List of COV Subscription Specifications' = (PROPS or an empty list)
  - ELSE
    - WAIT (2 \* L seconds)
7. TRANSMIT ConfirmedCOVNotificationMultiple-Request,
  - 'Subscriber Process Identifier' = ID1,
  - 'Initiating Device Identifier' = TD,
  - 'Time Remaining' = (a value  $\sim$  L),
  - 'Timestamp' = (any appropriate value or absent if it is not a timestamped subscription)
  - 'List of COV Notifications' = (values appropriate to the properties in PROPS)
8. RECEIVE BACnet-Error-PDU,
  - 'Error Class' = SERVICES,
  - 'Error Code' = UNKNOWN\_SUBSCRIPTION |
  - (BACnet-SimpleAck-PDU)

#### 8.X12.2.2 Unknown Subscription

Purpose: To verify that an appropriate response is returned if a COVM notification arrives that contains arguments or parameters which do not match any current subscriptions.

Test Concept: The TD sends a ConfirmedCOVNotificationMultiple-Request which does not correspond to any existing subscriptions. Verify that the IUT responds with either an error message or a SimpleACK.

Configuration Requirements: At the start of the test, the IUT shall have no outstanding COVM subscriptions with TD using process identifier ID2.

#### Test Steps:

1. TRANSMIT ConfirmedCOVNotificationMultiple-Request,
  - 'Subscriber Process Identifier' = ID2,
  - 'Initiating Device Identifier' = TD,
  - 'Time Remaining' = (any valid value),
  - 'List of COV Notifications' = (any valid list of property notifications)
2. RECEIVE
  - (BACnet-Error-PDU,
  - 'Error Class' = SERVICES,
  - 'Error Code' = (UNKNOWN\_SUBSCRIPTION)) |

(BACnet-SimpleACK-PDU)

[Add new **Clause 8.X41**, p. 300]

### **8.X41 AuditLogQuery Initiation Tests**

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

#### **8.X41.1 Reading a Range of Items Using Any Valid Query**

**Purpose:** To verify that the IUT can initiate one or more AuditLogQuery requests that access a tester-specified portion of an audit log, using any valid range.

**Test Concept:** The TD contains an Audit Log object that has a logical set of log records, S1. The tester selects a portion of S1 to be returned and causes the IUT to request those records using any valid range. The test then verifies that the IUT can display the records in a manner consistent with those that the TD returns.

**Configuration Requirements:** The TD contains an audit log object, L1, which has a set of records, S1. The IUT is configured to display the returned set of log records.

**Test Steps:**

1. MAKE (L1 contain the set of records S1)
2. MAKE (the IUT request a range of samples from L1) {
3. WHILE (not all records from the tester-selected range have been returned)
  - RECEIVE AuditLogQuery-Request,
    - 'Audit Log' = (L1),
    - 'Query Parameters' = (any valid query),
    - 'Start at Sequence Number' = (an appropriate sequence number) -- or absent
    - 'Requested Count' = (any valid range)
  - TRANSMIT AuditLogQuery-ACK,
    - 'AuditLog' = (L1),
    - 'Records' = (a set of records appropriate for this response),
    - 'NoMoreItems' = (an appropriate value for this response)
- }
4. CHECK (the records returned in step 3 include the tester-selected range)
5. MAKE (the IUT display the tester-selected range)
6. CHECK (the records displayed in step 5 are consistent with the records returned in step 3)

## 135.1-2019s-9 Add new and correct existing Application Service Execution Tests

### Rationale

Errors have been identified in a number of application service execution tests in ANSI/ASHRAE Standard 135.1-2019.

In addition, the test coverage is increased by the addition of new tests.

[Change **Clause 9**, p. 301]

## 9. APPLICATION SERVICE EXECUTION TESTS

The test cases defined in this clause shall be used to verify that a BACnet device correctly implements the service procedure for the specified application service. BACnet devices shall be tested for the proper execution of each application service for which the PICS indicates execution is supported.

For each application service included in this clause several test cases are defined that collectively test the various options and features defined for the service in the BACnet standard. A test case is a sequence of one or more messages that are exchanged between the implementation under test (IUT) and the testing device (TD) in order to determine if a particular option or feature is correctly implemented. Multiple test cases that have a similar or related purpose are collected into test groups.

Under some circumstances an IUT may be unable to demonstrate conformance to a particular test case because the test applies to a feature that requires a particular BACnet object or optional property that is not supported in the IUT. For example, a device may support the File Access services but restrict files to stream access only. Such a device would have no way to demonstrate that it could implement the record access features of the File Access services. When this type of situation occurs the IUT shall be considered to be in conformance with BACnet provided the PICS documentation clearly indicates the restriction. Failure to document the restriction shall constitute nonconformance to the BACnet standard. All features and optional parameters for BACnet application services shall be supported unless a conflict arises because of unsupported objects or unsupported optional properties.

For each application service the tests are divided into two types, positive tests and negative tests. The positive tests verify that the IUT can correctly handle cases where the service is expected to be successfully completed. The negative tests verify correct handling for various error cases that may occur. Negative tests include inappropriate service parameters, but they do not include cases with encoding errors or otherwise malformed PDUs. Tests to ensure that the IUT can handle malformed PDUs are defined in 13.4.

Many test cases allow flexibility in the value to be used in a service parameter. The tester is free to choose any value within the constraints defined in the test case. The IUT shall be able to respond correctly to any valid selection the tester might make. The EPICS is considered to be a definitive reference indicating the BACnet functionality supported and the configuration of the object database. Any discrepancies between the BACnet functionality ~~or the value of properties in the object database as defined in the EPICS, and the values returned in messages defined for a test case constitutes a failure of the test. For example, if a test step involved reading a property of an object in the database the returned value must match the value provided in the EPICS.~~ *defined in the EPICS and the functionality demonstrated by the device during testing shall constitute a failure. For example, it is considered a failure if a test step involves writing to a property and the EPICS indicates the property is writable, but the device returns an error indicating 'write access denied'.*

### 9.1 AcknowledgeAlarm Service Execution Tests

#### 9.1.1 Positive AcknowledgeAlarm Service Execution Tests

[Change **Clause 9.1.1.1**, p. 301]

[Reviewer Note: Made changes to allow cases where only one Recipient\_List entry is supported. Made changes to include non-supported transitions.]

### 9.1.1.1 Successful Alarm Acknowledgment of Confirmed Event Notifications Using the Time Form of the 'Time of Acknowledgment' Parameter

Purpose: To verify the successful acknowledgment of an alarm signaled by a ConfirmedEventNotification, including notification of other workstations and updating of the Acked\_Transitions status. The Time form of the 'Time of Acknowledgment' parameter is used.

Test Concept: An alarm is triggered that causes the IUT to notify the TD ~~and one other device~~ *all other recipients in the Recipient\_List*. The TD acknowledges the alarm and verifies that the acknowledgment is properly noted by the IUT. The IUT notifies all other recipients that the alarm has been acknowledged.

Configuration Requirements: The IUT shall be configured with ~~at one~~ object that can detect alarm conditions and send confirmed notifications. The Acked\_Transitions property shall have the value B'111' indicating that all transitions have been acknowledged. The TD and one other BACnet device, if the IUT supports multiple recipients, shall be recipients of the alarm notification. *D1 is either the pTimeDelay, or pTimeDelayNormal parameter, or 0 (for transitions to and from FAULT state) depending on the event transition.*

*Notes to Tester: The destination address used for the acknowledgment notification in step 12 shall be the same address used in step 5. Inclusion of the 'To State' parameter in acknowledgement notifications was added in protocol version 1, protocol revision 1. Implementations that precede this version will not include this parameter. When multiple event notifications are expected for a specific event, the order that the IUT transmits them in is irrelevant. If the IUT can only be configured with one recipient in the Recipient\_List property of the issuing Notification\_class object, skip all steps related to receipt of the second notification.*

Test Steps:

1. MAKE (a change that triggers the detection of an alarm event in the IUT)
2. WAIT (~~pTimeDelay~~D1)
3. BEFORE **Notification Fail Time**
  - RECEIVE ConfirmedEventNotification-Request,
 

'Process Identifier' =	(the process identifier configured for this event),
'Initiating Device Identifier' =	IUT,
'Event Object Identifier' =	(the object detecting the alarm),
'Time Stamp' =	(T1: any valid time stamp),
'Notification Class' =	(the notification class configured for this event),
'Priority' =	(the priority configured for this event),
'Event Type' =	(any valid event type),
'Message Text' =	(optional, any valid message text),
'Notify Type' =	(the notify type configured for this event),
'AckRequired' =	TRUE,
'From State' =	<del>NORMAL</del> (any appropriate event state),
'To State' =	(any appropriate <del>non-normal</del> event state),
'Event Values' =	(the values appropriate to the event type)
  - 4. TRANSMIT BACnet-SimpleACK-PDU
  - 5. RECEIVE
    - DESTINATION = (at least one device other than the TD),
    - SOURCE = IUT,
    - ConfirmedEventNotification-Request,
 

'Process Identifier' =	(the process identifier configured for this event),
'Initiating Device Identifier' =	IUT,
'Event Object Identifier' =	(the object detecting the alarm),
'Time Stamp' =	(the timestamp or sequence number received in step 3),
'Notification Class' =	(the notification class configured for this event),
'Priority' =	(the priority configured for this event),
'Event Type' =	(any valid event type),
'Message Text' =	(optional, any valid message text),
'Notify Type' =	(the notify type configured for this event),
'AckRequired' =	TRUE,
'From State' =	<del>NORMAL</del> (any appropriate event state),
'To State' =	(any appropriate <del>non-normal</del> event state),
'Event Values' =	(the values appropriate to the event type)

6. TRANSMIT BACnet-SimpleACK-PDU
7. VERIFY (the 'Event Object Identifier' from the event notification), Acked\_Transitions = ~~(FALSE, TRUE, TRUE)~~ (appropriate bit FALSE, the others TRUE)
8. TRANSMIT AcknowledgeAlarm-Request,
 

'Acknowledging Process Identifier' =	(the value of the 'Process Identifier' parameter in the event notification),
'Event Object Identifier' =	(the 'Event Object Identifier' from the event notification),
'Event State Acknowledged' =	(the state specified in the 'To State' parameter of the notification),
'Time Stamp' =	(the time stamp conveyed in the notification),
'Time of Acknowledgment' =	(the TD's current time using a Time format)
9. RECEIVE BACnet-Simple-ACK-PDU
10. IF (Protocol\_Revision is present and Protocol\_Revision ≥ 1) THEN
 

**BEFORE Notification Fail Time**

RECEIVE ConfirmedEventNotification-Request,

'Process Identifier' =	(the process identifier configured for this event),
'Initiating Device Identifier' =	IUT,
'Event Object Identifier' =	(the object detecting the alarm),
'Time Stamp' =	(any valid time stamp),
'Notification Class' =	(the notification class configured for this event),
'Priority' =	(the priority configured for this event),
'Event Type' =	(the event type included in step 3),
'Message Text' =	(optional, any valid message text),
'Notify Type' =	ACK_NOTIFICATION,
'To State' =	(the 'To State' used in step 3)

ELSE

**BEFORE Notification Fail Time**

RECEIVE ConfirmedEventNotification-Request,

'Process Identifier' =	(the process identifier configured for this event),
'Initiating Device Identifier' =	IUT,
'Event Object Identifier' =	(the object detecting the alarm),
'Time Stamp' =	(any valid time stamp),
'Notification Class' =	(the notification class configured for this event),
'Priority' =	(the priority configured for this event),
'Event Type' =	(the event type included in step 3),
'Message Text' =	(optional, any valid message text),
'Notify Type' =	ACK_NOTIFICATION
'To State' =	<del>(the 'To State' used in step 3)</del>
11. TRANSMIT BACnet-SimpleACK-PDU
12. IF (Protocol\_Revision is present and Protocol\_Revision ≥ 1) THEN
 

**BEFORE Notification Fail Time**

RECEIVE

DESTINATION =	(at least one device other than the TD),
SOURCE =	IUT,

ConfirmedEventNotification-Request,

'Process Identifier' =	(the process identifier configured for this event),
'Initiating Device Identifier' =	IUT,
'Event Object Identifier' =	(the object detecting the alarm),
'Time Stamp' =	(the timestamp or sequence number received in step 10),
'Notification Class' =	(the notification class configured for this event),
'Priority' =	(the priority configured for this event),
'Event Type' =	(the event type included in step 3),
'Message Text' =	(optional, any valid message text),
'Notify Type' =	ACK_NOTIFICATION,
'To State' =	(the 'To State' used in step 3)

ELSE

**BEFORE Notification Fail Time**

RECEIVE

DESTINATION =	(at least one device other than the TD),
SOURCE =	IUT,

ConfirmedEventNotification-Request,



'Process Identifier' = (the process identifier configured for this event),  
 'Initiating Device Identifier' = IUT,  
 'Event Object Identifier' = (the object detecting the alarm),  
 'Time Stamp' = (the timestamp or sequence number received in step 10),  
 'Notification Class' = (the notification class configured for this event),  
 'Priority' = (the priority configured for this event),  
 'Event Type' = (the event type included in step 3),  
 'Message Text' = (optional, any valid message text),  
 'Notify Type' = ACK\_NOTIFICATION

13. TRANSMIT BACnet-SimpleACK-PDU

14. VERIFY (the 'Event Object Identifier' from the event notification), Acked\_Transitions = (TRUE,TRUE,TRUE)

~~Notes to Tester: The destination address used for the acknowledgment notification in step 12 shall be the same address used in step 5. Inclusion of the 'To State' parameter in acknowledgement notifications was added in protocol version 1, protocol revision 1. Implementations that precede this version will not include this parameter. When multiple event notifications are expected for a specific event, the order that the IUT transmits them in is irrelevant.~~

[Change Clause 9.1.1.4, p. 305]

[Reviewer Note: Made changes to include not supported transitions.]

#### 9.1.1.4 Successful Alarm Acknowledgment of Unconfirmed Event Notifications Using the Time Form of the 'Time of Acknowledgment' Parameter

Purpose: To verify the successful acknowledgment of an alarm signaled by an UnconfirmedEventNotification, including notification of other workstations and updating of the Acked\_Transitions status. The Time form of the 'Time of Acknowledgment' parameter is used.

Test Concept: An alarm is triggered that causes the IUT to notify the TD and one other device. The TD acknowledges the alarm and verifies that the acknowledgment is properly noted by the IUT. The IUT notifies all other recipients that the alarm has been acknowledged.

Configuration Requirements: The IUT shall be configured with at least one object that can detect alarm conditions and send unconfirmed notifications. The Acked\_Transitions property shall have the value B'111' indicating that all transitions have been acknowledged. The TD and one other BACnet device, if the IUT supports multiple recipients, shall be recipients of the alarm notification. *D1 is either the pTimeDelay, or pTimeDelayNormal parameter, or 0 (for transitions to and from FAULT state) depending on the event transition.*

~~Notes to Tester: The destination address used for the acknowledgment notification in step 8 shall be the same address used in step 3. The destination address used for the acknowledgment notification in step 9 shall be the same address used in step 4. Inclusion of the 'To State' parameter in acknowledgement notifications was added in protocol version 1, protocol revision 1. Implementations that precede this version will not include this parameter. When multiple event notifications are expected for a specific event, the order that the IUT transmits them in is irrelevant. If the IUT can only be configured with one recipient in the Recipient\_List property of the issuing Notification\_class object, omit steps 4 and 9.~~

Test Steps:

1. MAKE (a change that triggers the detection of an alarm event in the IUT)
2. WAIT (~~pTimeDelay~~D1)
3. BEFORE **Notification Fail Time**  
 RECEIVE,  
 DESTINATION = LOCAL BROADCAST | GLOBAL BROADCAST | TD,  
 SOURCE = IUT,  
 UnconfirmedEventNotification-Request,  
 'Process Identifier' = (the process identifier configured for this event),  
 'Initiating Device Identifier' = IUT,  
 'Event Object Identifier' = (the object detecting the alarm),  
 'Time Stamp' = (any valid time stamp),  
 'Notification Class' = (the notification class configured for this event),  
 'Priority' = (the priority configured for this event type),

'Event Type' = (any valid event type),  
'Message Text' = (optional, any valid message text),  
'Notify Type' = (the notify type configured for this event),  
'AckRequired' = TRUE,  
'From State' = ~~NORMAL~~(any appropriate event state),  
'To State' = (any appropriate event state),  
'Event Values' = (the values appropriate to the event type)

4. IF (the notification in step 3 was not a broadcast) THEN  
RECEIVE  
DESTINATION = (a device other than the TD),  
SOURCE = IUT,  
UnconfirmedEventNotification-Request,  
'Process Identifier' = (the process identifier configured for this event),  
'Initiating Device Identifier' = IUT,  
'Event Object Identifier' = (the object detecting the alarm),  
'Time Stamp' = (the timestamp or sequence number received in step 3),  
'Notification Class' = (the notification class configured for this event),  
'Priority' = (the priority configured for this event type),  
'Event Type' = (any valid event type),  
'Message Text' = (optional, any valid message text),  
'Notify Type' = (the notify type configured for this event),  
'AckRequired' = TRUE,  
'From State' = ~~NORMAL~~(any appropriate event state),  
'To State' = (any appropriate event state),  
'Event Values' = (the values appropriate to the event type)

5. VERIFY (the 'Event Object Identifier' from the event notification), Acked\_Transitions =  
~~(FALSE,TRUE,TRUE)~~(appropriate bit FALSE, the others TRUE)

6. TRANSMIT AcknowledgeAlarm-Request,  
'Acknowledging Process Identifier' = (the value of the 'Process Identifier' parameter in the event notification),  
'Event Object Identifier' = (the 'Event Object Identifier' from the event notification),  
'Event State Acknowledged' = (the state specified in the 'To State' parameter of the notification),  
'Time Stamp' = (any valid time stamp),  
'Acknowledgement Source' = (any valid value),  
'Time of Acknowledgment' = (the TD's current time using a Time format)

7. RECEIVE ~~BACnet Simple ACK PDU~~ BACnet-SimpleACK-PDU

8. IF (Protocol\_Revision is present AND Protocol\_Revision ≥ 1) THEN  
BEFORE Notification Fail Time  
RECEIVE  
DESTINATION = LOCAL BROADCAST | GLOBAL BROADCAST | TD,  
SOURCE = IUT,  
UnconfirmedEventNotification-Request,  
'Process Identifier' = (the process identifier configured for this event),  
'Initiating Device Identifier' = IUT,  
'Event Object Identifier' = (the object detecting the alarm),  
'Time Stamp' = (any valid time stamp),  
'Notification Class' = (the notification class configured for this event),  
'Priority' = (the priority configured for this event type),  
'Event Type' = (any valid event type),  
'Message Text' = (optional, any valid message text),  
'Notify Type' = ACK\_NOTIFICATION,  
'To State' = (the 'To State' used in step 3 or 4)

ELSE  
BEFORE Notification Fail Time  
RECEIVE  
DESTINATION = LOCAL BROADCAST | GLOBAL BROADCAST | TD,  
SOURCE = IUT,  
UnconfirmedEventNotification-Request,  
'Process Identifier' = (the process identifier configured for this event),  
'Initiating Device Identifier' = IUT,  
'Event Object Identifier' = (the object detecting the alarm),

```

        'Time Stamp' = (the current time or sequence number),
        'Notification Class' = (the notification class configured for this event),
        'Priority' = (the priority configured for this event type),
        'Event Type' = (any valid event type),
        'Message Text' = (optional, any valid message text),
        'Notify Type' = ACK_NOTIFICATION
9. IF (the notification in step 8 was not broadcast) THEN
    IF (Protocol_Revision is present AND Protocol_Revision ≥ 1) THEN
        RECEIVE
        DESTINATION = (at least one device other than the TD),
        SOURCE = IUT,
        UnconfirmedEventNotification-Request,
        'Process Identifier' = (the process identifier configured for this event),
        'Initiating Device Identifier' = IUT,
        'Event Object Identifier' = (the object detecting the alarm),
        'Time Stamp' = (the timestamp or sequence number from the notification in step 8),
        'Notification Class' = (the notification class configured for this event),
        'Priority' = (the priority configured for this event type),
        'Event Type' = (any valid event type),
        'Message Text' = (optional, any valid message text),
        'Notify Type' = ACK_NOTIFICATION,
        'To State' = (the 'To State' used in step 3 or 4)
    ELSE
        RECEIVE
        DESTINATION = (at least one device other than the TD),
        SOURCE = IUT,
        UnconfirmedEventNotification-Request,
        'Process Identifier' = (the process identifier configured for this event),
        'Initiating Device Identifier' = IUT,
        'Event Object Identifier' = (the object detecting the alarm),
        'Time Stamp' = (the timestamp or sequence number from the notification in step 8),
        'Notification Class' = (the notification class configured for this event),
        'Priority' = (the priority configured for this event type),
        'Event Type' = (any valid event type),
        'Message Text' = (optional, any valid message text),
        'Notify Type' = ACK_NOTIFICATION,
10. VERIFY (the 'Event Object Identifier' from the event notification), Acked_Transitions = (TRUE,TRUE,TRUE)

```

~~Notes to Tester: The destination address used for the acknowledgment notification in step 8 shall be the same address used in step 3. The destination address used for the acknowledgment notification in step 9 shall be the same address used in step 4. Inclusion of the 'To State' parameter in acknowledgement notifications was added in protocol version 1, protocol revision 1. Implementations that precede this version will not include this parameter. When multiple event notifications are expected for a specific event, the order that the IUT transmits them in is irrelevant.~~

[Change **Clause 9.1.1.10**, p. 313]

[Reviewer Note: Made changes to include not supported transitions.]

#### **9.1.1.10 Successful Alarm Re-Acknowledgment of Confirmed Event Notifications**

**Purpose:** To verify the successful re-acknowledgment of an event signaled by a ConfirmedEventNotification, including notification of other workstations and updating of the Acked\_Transitions status.

**Test Concept:** An event is triggered and the IUT notifies the TD and one other device. The TD acknowledges the event and verifies that the acknowledgment is properly noted by the IUT. The IUT notifies all recipients that the event has been acknowledged. The TD then acknowledges the event again, and the IUT again notifies all recipients. This behavior was not defined before Protocol\_Revision 7 and so this test shall only be performed if Protocol\_Revision is present (i.e., Protocol\_Revision ≥ 7).

Configuration Requirements: The IUT shall be configured with at least one event-initiating object that generates offnormal transitions and sends confirmed notifications. The Aacked\_Transitions property shall have the value B'111', indicating that all transitions have been acknowledged. The TD and one other BACnet device, if the IUT supports multiple recipients, shall be recipients of the event notification. *DI is either the pTimeDelay, or pTimeDelayNormal parameter, or 0 (for transitions to and from FAULT state) depending on the event transition.*

*Notes to Tester: The destination address used for the acknowledgment notification in steps 12 and 19 shall be the same address used in step 4. When multiple event notifications are expected for a specific event, the order that the IUT transmits them in is irrelevant. If the IUT can only be configured with one recipient in the Recipient\_List property of the issuing Notification\_class object, omit steps 5, 6, 12, 13, 19, and 20.*

#### Test Steps:

1. MAKE (a change that triggers the detection of an event in the IUT)
2. WAIT (~~pTimeDelay~~DI)
3. BEFORE Notification Fail Time
  - RECEIVE ConfirmedEventNotification-Request,
    - 'Process Identifier' = (the process identifier configured for this event),
    - 'Initiating Device Identifier' = IUT,
    - 'Event Object Identifier' = (the event-initiating object),
    - 'Time Stamp' = (T1: any valid time stamp),
    - 'Notification Class' = (the notification class configured for this event),
    - 'Priority' = (the priority configured for this event),
    - 'Event Type' = (E1: any valid event type),
    - 'Message Text' = (optional, any valid message text),
    - 'Notify Type' = (the notify type configured for this event),
    - 'AckRequired' = TRUE,
    - 'From State' = ~~NORMAL~~(any appropriate event state),
    - 'To State' = (S1: any appropriate ~~offnormal~~ event state),
    - 'Event Values' = (the values appropriate to the event type)
4. TRANSMIT BACnet-SimpleACK-PDU
5. RECEIVE
  - DESTINATION = (a device other than the TD),
  - SOURCE = IUT,
  - ConfirmedEventNotification-Request,
    - 'Process Identifier' = (the process identifier configured for this event),
    - 'Initiating Device Identifier' = IUT,
    - 'Event Object Identifier' = (the event-initiating object),
    - 'Time Stamp' = (T1),
    - 'Notification Class' = (the notification class configured for this event),
    - 'Priority' = (the priority configured for this event),
    - 'Event Type' = (any valid event type),
    - 'Message Text' = (optional, any valid message text),
    - 'Notify Type' = (the notify type configured for this event),
    - 'AckRequired' = TRUE,
    - 'From State' = ~~NORMAL~~(any appropriate event state),
    - 'To State' = (any appropriate ~~offnormal~~ event state),
    - 'Event Values' = (the values appropriate to the event type)
6. TRANSMIT BACnet-SimpleACK-PDU
7. VERIFY (the 'Event Object Identifier' from the event notification), Aacked\_Transitions = (~~FALSE,TRUE,TRUE~~)  
(appropriate bit FALSE, the others TRUE)
8. TRANSMIT AcknowledgeAlarm-Request,
  - 'Acknowledging Process Identifier' = (the value of the 'Process Identifier' parameter in the event notification),
  - 'Event Object Identifier' = (the 'Event Object Identifier' from the event notification),
  - 'Event State Acknowledged' = (the state specified in the 'To State' parameter of the notification),
  - 'Time Stamp' = (any valid time stamp),
  - 'Acknowledgment Source' = (any valid value)
  - 'Time of Acknowledgment' = (any of the forms specified for this parameter)
9. RECEIVE ~~BACnet-SimpleACK-PDU~~BACnet-SimpleACK-PDU
10. BEFORE Notification Fail Time

RECEIVE ConfirmedEventNotification-Request,  
 'Process Identifier' = (the process identifier configured for this event),  
 'Initiating Device Identifier' = IUT,  
 'Event Object Identifier' = (the event-initiating object),  
 'Time Stamp' = (T+2: any valid time stamp),  
 'Notification Class' = (the notification class configured for this event),  
 'Priority' = (the priority configured for this event),  
 'Event Type' = (E1),  
 'Message Text' = (optional, any valid message text),  
 'Notify Type' = ACK\_NOTIFICATION,  
 'To State' = (S1)

11. TRANSMIT BACnet-SimpleACK-PDU

12. RECEIVE

DESTINATION = (a device other than the TD),  
 SOURCE = IUT,  
 ConfirmedEventNotification-Request,  
 'Process Identifier' = (the process identifier configured for this event),  
 'Initiating Device Identifier' = IUT,  
 'Event Object Identifier' = (the event-initiating object),  
 'Time Stamp' = (T+2),  
 'Notification Class' = (the notification class configured for this event),  
 'Priority' = (the priority configured for this event),  
 'Event Type' = (E1),  
 'Message Text' = (optional, any valid message text),  
 'Notify Type' = ACK\_NOTIFICATION,  
 'To State' = (S1)

13. TRANSMIT BACnet-SimpleACK-PDU

14. VERIFY (the 'Event Object Identifier' from the event notification), Acked\_Transitions = (TRUE, TRUE, TRUE)

15. TRANSMIT AcknowledgeAlarm-Request,

'Acknowledging Process Identifier' = (the value of the 'Process Identifier' parameter in the event notification),  
 'Event Object Identifier' = (the 'Event Object Identifier' from the event notification),  
 'Event State Acknowledged' = (the state specified in the 'To State' parameter of the notification),  
 'Time Stamp' = (any valid time stamp),  
 'Acknowledgment Source' = (any valid value)  
 'Time of Acknowledgment' = (any of the forms specified for this parameter)

16. RECEIVE BACnet-SimpleACK-PDU

17. BEFORE Notification Fail Time

RECEIVE ConfirmedEventNotification-Request,  
 'Process Identifier' = (the process identifier configured for this event),  
 'Initiating Device Identifier' = IUT,  
 'Event Object Identifier' = (the event-initiating object),  
 'Time Stamp' = (T23: any valid time stamp),  
 'Notification Class' = (the notification class configured for this event),  
 'Priority' = (the priority configured for this event),  
 'Event Type' = (E1),  
 'Message Text' = (optional, any valid message text),  
 'Notify Type' = ACK\_NOTIFICATION,  
 'To State' = (S1)

18. TRANSMIT BACnet-SimpleACK-PDU

19. RECEIVE

DESTINATION = (at least one device other than the TD),  
 SOURCE = IUT,  
 ConfirmedEventNotification-Request,  
 'Process Identifier' = (the process identifier configured for this event),  
 'Initiating Device Identifier' = IUT,  
 'Event Object Identifier' = (the event-initiating object),  
 'Time Stamp' = (T23),  
 'Notification Class' = (the notification class configured for this event),  
 'Priority' = (the priority configured for this event),  
 'Event Type' = (E1),

'Message Text' = (optional, any valid message text),  
 'Notify Type' = ACK\_NOTIFICATION,  
 'To State' = (S1)

20. TRANSMIT BACnet-SimpleACK-PDU

21. VERIFY (the 'Event Object Identifier' from the event notification), Acked\_Transitions = (TRUE, TRUE, TRUE)

~~Notes to Tester: The destination address used for the acknowledgment notification in steps 12 and 19 shall be the same address used in step 4. When multiple event notifications are expected for a specific event, the order that the IUT transmits them in is irrelevant. If the IUT can only be configured with one recipient in the Recipient\_List property of the issuing Notification\_class object, omit steps 5, 6, 12, 13, 19, and 20.~~

[Change **Clause 9.1.1.11**, p. 315]

[Reviewer Note: Made changes to include not supported transitions.]

#### 9.1.1.11 Successful Alarm Re-Acknowledgment of Unconfirmed Event Notifications

Purpose: To verify the successful re-acknowledgment of an event signaled by an UnconfirmedEventNotification, including notification of other workstations and updating of the Acked\_Transitions status.

Test Concept: An event is triggered and the IUT notifies the TD and one other device. The TD acknowledges the event and verifies that the acknowledgment is properly noted by the IUT. The IUT notifies all recipients that the event has been acknowledged. The TD then acknowledges the event again, and the IUT again notifies all recipients. This behavior was not defined before Protocol\_Revision 7 and so this test shall only be performed if Protocol\_Revision is present (i.e., Protocol\_Revision ≥ 7).

Configuration Requirements: The IUT shall be configured with at least one event-initiating object that generates offnormal transitions and sends unconfirmed notifications. The Acked\_Transitions property shall have the value B'111', indicating that all transitions have been acknowledged. The TD and one other BACnet device, if the IUT supports multiple recipients, shall be recipients of the event notification. *DI is either the pTimeDelay, or pTimeDelayNormal parameter, or 0 (for transitions to and from FAULT state) depending on the event transition.*

~~Notes to Tester: The destination address used for the acknowledgment notification in steps 9 and 14 shall be the same address used in step 4. When multiple event notifications are expected for a specific event, the order that the IUT transmits them in is irrelevant. If the IUT can only be configured with one recipient in the Recipient\_List property of the issuing Notification\_class object, omit steps 4, 9, and 14.~~

Test Steps:

1. MAKE (a change that triggers the detection of an offnormal event in the IUT)
2. WAIT (~~pTimeDelay~~DI)
3. BEFORE **Notification Fail Time**
  - RECEIVE UnconfirmedEventNotification-Request,
    - 'Process Identifier' = (the process identifier configured for this event),
    - 'Initiating Device Identifier' = IUT,
    - 'Event Object Identifier' = (the event-initiating object),
    - 'Time Stamp' = (T1: any valid time stamp),
    - 'Notification Class' = (the notification class configured for this event),
    - 'Priority' = (the priority configured for this event),
    - 'Event Type' = (E1: any valid event type),
    - 'Message Text' = (optional, any valid message text),
    - 'Notify Type' = (the notify type configured for this event),
    - 'AckRequired' = TRUE,
    - 'From State' = ~~NORMAL~~(any appropriate event state),
    - 'To State' = (S1: any appropriate ~~offnormal~~ event state),
    - 'Event Values' = (the values appropriate to the event type)
4. RECEIVE
  - DESTINATION = (at least one device other than the TD),
  - SOURCE = IUT,

UnconfirmedEventNotification-Request,

'Process Identifier' = (the process identifier configured for this event),  
 'Initiating Device Identifier' = IUT,  
 'Event Object Identifier' = (the event-initiating object),  
 'Time Stamp' = (T1),  
 'Notification Class' = (the notification class configured for this event),  
 'Priority' = (the priority configured for this event),  
 'Event Type' = (any valid event type),  
 'Message Text' = (optional, any valid message text),  
 'Notify Type' = (the notify type configured for this event),  
 'AckRequired' = TRUE,  
 'From State' = ~~NORMAL~~ (any appropriate event state),  
 'To State' = (any appropriate ~~offnormal~~ event state),  
 'Event Values' = (the values appropriate to the event type)

5. VERIFY (the 'Event Object Identifier' from the event notification), Acked\_Transitions = ~~(FALSE, TRUE, TRUE)~~ (appropriate bit FALSE, the others TRUE)

6. TRANSMIT AcknowledgeAlarm-Request,

'Acknowledging Process Identifier' = (the value of the 'Process Identifier' parameter in the event notification),  
 'Event Object Identifier' = (the 'Event Object Identifier' from the event notification),  
 'Event State Acknowledged' = (the state specified in the 'To State' parameter of the notification),  
 'Time Stamp' = (any valid time stamp),  
 'Acknowledgment Source' = (any valid value)  
 'Time of Acknowledgment' = (any of the forms specified for this parameter)

7. RECEIVE BACnet-SimpleACK-PDU

8. BEFORE **Notification Fail Time**

RECEIVE UnconfirmedEventNotification-Request,

'Process Identifier' = (the process identifier configured for this event),  
 'Initiating Device Identifier' = IUT,  
 'Event Object Identifier' = (the event-initiating object),  
 'Time Stamp' = (T2: any valid time stamp),  
 'Notification Class' = (the notification class configured for this event),  
 'Priority' = (the priority configured for this event),  
 'Event Type' = (E1),  
 'Message Text' = (optional, any valid message text),  
 'Notify Type' = ACK\_NOTIFICATION,  
 'To State' = (S1)

9. RECEIVE

DESTINATION = (a device other than the TD),

SOURCE = IUT,

UnconfirmedEventNotification-Request,

'Process Identifier' = (the process identifier configured for this event),  
 'Initiating Device Identifier' = IUT,  
 'Event Object Identifier' = (the event-initiating object),  
 'Time Stamp' = (T2),  
 'Notification Class' = (the notification class configured for this event),  
 'Priority' = (the priority configured for this event),  
 'Event Type' = (E1),  
 'Message Text' = (optional, any valid message text),  
 'Notify Type' = ACK\_NOTIFICATION,  
 'To State' = (S1)

10. VERIFY (the 'Event Object Identifier' from the event notification), Acked\_Transitions = (TRUE, TRUE, TRUE)

11. TRANSMIT AcknowledgeAlarm-Request,

'Acknowledging Process Identifier' = (the value of the 'Process Identifier' parameter in the event notification),  
 'Event Object Identifier' = (the 'Event Object Identifier' from the event notification),  
 'Event State Acknowledged' = (the state specified in the 'To State' parameter of the notification),  
 'Time Stamp' = (the time stamp conveyed in the notification),  
 'Acknowledgment Source' = (any valid value)  
 'Time of Acknowledgment' = (any of the forms specified for this parameter)

12. RECEIVE BACnet-SimpleACK-PDU

13. BEFORE **Notification Fail Time**

RECEIVE UnconfirmedEventNotification-Request,  
 'Process Identifier' = (the process identifier configured for this event),  
 'Initiating Device Identifier' = IUT,  
 'Event Object Identifier' = (the event-initiating object),  
 'Time Stamp' = (T3: any valid time stamp),  
 'Notification Class' = (the notification class configured for this event),  
 'Priority' = (the priority configured for this event),  
 'Event Type' = (E1),  
 'Message Text' = (optional, any valid message text),  
 'Notify Type' = ACK\_NOTIFICATION,  
 'To State' = (S1)

#### 14. RECEIVE

DESTINATION = (a device other than the TD),  
 SOURCE = IUT,  
 UnconfirmedEventNotification-Request,  
 'Process Identifier' = (the process identifier configured for this event),  
 'Initiating Device Identifier' = IUT,  
 'Event Object Identifier' = (the event-initiating object),  
 'Time Stamp' = (T3),  
 'Notification Class' = (the notification class configured for this event),  
 'Priority' = (the priority configured for this event),  
 'Event Type' = (E1),  
 'Notify Type' = ACK\_NOTIFICATION,  
 'To State' = (S1)

#### 15. VERIFY (the 'Event Object Identifier' from the event notification), Acked\_Transitions = (TRUE, TRUE, TRUE)

~~Notes to Tester: The destination address used for the acknowledgment notification in steps 9 and 14 shall be the same address used in step 4. When multiple event notifications are expected for a specific event, the order that the IUT transmits them in is irrelevant. If the IUT can only be configured with one recipient in the Recipient\_List property of the issuing Notification\_class object, omit steps 4, 9, and 14.~~

[Change **Clause 9.1.2.1**, p. 320]

[Reviewer Note: Made changes to include not supported transitions.]

#### 9.1.2.1 Unsuccessful Alarm Acknowledgment of Confirmed Event Notifications Because the 'Time Stamp' is Too Old

Purpose: To verify that an alarm remains unacknowledged if the time stamp in the acknowledgment does not match the most recent transition to the current alarm state.

Test Concept: An alarm is triggered that causes the IUT to notify the TD and one other device. The TD acknowledges the alarm using an old time stamp and verifies that the acknowledgment is not accepted by the IUT and that the IUT does not notify other devices that the alarm was acknowledged. The TD then acknowledges the alarm using the proper time stamp and verifies that the acknowledgment is properly noted by the IUT. The IUT notifies all other recipients that the alarm was acknowledged.

Configuration Requirements: The IUT shall be configured with at least one object that can detect alarm conditions and send confirmed notifications. The Acked\_Transitions property shall have the value B'111' indicating that all transitions have been acknowledged. The TD and one other BACnet device, if the IUT supports multiple recipients, shall be recipients of the alarm notification. *D1 is either the pTimeDelay, or pTimeDelayNormal parameter, or 0 (for transitions to and from FAULT state) depending on the event transition.*

*Notes to Tester: The destination address used for the acknowledgment notification in step 11 shall be the same address used in step 3. If the IUT can only be configured with one recipient in the Recipient\_List property of the issuing Notification\_class object, omit steps 5, 6, 15, and 16.*

Test Steps:

1. MAKE (a change that triggers the detection of an alarm event in the IUT)
2. WAIT (pTimeDelayD1)



### 3. BEFORE Notification Fail Time

RECEIVE ConfirmedEventNotification-Request,

'Process Identifier' = (the process identifier configured for this event),  
 'Initiating Device Identifier' = IUT,  
 'Event Object Identifier' = (the object detecting the alarm),  
 'Time Stamp' = (T1: any valid time stamp),  
 'Notification Class' = (the notification class configured for this event),  
 'Priority' = (the priority configured for this event type),  
 'Event Type' = (E1: any valid event type),  
 'Message Type' = (optional, any valid message text),  
 'Notify Type' = (the notify type configured for the event),  
 'AckRequired' = TRUE,  
 'From State' = ~~NORMAL~~(any appropriate event state),  
 'To State' = (S1: any appropriate ~~non-normal~~ event state),  
 'Event Values' = (the values appropriate to the event type)

### 4. TRANSMIT BACnet-SimpleACK-PDU

### 5. RECEIVE

DESTINATION = (a device other than the TD),

SOURCE = IUT,

ConfirmedEventNotification-Request,

'Process Identifier' = (the process identifier configured for this event),  
 'Initiating Device Identifier' = IUT,  
 'Event Object Identifier' = (the object detecting the alarm),  
 'Time Stamp' = (T1),  
 'Notification Class' = (the notification class configured for this event),  
 'Priority' = (the priority configured for this event type),  
 'Event Type' = (E2: any valid event type),  
 'Message Type' = (optional, any valid message text),  
 'Notify Type' = (the notify type configured for the event),  
 'AckRequired' = TRUE,  
 'From State' = ~~NORMAL~~(any appropriate event state),  
 'To State' = (S2: any appropriate ~~non-normal~~ event state),  
 'Event Values' = (the values appropriate to the event type)

### 6. TRANSMIT BACnet-SimpleACK-PDU

7. VERIFY (the 'Event Object Identifier' from the event notification), Acked\_Transitions = (FALSE,TRUE,TRUE)(appropriate bit FALSE, the others TRUE)

### 8. TRANSMIT AcknowledgeAlarm-Request,

'Acknowledging Process Identifier' = (the value of the 'Process Identifier' parameter in the event notification),  
 'Event Object Identifier' = (the 'Event Object Identifier' from the event notification),  
 'Event State Acknowledged' = (the state specified in the 'To State' parameter of the notification),  
 'Time Stamp' = (any valid time stamp),  
 'Acknowledgement Source' = (any valid value),  
 'Time of Acknowledgment' = (the current time using a Time format)

### 9. RECEIVE BACnet-Error-PDU

Error Class = SERVICES,

Error Code = INVALID\_TIME\_STAMP

10. VERIFY (the 'Event Object Identifier' from the event notification), Acked\_Transitions = (FALSE,TRUE,TRUE)

### 11. TRANSMIT AcknowledgeAlarm-Request,

'Acknowledging Process Identifier' = (the process identifier configured for this event),  
 'Event Object Identifier' = (the 'Event Object Identifier' from the event notification),  
 'Event State Acknowledged' = (the state specified in the 'To State' parameter of the notification),  
 'Time Stamp' = (the time stamp conveyed in the notification),  
 'Time of Acknowledgment' = (the current time using a Time format)

### 12. RECEIVE ~~BACnet-SimpleACK-PDU~~ BACnet-SimpleACK-PDU

### 13. IF (Protocol\_Revision is present and Protocol\_Revision ≥ 1) THEN

BEFORE Notification Fail Time

RECEIVE

ConfirmedEventNotification-Request,

'Process Identifier' = (the process identifier configured for this event),  
 'Initiating Device Identifier' = IUT,

'Event Object Identifier' = (the object detecting the alarm),  
 'Time Stamp' = (T2: any valid time stamp),  
 'Notification Class' = (the notification class configured for this event),  
 'Priority' = (the priority configured for this event type),  
 'Event Type' = (any valid event type),  
 'Message Type' = (optional, any valid message text),  
 'Notify Type' = ACK\_NOTIFICATION,  
 'To State' = (S1 or S2)

ELSE

BEFORE Notification Fail Time

RECEIVE

ConfirmedEventNotification-Request,  
 'Process Identifier' = (the process identifier configured for this event),  
 'Initiating Device Identifier' = IUT,  
 'Event Object Identifier' = (the object detecting the alarm),  
 'Time Stamp' = (T2: any valid time stamp),  
 'Notification Class' = (the notification class configured for this event),  
 'Priority' = (the priority configured for this event type),  
 'Event Type' = (any valid event type),  
 'Message Type' = (optional, any valid message text),  
 'Notify Type' = ACK\_NOTIFICATION

14. TRANSMIT BACnet-SimpleACK-PDU

15. IF (Protocol\_Revision is present and Protocol\_Revision ≥ 1) THEN

RECEIVE

DESTINATION = (a device other than the TD),  
 SOURCE = IUT,  
 ConfirmedEventNotification-Request,  
 'Process Identifier' = (the process identifier configured for this event),  
 'Initiating Device Identifier' = IUT,  
 'Event Object Identifier' = (the object detecting the alarm),  
 'Time Stamp' = (T2),  
 'Notification Class' = (the notification class configured for this event),  
 'Priority' = (the priority configured for this event type),  
 'Event Type' = (any valid event type),  
 'Notify Type' = ACK\_NOTIFICATION,  
 'To State' = (S1 or S2)

ELSE

RECEIVE

DESTINATION = (a device other than the TD),  
 SOURCE = IUT,  
 ConfirmedEventNotification-Request,  
 'Process Identifier' = (the process identifier configured for this event),  
 'Initiating Device Identifier' = IUT,  
 'Event Object Identifier' = (the object detecting the alarm),  
 'Time Stamp' = (T2),  
 'Notification Class' = (the notification class configured for this event),  
 'Priority' = (the priority configured for this event type),  
 'Event Type' = (any valid event type),  
 'Message Type' = (optional, any valid message text),  
 'Notify Type' = ACK\_NOTIFICATION

16. TRANSMIT BACnet-SimpleACK-PDU

17. VERIFY (the 'Event Object Identifier' from the event notification), Acked\_Transitions = (TRUE,TRUE,TRUE)

~~Notes to Tester: The destination address used for the acknowledgment notification in step 11 shall be the same address used in step 3. If the IUT can only be configured with one recipient in the Recipient\_List property of the issuing Notification\_class object, omit steps 5, 6, 15, and 16.~~

[Change **Clause 9.1.2.5**, p. 324]

[Reviewer Note: Made changes to include not supported transitions.]

### 9.1.2.5 Unsuccessful Alarm Acknowledgment of Unconfirmed Event Notifications Because the 'Time Stamp' is Too Old

Purpose: To verify that an alarm remains unacknowledged if the time stamp in the acknowledgment does not match the most recent transition to the current alarm state.

Test Concept: An alarm is triggered that causes the IUT to notify the TD and one other device. The TD acknowledges the alarm using an old time stamp and verifies that the acknowledgment is not accepted by the IUT and that the IUT does not notify other devices that the alarm was acknowledged. The TD then acknowledges the alarm using the proper time stamp and verifies that the acknowledgment is properly noted by the IUT. The IUT notifies all other recipients that the alarm was acknowledged.

Configuration Requirements: The IUT shall be configured with at least one object that can detect alarm conditions and send unconfirmed notifications. The Acked\_Transitions property shall have the value B'111' indicating that all transitions have been acknowledged. The TD and one other BACnet device, if the IUT supports multiple recipients, shall be recipients of the alarm notification. *D1 is either the pTimeDelay, or pTimeDelayNormal parameter, or 0 (for transitions to and from FAULT state) depending on the event transition.*

Notes to Tester: The destination address used for the acknowledgment notification in step 11 shall be the same address used in step 3. The destination address used for the acknowledgment notification in step 12 shall be the same address used in step 4. When multiple event notifications are expected for a specific event, the order that the IUT transmits them in is irrelevant. If the IUT can only be configured with one recipient in the Recipient\_List property of the issuing Notification\_class object, omit steps 4 and 12.

Test Steps:

1. MAKE (a change that triggers the detection of an alarm event in the IUT)
2. WAIT ~~pTimeDelay~~*D1*
3. BEFORE Notification Fail Time
  - RECEIVE UnconfirmedEventNotification-Request,
    - 'Process Identifier' = (the process identifier configured for this event),
    - 'Initiating Device Identifier' = IUT,
    - 'Event Object Identifier' = (the object detecting the alarm),
    - 'Time Stamp' = (T1: any valid time stamp),
    - 'Notification Class' = (the notification class configured for this event),
    - 'Priority' = (the priority configured for this event type),
    - 'Event Type' = (any valid event type),
    - 'Message Text' = (optional, any valid message text),
    - 'Notify Type' = (the notify type configured for the event),
    - 'AckRequired' = TRUE,
    - 'From State' = ~~NORMAL~~*(any appropriate event state)*,
    - 'To State' = (S1: any appropriate ~~non-normal~~ event state),
    - 'Event Values' = (the values appropriate to the event type)
4. IF (the notification in step 3 was not a broadcast) THEN
  - RECEIVE
    - DESTINATION = (at least one device other than the TD),
    - SOURCE = IUT,
    - UnconfirmedEventNotification-Request,
      - 'Process Identifier' = (the process identifier configured for this event),
      - 'Initiating Device Identifier' = IUT,
      - 'Event Object Identifier' = (the object detecting the alarm),
      - 'Time Stamp' = (T1),
      - 'Notification Class' = (the notification class configured for this event),
      - 'Priority' = (the priority configured for this event type),
      - 'Event Type' = (any valid event type),
      - 'Message Text' = (optional, any valid message text),
      - 'Notify Type' = (the notify type configured for the event),
      - 'AckRequired' = TRUE,
      - 'From State' = ~~NORMAL~~*(any appropriate event state)*,

'To State' = (S2: any appropriate ~~non-normal~~ event state),  
 'Event Values' = (the values appropriate to the event type)

5. VERIFY (the 'Event Object Identifier' from the event notification), Acked\_Transitions = (FALSE,TRUE,TRUE)(appropriate bit FALSE, the others TRUE)

6. TRANSMIT AcknowledgeAlarm-Request,  
 'Acknowledging Process Identifier' = (the value of the 'Process Identifier' parameter in the event notification),  
 'Event Object Identifier' = (the 'Event Object Identifier' from the event notification),  
 'Event State Acknowledged' = (the state specified in the 'To State' parameter of the notification),  
 'Time Stamp' = (a time stamp older than the one conveyed in the notification),  
 'Acknowledgement Source' = (any valid value),  
 'Time of Acknowledgment' = (the TD's current time using a Time format)

7. RECEIVE BACnet-Error-PDU  
 Error Class = SERVICES,  
 Error Code = INVALID\_TIME\_STAMP

8. VERIFY (the 'Event Object Identifier' from the event notification), Acked\_Transitions = (FALSE,TRUE,TRUE)

9. TRANSMIT AcknowledgeAlarm-Request,  
 'Acknowledging Process Identifier' = (the process identifier configured for this event),  
 'Event Object Identifier' = (the 'Event Object Identifier' from the event notification),  
 'Event State Acknowledged' = (the state specified in the 'To State' parameter of the notification),  
 'Time Stamp' = (the time stamp conveyed in the notification),  
 'Acknowledgement Source' = (any valid value),  
 'Time of Acknowledgment' = (the TD's current time using a Time format)

10. RECEIVE ~~BACnet-Simple-ACK-PDU~~ BACnet-SimpleACK-PDU

11. IF (Protocol\_Revision is present and Protocol\_Revision ≥ 1) THEN  
 BEFORE **Notification Fail Time**  
 RECEIVE  
 DESTINATION = LOCAL BROADCAST | GLOBAL BROADCAST | TD,  
 SOURCE = IUT,  
 UnconfirmedEventNotification-Request,  
 'Process Identifier' = (the process identifier configured for this event),  
 'Initiating Device Identifier' = IUT,  
 'Event Object Identifier' = (the object detecting the alarm),  
 'Time Stamp' = (T2: any valid time stamp),  
 'Notification Class' = (the notification class configured for this event),  
 'Priority' = (the priority configured for this event type),  
 'Event Type' = (any valid event type),  
 'Message Text' = (optional, any valid message text),  
 'Notify Type' = ACK\_NOTIFICATION,  
 'To State' = (S1 or S2)

ELSE  
 BEFORE **Notification Fail Time**  
 RECEIVE  
 DESTINATION = LOCAL BROADCAST | GLOBAL BROADCAST | TD,  
 SOURCE = IUT,  
 UnconfirmedEventNotification-Request,  
 'Process Identifier' = (the process identifier configured for this event),  
 'Initiating Device Identifier' = IUT,  
 'Event Object Identifier' = (the object detecting the alarm),  
 'Time Stamp' = (T2: any valid time stamp),  
 'Notification Class' = (the notification class configured for this event),  
 'Priority' = (the priority configured for this event type),  
 'Event Type' = (any valid event type),  
 'Message Text' = (optional, any valid message text),  
 'Notify Type' = ACK\_NOTIFICATION

12. IF (the notification in step 11 was not broadcast) THEN  
 IF (Protocol\_Revision is present AND Protocol\_Revision ≥ 1) THEN  
 RECEIVE  
 DESTINATION = (at least one device other than the TD),  
 SOURCE = IUT,  
 UnconfirmedEventNotification-Request,

'Process Identifier' = (the process identifier configured for this event),  
 'Initiating Device Identifier' = IUT,  
 'Event Object Identifier' = (the object detecting the alarm),  
 'Time Stamp' = (T2),  
 'Notification Class' = (the notification class configured for this event),  
 'Priority' = (the priority configured for this event type),  
 'Event Type' = (any valid event type),  
 'Message Text' = (optional, any valid message text),  
 'Notify Type' = ACK\_NOTIFICATION,  
 'To State' = (S1 or S2)

ELSE

RECEIVE

DESTINATION = (at least one device other than the TD),  
 SOURCE = IUT,  
 UnconfirmedEventNotification-Request,  
 'Process Identifier' = (the process identifier configured for this event),  
 'Initiating Device Identifier' = IUT,  
 'Event Object Identifier' = (the object detecting the alarm),  
 'Time Stamp' = (T2),  
 'Notification Class' = (the notification class configured for this event),  
 'Priority' = (the priority configured for this event type),  
 'Event Type' = (any valid event type),  
 'Message Text' = (optional, any valid message text),  
 'Notify Type' = ACK\_NOTIFICATION

13. VERIFY (the 'Event Object Identifier' from the event notification), Acked\_Transitions = (TRUE,TRUE,TRUE)

~~Notes to Tester: The destination address used for the acknowledgment notification in step 11 shall be the same address used in step 3. The destination address used for the acknowledgment notification in step 12 shall be the same address used in step 4. When multiple event notifications are expected for a specific event, the order that the IUT transmits them in is irrelevant. If the IUT can only be configured with one recipient in the Recipient\_List property of the issuing Notification\_class object, omit steps 4 and 12.~~

## 9.2 ConfirmedCOVNotification Service Execution Tests

### 9.2.1 Positive ConfirmedCOVNotification Service Execution Tests

[Change Clause 9.2.1.1, p. 328]

[Reviewer Note: The existing test did not account for other properties which are expected for certain object types.]

#### 9.2.1.1 Change of Value Notifications

Purpose: To verify that the IUT can execute ConfirmedCOVNotification requests from object types that provide the Present\_Value and Status\_Flags properties in COV notifications.

*Test Concept: The IUT is made to subscribe for COV from an object of the type being tested. The TD then sends a COV notification to the IUT and verifies that the IUT exhibits any actions identified by the vendor.*

Test Steps:

1. RECEIVE SubscribeCOV,  
     'Subscriber Process Identifier' = (any valid process identifier),  
     'Monitored Object Identifier' = X,  
     'Issue Confirmed Notifications' = TRUE,  
     'Lifetime' = (a value 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' = (the time remaining in the subscription),  
 'List of Values' = (Present\_Value~~and~~, Status\_Flags, and *additional properties* appropriate to object type X)

4. RECEIVE BACnet-SimpleACK-PDU

5. CHECK (to ensure that any appropriate functions defined by the manufacturer, such as displaying information on a workstation screen are carried out)

[Add new **Clause 9.2.1.X5**, p. 329]

### 9.2.1.X5 ConfirmedCOVNotification from Access Door Object

Purpose: To verify that the IUT can execute ConfirmedCOVNotification requests from Access Door objects.

Test Steps:

1. RECEIVE SubscribeCOV-Request,  
     'Subscriber Process Identifier' = (any valid process identifier value > 0),  
     'Monitored Object Identifier' = (any Access Door object, X),  
     'Issue Confirmed Notifications' = TRUE,  
     'Lifetime' = (a value greater than one minute)
2. TRANSMIT BACnet-SimpleACK-PDU
3. TRANSMIT ConfirmedCOVNotification-Request,  
     'Subscriber Process Identifier' = (the process identifier used in step 1),  
     'Initiating Device Identifier' = TD,  
     'Monitored Object Identifier' = X,  
     'Time Remaining' = (the time remaining in the subscription),  
     'List of Values' = (the initial Present\_Value, initial Status\_Flags, and Door\_Alarm\_State if X has a Door\_Alarm\_State property)
4. RECEIVE BACnet-SimpleACK-PDU
5. CHECK (to ensure that any appropriate functions defined by the manufacturer, such as displaying information on a workstation screen are carried out)

[Add new **Clause 9.2.1.X6**, p. 329]

### 9.2.1.X6 ConfirmedCOVNotification from Access Point Object

Purpose: To verify that the IUT can execute ConfirmedCOVNotification requests from Access Point objects.

Test Steps:

1. RECEIVE SubscribeCOV-Request,  
     'Subscriber Process Identifier' = (PI: any valid process identifier value > 0),  
     'Monitored Object Identifier' = (X: any Access Point object),  
     'Issue Confirmed Notifications' = TRUE,  
     'Lifetime' = (a value greater than one minute)
2. TRANSMIT BACnet-SimpleACK-PDU
3. TRANSMIT ConfirmedCOVNotification-Request,  
     'Subscriber Process Identifier' = PI,  
     'Initiating Device Identifier' = TD,  
     'Monitored Object Identifier' = X,  
     'Time Remaining' = (the time remaining in the subscription),  
     'List of Values' = (any valid set of values)
4. RECEIVE BACnet-SimpleACK-PDU
5. CHECK (to ensure that any appropriate functions defined by the manufacturer, such as displaying information on a workstation screen are carried out)

[Add new **Clause 9.2.1.X7**, p. 329]

### 9.2.1.X7 ConfirmedCOVNotification from Credential Data Input Object

Purpose: To verify that the IUT can execute ConfirmedCOVNotification requests from Credential Data Input objects.

Test Steps:

1. RECEIVE SubscribeCOV-Request,  
     'Subscriber Process Identifier' = (PI: any valid process identifier value > 0),  
     'Monitored Object Identifier' = (X: any Credential Data Input object),  
     'Issue Confirmed Notifications' = TRUE,  
     'Lifetime' = (a value greater than one minute)
2. TRANSMIT BACnet-SimpleACK-PDU
3. TRANSMIT ConfirmedCOVNotification-Request,  
     'Subscriber Process Identifier' = PI,  
     'Initiating Device Identifier' = TD,  
     'Monitored Object Identifier' = X,  
     'Time Remaining' = (the time remaining in the subscription),  
     'List of Values' = (any valid set of values)
4. RECEIVE BACnet-SimpleACK-PDU
5. CHECK (to ensure that any appropriate functions defined by the manufacturer, such as displaying information on a workstation screen are carried out)

### 9.2.2 Negative ConfirmedCOVNotification Service Execution Tests

[Change **Clause 9.2.2.1**, p. 329]

[Reviewer Note: Corrected tests per BTL-CR-0299 and added Configuration Requirements section.]

#### 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.

*Configuration Requirements: If the IUT does not support initiation of SubscribeCOV-Request with 'Issue Confirmed Notifications' equal to TRUE, then this test shall be skipped.*

Test Steps:

1. RECEIVE SubscribeCOV-Request,  
     'Subscriber Process Identifier' = (any valid process identifier, PI),  
     '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)~~ (any valid Lifetime)
2. TRANSMIT BACnet-SimpleACK-PDU
3. TRANSMIT ConfirmedCOVNotification-Request,  
     'Subscriber Process Identifier' = ~~(the process identifier used in step 1, PI)~~,  
     '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 (the IUT can cancel the subscription) THEN  
     RECEIVE SubscribeCOV-Request,  
     'Subscriber Process Identifier' = (PI),

```

        'Monitored Object Identifier' = X
    ELSE
        MAKE (the IUT stop resubscribing, if it resubscribes automatically)
53. WAIT (a value two times at least Lifetime, but sufficient to ensure the subscription has expired)
64. TRANSMIT ConfirmedCOVNotification-Request,
    'Subscriber Process Identifier' = (the process identifier used in step 2P1),
    '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)
75. 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)

```

[Change **Clause 9.2.2.2**, p. 330]

[Reviewer Note: 135-2008h allows for a SimpleAck or a specific error code to return if a subscription does not exist.]

#### 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.

*Configuration Requirements: If the IUT does not support initiation of SubscribeCOV-Request with 'Issue Confirmed Notifications' equal to TRUE, then this test shall be skipped.*

Test Steps:

```

1. RECEIVE SubscribeCOV-Request,
    '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 minuteany valid Lifetime)
2. TRANSMIT BACnet-SimpleACK-PDU
3. TRANSMIT ConfirmedCOVNotification-Request,
    'Subscriber Process Identifier' = (a process identifier different from the one used in step 2I),
    '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)

```



[Change **Clause 9.2.2.4**, p. 331]

[Reviewer Note: 135-2008h allows for a SimpleAck or a specific error code to return if a subscription does not exist.]

### 9.2.2.4 Change of Value Notifications with Invalid Monitored Object Identifier

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

*Configuration Requirements: If the IUT does not support initiation of SubscribeCOV-Request with 'Issue Confirmed Notifications' equal to TRUE, then this test shall be skipped.*

*Notes to Tester: If possible, select an object Y for which IUT supports COV Subscription.*

Test Steps:

1. RECEIVE SubscribeCOV-Request,  
     '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)~~ any valid Lifetime)
2. TRANSMIT BACnet-SimpleACK-PDU
3. TRANSMIT ConfirmedCOVNotification-Request,  
     'Subscriber Process Identifier' = (the process identifier used in step 21),  
     'Initiating Device Identifier' = TD,  
     'Monitored Object Identifier' = (any object Y in the IUT ~~supporting COV notification~~ except X, and for which IUT does not already have an active subscription),  
     'Time Remaining' = (any amount of time greater than 0),  
     'List of Values' = (a list of values appropriate to object Y)
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)

## 9.3 UnconfirmedCOVNotification Service Execution Tests

[Add new **Clause 9.3.X6**, p. 333]

### 9.3.X6 UnconfirmedCOVNotification from Access Door Object

Purpose: To verify that the IUT can execute UnconfirmedCOVNotification requests from Access Door objects.

Test Steps:

1. RECEIVE SubscribeCOV-Request,  
     'Subscriber Process Identifier' = (any valid process identifier value > 0),  
     'Monitored Object Identifier' = (any Access Door object, X),  
     'Issue Confirmed Notifications' = FALSE,  
     'Lifetime' = (a value greater than one minute)
2. TRANSMIT BACnet-SimpleACK-PDU

3. TRANSMIT UnconfirmedCOVNotification-Request,  
     'Subscriber Process Identifier' = (the process identifier used in step 1),  
     'Initiating Device Identifier' = TD,  
     'Monitored Object Identifier' = X,  
     'Time Remaining' = (the time remaining in the subscription),  
     'List of Values' = (the initial Present\_Value, initial Status\_Flags, and  
                             Door\_Alarm\_State if X has a Door\_Alarm\_State property)
4. CHECK (to ensure that any appropriate functions defined by the manufacturer, such as displaying information on a workstation screen are carried out)

[Add new **Clause 9.3.X7**, p. 333]

### **9.3.X7 UnconfirmedCOVNotification from Access Point Object**

Purpose: To verify that the IUT can execute UnconfirmedCOVNotification requests from Access Point objects.

Test Steps:

1. RECEIVE SubscribeCOV-Request,  
     'Subscriber Process Identifier' = (PI: any valid process identifier value > 0),  
     'Monitored Object Identifier' = (X: any Access Door object),  
     'Issue Confirmed Notifications' = FALSE,  
     'Lifetime' = (a value greater than one minute)
2. TRANSMIT BACnet-SimpleACK-PDU
3. TRANSMIT UnconfirmedCOVNotification-Request,  
     'Subscriber Process Identifier' = PI,  
     'Initiating Device Identifier' = TD,  
     'Monitored Object Identifier' = X,  
     'Time Remaining' = (the time remaining in the subscription),  
     'List of Values' = (any valid set of values)
4. CHECK (to ensure that any appropriate functions defined by the manufacturer, such as displaying information on a workstation screen are carried out)

[Add new **Clause 9.3.X8**, p. 333]

### **9.3.X8 UnconfirmedCOVNotification from Credential Data Input Object**

Purpose: To verify that the IUT can execute UnconfirmedCOVNotification requests from Credential Data Input objects.

Test Steps:

1. RECEIVE SubscribeCOV-Request,  
     'Subscriber Process Identifier' = (PI: any valid process identifier value > 0),  
     'Monitored Object Identifier' = (X: any Access Door object),  
     'Issue Confirmed Notifications' = FALSE,  
     'Lifetime' = (a value greater than one minute)
2. TRANSMIT BACnet-SimpleACK-PDU
3. TRANSMIT UnconfirmedCOVNotification-Request,  
     'Subscriber Process Identifier' = PI,  
     'Initiating Device Identifier' = TD,  
     'Monitored Object Identifier' = X,  
     'Time Remaining' = (the time remaining in the subscription),  
     'List of Values' = (any valid set of values)
4. CHECK (to ensure that any appropriate functions defined by the manufacturer, such as displaying information on a workstation screen are carried out)

[Change **Clause 9.3.2**, p. 332]

[Reviewer Note: The existing test did not account for other properties which are expected for certain object types.]

### 9.3.2 Change of Value Notifications

Purpose: To verify that the IUT can execute UnconfirmedCOVNotification requests from objects that provide the Present\_Value and Status\_Flags properties in COV notifications.

*Test Concept: The IUT is made to subscribes for COV from an object of the type being tested. The TD then sends a COV notification to the IUT and verifies that the IUT exhibits any actions identified by the vendor.*

Test Steps:

1. RECEIVE SubscribeCOV,  
     'Subscriber Process Identifier' = (any valid process identifier),  
     'Monitored Object Identifier' = X,  
     'Issue Confirmed Notifications' = FALSE,  
     'Lifetime' = (a value greater than 1 minute)
2. TRANSMIT BACnet-SimpleACK-PDU
3. TRANSMIT UnconfirmedCOVNotification-Request,  
     'Subscriber Process Identifier' = (the process identifier used in step 2),  
     'Initiating Device Identifier' = TD,  
     'Monitored Object Identifier' = X,  
     'Time Remaining' = (the time remaining for the subscription),  
     'List of Values' = (Present\_Value, and Status\_Flags, and additional properties appropriate to object type X)
4. CHECK (to ensure that any appropriate functions defined by the manufacturer, such as displaying information on a workstation screen are carried out)

## 9.4 ConfirmedEventNotification Service Execution Tests

[Add new **Clause 9.4.X2**, p. 336]

[Reviewer Note: Added a test for receiving a BACnetPropertyStates value which is encoded using an extended-value.]

### 9.4.X2 Decoding BACnetPropertyStates in 'Event Values'

Purpose: To verify that the IUT is correctly accepting 'Event Values', which contain BACnetPropertyStates values across the full value range of context tags, ensuring proper decoding of the various forms of the production.

*Test Concept: Send 3 ConfirmedEventNotifications to the IUT conveying a CHANGE\_OF\_STATE event. After each notification verify that the IUT accepts and processes the notification. The first notification is sent with a new-state, NS1, having a context tag value in the range 0 .. 62. The second notification is sent with a new-state, NS2, having a context tag value in the range 64 .. 253 (a vendor proprietary discrete datatype). The third notification is sent with a new-state, NS3, having a context tag value 254 (a standard discrete datatype) or greater and encoded with a context tag of 63 (the extended-value choice) using the special encoding rules defined in the comment at the end of the BACnetPropertyStates production in Clause 21.*

Test Steps:

- new-state with a tag value in the range 15 .. 62
1. TRANSMIT ConfirmedEventNotification-Request,  
     'Process Identifier' = (any valid process identifier),  
     'Initiating Device Identifier' = TD,  
     'Event Object Identifier' = (any valid value),  
     'Time Stamp' = (any valid time stamp),  
     'Notification Class' = (any valid value),  
     'Priority' = (any valid value),

```

'Event Type' = CHANGE_OF_STATE,
'Message Text' = (optional, any valid message text),
'Notify Type' = ALARM | EVENT,
'AckRequired' = TRUE | FALSE,
'From State' = (any valid normal or offnormal value),
'To State' = (any valid normal or offnormal value),
'Event Values' = (NS1, (T,F,?,?))
2. RECEIVE BACnet-SimpleACK-PDU
3. CHECK (that the IUT has utilized the value conveyed, correctly decoded)

-- new-state with a tag value in the range 64 .. 253
4. TRANSMIT ConfirmedEventNotification-Request,
    'Process Identifier' = (any valid process identifier),
    'Initiating Device Identifier' = TD,
    'Event Object Identifier' = (any valid value),
    'Time Stamp' = (any valid time stamp),
    'Notification Class' = (any valid value),
    'Priority' = (any valid value),
    'Event Type' = CHANGE_OF_STATE,
    'Message Text' = (optional, any valid message text),
    'Notify Type' = ALARM | EVENT,
    'AckRequired' = TRUE | FALSE,
    'From State' = (any valid normal or offnormal value),
    'To State' = (any valid normal or offnormal value),
    'Event Values' = (NS2, (T,F,?,?))
5. RECEIVE BACnet-SimpleACK-PDU
6. CHECK (that the IUT has correctly decoded the value by examining exposed actions related to the receipt of
    the event notification, if there are utilized the value conveyed, correctly decoded)

-- new-state with a tag value greater than 254,
7. TRANSMIT ConfirmedEventNotification-Request,
    'Process Identifier' = (any valid process identifier),
    'Initiating Device Identifier' = TD,
    'Event Object Identifier' = (any valid value),
    'Time Stamp' = (any valid time stamp),
    'Notification Class' = (any valid value),
    'Priority' = (any valid value),
    'Event Type' = CHANGE_OF_STATE,
    'Message Text' = (optional, any valid message text),
    'Notify Type' = ALARM | EVENT,
    'AckRequired' = TRUE | FALSE,
    'From State' = (any valid normal or offnormal value),
    'To State' = (any valid normal or offnormal value),
    'Event Values' = (NS3, (T,F,?,?))
8. RECEIVE BACnet-SimpleACK-PDU
9. CHECK (that the IUT has correctly decoded the value by examining exposed actions related to the receipt of
    the event notification, if there are any)

```

## 9.5 UnconfirmedEventNotification Service Execution Tests

[Add new **Clause 9.5.X2**, p. 337]

[Reviewer Note: Added a test for receiving a BACnetPropertyStates value which is encoded using an extended-value.]

## 9.5.X2 Decoding BACnetPropertyStates in 'Event Values'

Purpose: To verify that the IUT is correctly accepting 'Event Values', which contain BACnetPropertyStates values across the full value range of context tags, ensuring proper decoding of the various forms of the production.

Test Concept: Send 3 UnconfirmedEventNotifications to the IUT conveying a CHANGE\_OF\_STATE event. After each notification verify that the IUT accepts and processes the notification. The first notification is sent with a new-state, NS1, having a context tag value in the range 0 .. 62. The second notification is sent with a new-state, NS2, having a context tag value in the range 64 .. 253 (a vendor proprietary discrete datatype). The third notification is sent with a new-state, NS3, having a context tag value 254 (a standard discrete datatype) or greater and encoded with a context tag of 63 (the extended-value choice) using the special encoding rules defined in the comment at the end of the BACnetPropertyStates production in Clause 21.

Test Steps:

-- new-state with a tag value in the range 15 .. 62

1. TRANSMIT UnconfirmedEventNotification-Request,
  - 'Process Identifier' = (any valid process identifier),
  - 'Initiating Device Identifier' = TD,
  - 'Event Object Identifier' = (any valid value),
  - 'Time Stamp' = (any valid time stamp),
  - 'Notification Class' = (any valid value),
  - 'Priority' = (any valid value),
  - 'Event Type' = CHANGE\_OF\_STATE,
  - 'Message Text' = (optional, any valid message text),
  - 'Notify Type' = ALARM | EVENT,
  - 'AckRequired' = TRUE | FALSE,
  - 'From State' = (any valid normal or offnormal value),
  - 'To State' = (any valid normal or offnormal value),
  - 'Event Values' = (NS1, (T,F,?,?))
2. CHECK (that the IUT has correctly decoded the value by examining exposed actions related to the receipt of the event notification, if there are any)

-- new-state with a tag value in the range 64 .. 253

3. TRANSMIT UnconfirmedEventNotification-Request,
  - 'Process Identifier' = (any valid process identifier),
  - 'Initiating Device Identifier' = TD,
  - 'Event Object Identifier' = (any valid value),
  - 'Time Stamp' = (any valid time stamp),
  - 'Notification Class' = (any valid value),
  - 'Priority' = (any valid value),
  - 'Event Type' = CHANGE\_OF\_STATE,
  - 'Message Text' = (optional, any valid message text),
  - 'Notify Type' = ALARM | EVENT,
  - 'AckRequired' = TRUE | FALSE,
  - 'From State' = (any valid normal or offnormal value),
  - 'To State' = (any valid normal or offnormal value),
  - 'Event Values' = (NS2, (T,F,?,?))
4. CHECK (that the IUT has correctly decoded the value by examining exposed actions related to the receipt of the event notification, if there are any)

-- new-state with a tag value greater than 254,

5. TRANSMIT UnconfirmedEventNotification-Request,
  - 'Process Identifier' = (any valid process identifier),
  - 'Initiating Device Identifier' = TD,
  - 'Event Object Identifier' = (any valid value),
  - 'Time Stamp' = (any valid time stamp),
  - 'Notification Class' = (any valid value),
  - 'Priority' = (any valid value),
  - 'Event Type' = CHANGE\_OF\_STATE,
  - 'Message Text' = (optional, any valid message text),
  - 'Notify Type' = ALARM | EVENT,

'AckRequired' = TRUE | FALSE,  
 'From State' = (any valid normal or offnormal value),  
 'To State' = (any valid normal or offnormal value),  
 'Event Values' = (NS3, (T,F,?,?))

9. CHECK (that the IUT has correctly decoded the value by examining exposed actions related to the receipt of the event notification, if there are any)

## 9.9 LifeSafetyOperation Service Execution Test

[Add new **Clause 9.9.1**, p. 345]

### 9.9.1 Positive LifeSafetyOperation Execution Tests

[Change and renumber **Clause 9.9.1**, p. 345]

[Reviewer Note: Modified to not rely on Event\_State so it can be applied to objects which do not support event reporting.]

#### ~~9.9.19.9.1.1~~ Reset Single Object Execution Tests

~~BACnet Reference Clause: 13.13.~~

Purpose: To verify that the IUT *correctly resets a latched life safety object when executing* ~~can correctly execute a~~ LifeSafetyOperation service request ~~to a single directed at the Life Safety Object-life safety object.~~

Test Concept: A ~~Life-Safety~~ life safety object, which latches its Present\_Value, is toggled between a ~~NORMAL and OFFNORMAL state~~ normal and non-normal BACnetLifeSafetyState. The STATUS object's Present\_Value should latch in the ~~OFFNORMAL non-normal~~ state until the LifeSafetyOperation service request is ~~transmitted~~ executed. This is repeated for each LifeSafetyOperation request type supported by the device. ~~This test may be omitted when the device does not support latching.~~

Configuration Requirements: ~~The IUT must support a Life Safety Object that supports alarming. This object is configured into the NORMAL state, driven to the OFFNORMAL state and returned to the NORMAL state by adjusting its Present\_Value. The STATUS should remain latched in the OFFNORMAL or FAULT state until the reset is issued. This test must be repeated for each reset function supported. The Life-Safety object starts the test with its Present\_Value in a normal state as interpreted by the life safety object. This test shall be skipped if the device does not support latching.~~

Test Steps:

1. REPEAT X = (All supported enumerations that reset the object) DO {
2. MAKE (the selected object ~~enter~~ enters a latched non-normal state where enumeration X will reset the object)
3. VERIFY Present\_Value = (S1: a non-NORMAL state)
4. VERIFY Tracking\_Value = S1
5. MAKE (~~Event\_State = NORMAL~~ remove the non-normal condition)
6. VERIFY Present\_Value = S1
7. VERIFY Tracking\_Value = (S2: a NORMAL state)
- IF (enumeration value creates an Event\_State = FAULT) THEN
- CHECK (Event\_State = FAULT)
- ELSE
- CHECK (Event\_State = OFFNORMAL)
67. TRANSMIT LifeSafetyOperation-Request,  
     'Requesting Process Identifier' = (any valid identifier),  
     'Requesting Source' = (any valid character string),  
     'Request' = X, (~~any valid LifeSafetyOperation request~~),  
     'Object Identifier' = (the selected object)
78. RECEIVE BACnet-SimpleACK-PDU
88. VERIFY (~~Object~~), STATUS Present\_Value = S2 NORMAL)
- }

[Change and renumber **Clause 9.9.2**, p. 345]

[Reviewer Note: The test is generalized to work for any number of objects. The test is written to not rely on Event\_State so it can be applied to objects which do not support event reporting.]

### **9.9.29.9.1.2 Reset Multiple Object Execution Tests**

~~BACnet Reference Clause: 13.13.~~

Purpose: To verify that the IUT *correctly resets multiple latched life safety objects when executing* ~~can correctly execute a LifeSafetyOperation service request to multiple Life Safety objects not directed at a specified life safety object.~~

Test Concept: ~~Two Multiple Life Safety objects, O1 ... On, are toggled between a NORMAL and an OFFNORMAL state normal and non-normal BACnetLifeSafetyState. The STATUS objects' Present\_Value properties should latch in the OFFNORMAL non-normal state until the LifeSafetyOperation service request is transmitted executed. This is repeated for each LifeSafetyOperation request type supported by the device. This test may be omitted when the device does not support latching.~~

Configuration Requirements: ~~The IUT must support a Life Safety Object that supports alarming and have at least two objects that can be latched. This object is configured into the NORMAL state, driven to the OFFNORMAL state or FAULT and returned to the NORMAL state by adjusting its Present\_Value. The STATUS should remain latched in the OFFNORMAL state until the reset is issued. This test must be repeated for each reset function supported. The Life-Safety objects start the test with their Present\_Value properties in a normal state as interpreted by the life safety objects. This test shall be skipped if the device does not support latching Life Safety objects. If the IUT supports a single latching life safety object, apply this test to the single object.~~

Test Steps:

1. REPEAT X = (All supported enumerations that reset the *objects object*) DO {
2. MAKE (the selected objects enter a latched *non-normal state* states where enumeration X will reset the objects)
3. REPEAT Oi = (each selected life safety object) {  
VERIFIY Oi, Present\_Value = (Si: a non-normal value)  
VERIFIY Oi, Tracking\_Value = Si  
}
4. MAKE (~~Event\_State = NORMAL~~ remove the non-normal conditions for both objects)
5. REPEAT Oi = (each selected life safety object) {  
VERIFIY Oi, Present\_Value = Si  
VERIFIY Oi, Tracking\_Value = TVi  
}
- ~~IF (enumeration value creates an Event\_State = FAULT) THEN~~  
~~CHECK (Event\_State = FAULT)~~  
~~ELSE~~  
~~CHECK (Event\_State = OFFNORMAL)~~
6. TRANSMIT LifeSafetyOperation-Request,  
'Requesting Process Identifier' = (any valid identifier),  
'Requesting Source' = (any valid character string),  
'Request' = X, (~~any valid LifeSafetyOperation request~~);
7. RECEIVE BACnet-SimpleACK-PDU
- ~~7. VERIFIY (Object), STATUS = NORMAL~~
8. REPEAT Oi = (each selected life safety object) {  
VERIFIY Oi, Present\_Value = TVi  
}

[Change and renumber **Clause 9.9.3**, p. 346]

[Reviewer Note: The test is generalized to work for any number of objects. The test is written to not rely on Event\_State so it can be applied to objects which do not support event reporting.]

### 9.9.39.1.3 Silencing/Unsilencing Execution Tests

~~BACnet Reference Clause: 13.13.~~

Purpose: To verify that the IUT can correctly execute a LifeSafetyOperation service request to silence *and unsilence* an alarming device.

Test Concept: An audible device and/or visual device is attached to the IUT and is sounding/flashing *because a life safety object has entered a non-normal state and the property Silenced is UNSILENCED*. A LifeSafetyOperation service request is transmitted to silence the sounder/strobe. *Then, the life safety object remains in the non-normal state with Silenced equal to SILENCED. A LifeSafetyOperation service request is transmitted to unsilence the sounder/strobe (reactivate it), and it is verified that the object is unsilenced.*

*There are different allowable BACnetSilencedState values based on the silence operation performed and the setup of the IUT. In the below tables, N/A marks an operation that is inappropriate for the test with the corresponding IUT setup.*

<b>Only Sounder Attached</b>					
<b>Silence Operation</b>	<b>Allowable Silenced State</b>		<b>Unsilence Operation</b>	<b>Allowable Silenced State</b>	
<b>SILENCE</b>	<b>ALL_SILENCED,  AUDIBLE_SILENCED,  proprietary</b>		<b>UNSILENCE</b>	<b>UNSILENCED,  proprietary</b>	
<b>SILENCE_AUDIBLE</b>	<b>ALL_SILENCED,  AUDIBLE_SILENCED,  proprietary</b>		<b>UNSILENCE_AUDIBLE</b>	<b>UNSILENCED,  proprietary</b>	
<b>SILENCE_VISUAL</b>	<b>N/A</b>		<b>UNSILENCE_VISUAL</b>	<b>N/A</b>	



<b>Only Strobe Attached</b>					
<b><i>Silence Operation</i></b>	<b><i>Allowable State</i></b>	<b><i>Silenced</i></b>	<b><i>Unsilenced Operation</i></b>	<b><i>Allowable State</i></b>	<b><i>Silenced</i></b>
<b><i>SILENCE</i></b>	<b><i>ALL_SILENCED,</i></b> <b><i>VISUAL_SILENCED,</i></b> <b><i>proprietary</i></b>		<b><i>UNSILENCE</i></b>	<b><i>UNSILENCED,</i></b> <b><i>proprietary</i></b>	
<b><i>SILENCE_AUDIBLE</i></b>	<b><i>N/A</i></b>		<b><i>UNSILENCE_AUDIBLE</i></b>	<b><i>N/A</i></b>	
<b><i>SILENCE_VISUAL</i></b>	<b><i>ALL_SILENCED,</i></b> <b><i>VISUAL_SILENCED,</i></b> <b><i>proprietary</i></b>		<b><i>UNSILENCE_VISUAL</i></b>	<b><i>UNSILENCED,</i></b> <b><i>proprietary</i></b>	

<b><i>Sounder And Strobe Attached</i></b>					
<b><i>Silence Operation</i></b>	<b><i>Allowable State</i></b>	<b><i>Silenced</i></b>	<b><i>Unsilenced Operation</i></b>	<b><i>Allowable State</i></b>	<b><i>Silenced</i></b>
<b><i>SILENCE</i></b>	<b><i>ALL_SILENCED,</i></b>		<b><i>UNSILENCE</i></b>	<b><i>UNSILENCED,</i></b>	
	<b><i>proprietary</i></b>			<b><i>proprietary</i></b>	
<b><i>SILENCE_AUDIBLE</i></b>	<b><i>AUDIBLE_SILENCED,</i></b>	<b><i>proprietary</i></b>	<b><i>UNSILENCE_AUDIBLE</i></b>	<b><i>SILENCED_VISUAL,</i></b>	
			<b><i>(all silenced)</i></b>	<b><i>proprietary</i></b>	
			<b><i>UNSILENCE_AUDIBLE</i></b>	<b><i>UNSILENCED,</i></b>	
			<b><i>(audible silenced,</i></b>	<b><i>proprietary</i></b>	
			<b><i>visual active)</i></b>		
			<b><i>UNSILENCE_AUDIBLE</i></b>	<b><i>N/A</i></b>	
			<b><i>(audible active, visual</i></b>		
			<b><i>silenced)</i></b>		
<b><i>SILENCE_VISUAL</i></b>	<b><i>VISUAL_SILENCED,</i></b>	<b><i>proprietary</i></b>	<b><i>UNSILENCE_VISUAL</i></b>	<b><i>SILENCED_AUDIBLE,</i></b>	
			<b><i>(all silenced)</i></b>	<b><i>proprietary</i></b>	
			<b><i>UNSILENCE_VISUAL</i></b>	<b><i>N/A</i></b>	
			<b><i>(audible silenced,</i></b>		
			<b><i>visual active)</i></b>		
			<b><i>UNSILENCE_VISUAL</i></b>	<b><i>UNSILENCED,</i></b>	
			<b><i>(audible active, visual</i></b>	<b><i>proprietary</i></b>	
			<b><i>silenced)</i></b>		

Configuration Requirements: The IUT must be fitted with needed audible and visual equipment.

Notes to Tester: Source object needs to get silence only for configured objects.

Test Steps:

1. REPEAT X = (All supported enumerations that silence the object) DO {
2. MAKE (the selected object enter a state where enumeration X will *silence the sounder/strobe* commence alerts)
3. VERIFY Silenced = (Unsilenced or a proprietary value with a similar semantic)
5. MAKE (Event\_State = NORMAL)
64. TRANSMIT LifeSafetyOperation-Request,  
'Requesting Process Identifier' = (any valid identifier),  
'Requesting Source' = (any valid character string),

- 'Request' = ~~X, (any valid LifeSafetyOperation request);~~  
 'Object Identifier' = ~~(absent or the selected object)~~
75. RECEIVE BACnet-SimpleACK-PDU  
 86. CHECK (that the sounder/strobe is inactive)  
 97. *VERIFY Silenced = (an allowable silenced state based on the IUT setup and operation request X)*  
 108. *TRANSMIT LifeSafetyOperation-Request,*  
     '*Requesting Process Identifier*' = *(any valid identifier),*  
     '*Requesting Source*' = *(any valid character string),*  
     '*Request*' = *(any valid LifeSafetyOperation request),*  
     '*Object Identifier*' = *(the selected object)*  
 9. RECEIVE BACnet-SimpleACK-PDU  
 10. CHECK (the sounder / strobe active again, as appropriate to the operation)  
 11. *VERIFY Silenced = (the appropriate state based on the operation and IUT condition)*  
 }

[Add new **Clause 9.9.2**, p. 345]

[Reviewer Note: The original 9.9.2 test was renumbered to 9.9.1.2.]

## 9.9.2 Negative LifeSafetyOperation Execution Tests

### 9.9.2.1 LifeSafetyOperation for an Object Which Does Not Exist

Purpose: To verify that the IUT correctly responds when a LifeSafetyOperation targets a non-existent object.

Test Concept: Send a LifeSafetyOperation request to the IUT targeting an object that does not exist in the IUT.

Test Steps:

- TRANSMIT LifeSafetyOperation-Request,  
     'Requesting Process Identifier' = (any valid value),  
     'Requesting Source' = (any valid value),  
     'Request' = (any valid LifeSafetyOperation request supported by the device),  
     'Object Identifier' = (any object not contained in the IUT's database)
- RECEIVE BACnet-Error PDU,  
     'Error Class' = OBJECT,  
     'Error Code' = UNKNOWN\_OBJECT

### 9.9.2.2 LifeSafetyOperation which is Invalid given the Object's Current State

Purpose: To verify that the IUT correctly responds when a LifeSafetyOperation's Request value is invalid given the object's current state.

Test Concept: Send a LifeSafetyOperation request, with a Request value that is not valid for the current state, to the IUT.

Configuration Requirements: The life safety object, O1, being tested is placed into a state where Request R, a valid LifeSafetyOperation value which the life safety object would accept in other states, is currently invalid. If there is no such state, request value pair that satisfies this requirement, this test shall be skipped.

Test Steps:

- TRANSMIT LifeSafetyOperation-Request,  
     'Requesting Process Identifier' = (any valid value),  
     'Requesting Source' = (any valid value),  
     'Request' = (R: a request value which is invalid given the life safety object's current state),  
     'Object Identifier' = O1

2. RECEIVE BACnet-Error PDU,  
    'Error Class' = OBJECT,  
    'Error Code' = INVALID\_OPERATION\_IN\_THIS\_STATE

### 9.9.2.3 LifeSafetyOperation On An Object Which Does Not Support It

Purpose: To verify that the IUT correctly responds when a LifeSafetyOperation's is received that targets an object that does not support it.

Test Concept: Send a LifeSafetyOperation request, with an Object Identifier referencing an object in the IUT which does not support life safety operations.

Test Steps:

1. TRANSMIT LifeSafetyOperation-Request,  
    'Requesting Process Identifier' = (any valid value),  
    'Requesting Source' = (any valid value),  
    'Request' = (any valid value normally supported by the IUT),  
    'Object Identifier' = (an object in the IUT which does not support life safety operations)
2. RECEIVE BACnet-Error PDU,  
    'Error Class' = OBJECT,  
    'Error Code' = OPTIONAL\_FUNCTIONALITY\_NOT\_SUPPORTED

## 9.10 SubscribeCOV Service Execution Tests

### 9.10.1 Positive SubscribeCOV Service Execution Tests

[Change **Clause 9.10.1.7**, p. 349]

[Reviewer Note: Updates description of 'Time Remaining' and adds validation that this value counts down as expected.]

#### 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:

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. 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~~ causes 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' = (TR: a value greater than 0 and less than *or equal to* 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' = (TR: a value greater than 0 and less than *or equal to* 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 (a time that should change the 'Time Remaining' and which is less than the lifetime of the subscription)

7. MAKE (a change to the monitored object that causes a COV notification)

8. 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 TR),  
'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 TR),  
'List of Values' = (values appropriate to the object type of the monitored object including the changed value that triggered the notification)

9. WAIT (the lifetime of the subscription)

10. MAKE (a change to the monitored object that would cause a COV notification if there were an active subscription)

11. CHECK (verify that the IUT did not transmit a COV notification message)

[Change **Clause 9.10.1.8**, p. 350]

[Reviewer Note: Modified as per purpose, BACnet Clause 13.14.2, and updated the language as per BACnet 135.1, Clause 6.]

### 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:

1. TRANSMIT SubscribeCOV-Request,  
     'Subscriber Process Identifier' = *PIDI* (any valid process identifier),  
     'Monitored Object Identifier' = *OI* (any object supporting COV notifications),  
     'Issue Confirmed Notifications' = TRUE | FALSE,  
     'Lifetime' = 60
2. RECEIVE BACnet-SimpleACK-PDU
3. IF (the subscription was for confirmed notifications) THEN  
     **BEFORE Notification Fail Time**  
     RECEIVE ConfirmedCOVNotification-Request,  
     'Subscriber Process Identifier' = *PIDI* (the same identifier used in the subscription),  
     'Initiating Device Identifier' = IUT,  
     'Monitored Object Identifier' = *OI* (the same object used in the subscription),  
     'Time Remaining' = (60 or less than 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' = *PIDI* (the same identifier used in the subscription),  
     'Initiating Device Identifier' = IUT,  
     'Monitored Object Identifier' = *OI* (the same object used in the subscription),  
     'Time Remaining' = (~60, but not greater than 60),  
     'List of Values' = (values appropriate to the object type of the monitored object)
4. TRANSMIT SubscribeCOV-Request,  
     'Subscriber Process Identifier' = *PIDI* (any valid process identifier),  
     'Monitored Object Identifier' = *OI* (any object supporting COV notifications),  
     'Issue Confirmed Notifications' = TRUE | FALSE,  
     'Lifetime' = *TI*, a value between 180 and 300 seconds)
5. RECEIVE BACnet-SimpleACK-PDU
6. IF (the subscription was for confirmed notifications) THEN  
     **BEFORE Notification Fail Time**  
     RECEIVE ConfirmedCOVNotification-Request,  
     'Subscriber Process Identifier' = *PIDI* (the same identifier used in the subscription),  
     'Initiating Device Identifier' = IUT,  
     'Monitored Object Identifier' = *OI* (the same object used in the subscription),  
     'Time Remaining' = (~*TI*, but not greater than *TI* 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' = *PIDI* (the same identifier used in the subscription),  
     'Initiating Device Identifier' = IUT,  
     'Monitored Object Identifier' = *OI* (the same object used in the subscription),  
     'Time Remaining' = (~*TI*, but not greater than *TI* the requested subscription lifetime),  
     'List of Values' = (values appropriate to the object type of the monitored object)

[Add new **Clause 9.10.1.X1**, p. 353]

### 9.10.1.X1 Ensuring 5 Concurrent COV Subscribers

Purpose: This test case verifies that the IUT can support 5 concurrent subscriptions.

Test Concept: Have the TD subscribe with 5 different process identifiers,  $V_1$  through  $V_5$ , and then check to ensure that 5 notifications are sent when the monitored object changes.

Notes to Tester: The notification in step 3 can be received in any order by the TD.

#### Test Steps

1. REPEAT ( $X=V_1$  to  $V_5$ ) DO {
  - TRANSMIT SubscribeCOV-Request,
    - 'Subscriber Process Identifier' =  $X$ ,
    - 'Monitored Object Identifier' = (any object supporting COV notifications),
    - 'Issue Confirmed Notifications' = TRUE | FALSE,
    - 'Lifetime' = (any valid value that will allow the subscription to outlast the test)
  - RECEIVE BACnet-SimpleACK-PDU
  - IF (if confirmed notifications were requested) THEN
    - BEFORE **Notification Fail Time**
      - RECEIVE ConfirmedCOVNotification-Request,
        - 'Subscriber Process Identifier' =  $X$ ,
        - 'Initiating Device Identifier' = IUT,
        - 'Monitored Object Identifier' = (the same object used in the subscription),
        - 'Time Remaining' = (any valid value),
        - 'List of Values' = (the initial Present\_Value and initial Status\_Flags)
      - TRANSMIT BACnet-SimpleACK-PDU
    - ELSE
      - BEFORE **Notification Fail Time**
        - RECEIVE UnconfirmedCOVNotification-Request,
          - 'Subscriber Process Identifier' =  $X$ ,
          - 'Initiating Device Identifier' = IUT,
          - 'Monitored Object Identifier' = (the same object used in the subscription),
          - 'Time Remaining' = (any valid value),
          - 'List of Values' = (the initial Present\_Value and initial Status\_Flags)
  - 2. MAKE (Present\_Value = any value that differs from "initial Present\_Value" such that a COV notification would be generated)
  - 3. REPEAT ( $X=V_1$  to  $V_5$ ) DO {
    - IF (if confirmed notifications were requested) THEN
      - RECEIVE ConfirmedCOVNotification-Request,
        - 'Subscriber Process Identifier' =  $X$ ,
        - 'Initiating Device Identifier' = IUT,
        - 'Monitored Object Identifier' = (the same object used in the subscription),
        - 'Time Remaining' = (any valid value),
        - 'List of Values' = (the new Present\_Value and Status\_Flags)
      - TRANSMIT BACnet-SimpleACK-PDU
    - ELSE
      - RECEIVE UnconfirmedCOVNotification-Request,
        - 'Subscriber Process Identifier' =  $X$ ,
        - 'Initiating Device Identifier' = IUT,
        - 'Monitored Object Identifier' = (the same object used in the subscription),
        - 'Time Remaining' = (any valid value),

```

        'List of Values' = (the new Present_Value and Status_Flags)
    }

```

## 9.10.2 Negative SubscribeCOV Service Execution Tests

[Change **Clause 9.10.2.1**, p. 353]

[Reviewer Note: Added configuration requirements.]

### 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.

*Configuration Requirements: This test shall only be executed if IUT contains objects which will not accept a COV subscription. If every object in IUT will accept a COV subscription, then this test shall be skipped.*

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) |
      - (BACnet-Error PDU,
        - 'Error Class' = SERVICES,
        - 'Error Code' = SERVICE\_REQUEST\_DENIED | OTHER) |
      - (BACnet-Error PDU,
        - 'Error Class' = PROPERTY,
        - 'Error Code' = NOT\_COV\_PROPERTY)

[Add new **Clause 9.10.2.X1**, p. 354]

[Reviewer Note: 135-2008h-5 allows for a SimpleAck or a specific error code to return if a subscription does not exist.]

### 9.10.2.X1 The Monitored Object Does Not Exist

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

Notes to Tester: If the IUT is able to support objects other than those that currently exist, and none of those objects that currently do not exist would support COV notification if they did, then the IUT may return an error code of OPTIONAL\_FUNCTIONALITY\_NOT\_SUPPORTED instead of UNKNOWN\_OBJECT.

Test Steps:

1. TRANSMIT SubscribeCOV-Request,



```

'Subscriber Process Identifier' = (any valid process identifier),
'Monitored Object Identifier' = (any object of a type that supports COV and an instance which does not exist
                                in the IUT),
'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' = UNKNOWN_OBJECT
ELSE
    RECEIVE BACnet-Error PDU,
        'Error Class' = SERVICES,
        'Error Code' = SERVICE_REQUEST_DENIED | OTHER
    | (BACnet-Error PDU,
        'Error Class' = OBJECT,
        'Error Code' = UNKNOWN_OBJECT)

```

[Add new **Clause 9.10.2.X2**, p. 354]

[Reviewer Note: 135-2008h-5 allows for a SimpleAck or a specific error code to return if there is no space for a subscription.]

#### 9.10.2.X2 There Is No Space For A Subscription

Purpose: To verify that the IUT correctly responds to a SubscribeCOV request to establish a subscription when there is no space for a subscription.

Test Concept: Repeatedly subscribe to the same object each time with a different Process Identifier until the device runs out of resources and returns the appropriate error. This test only applies to IUTs that claim a Protocol\_Revision of 10 or higher.

Test Conditionality: If the device cannot be configured such that the maximum number of subscriptions the IUT can accept is less than 10000, then this test may be skipped.

Test Steps:

```

REPEAT PID = (1 through the maximum number of subscriptions the IUT can accept plus 1, or until the IUT returns
an Error-PDU) {
1. TRANSMIT SubscribeCOV-Request,
    'Subscriber Process Identifier' = PID,
    'Monitored Object Identifier' = (any object of that supports COV),
    'Issue Confirmed Notifications' = TRUE,
    'Lifetime' = 6000
2. RECEIVE BACnet-SimpleACK-PDU |
    (BACnet-Error-PDU,
        'Error Class' = RESOURCES,
        'Error Code' = NO_SPACE_TO_ADD_LIST_ELEMENT)
3. READ ACS = (Active_COV_Subscriptions)
4. IF (a BACnet-SimpleAck BACnet-SimpleACK-PDU was received in step 2) THEN
    CHECK (that the subscription is in ACS)
ELSE
    CHECK (that the subscription is not in ACS)
}

```

[Add new **Clause 9.10.2.X3**, p. 354]

[Reviewer Note: 135-2008h-5 defines COV notification service Error returns.]

### 9.10.2.X3 The Lifetime Parameter is Out of Range

Purpose: To verify that the IUT correctly responds to a SubscribeCOV request to establish a subscription when the Lifetime parameter is out of range.

Test Steps:

1. TRANSMIT SubscribeCOV-Request,  
     'Subscriber Process Identifier' = (any valid process identifier),  
     'Monitored Object Identifier' = (any object in the IUT that supports COV),  
     'Issue Confirmed Notifications' = TRUE,  
     'Lifetime' = (a value less than the maximum unsigned value supported by the IUT, but large enough to produce a Result(-) result by the IUT)
2. IF (Protocol\_Revision is present and Protocol\_Revision >= 10) THEN  
     RECEIVE BACnet-Error-PDU,  
     'Error Class' = SERVICES,  
     'Error Code' = VALUE\_OUT\_OF\_RANGE  
   ELSE  
     RECEIVE BACnet-Error-PDU,  
     'Error Class' = SERVICES,  
     'Error Code' = VALUE\_OUT\_OF\_RANGE | SERVICE\_REQUEST\_DENIED | OTHER  
     | (RECEIVE BACnet-Reject-PDU,  
         Reject Reason = PARAMETER\_OUT\_OF\_RANGE)

[Add new **Clause 9.10.X3**, p. 354]

## 9.10.X3 Positive Unsubscribed COVNotification Execution Tests

### 9.10.X3.1 Unsubscribed COVNotification Execution Test

Purpose: To verify that the IUT executes UnconfirmedCOVNotification service requests, with 'Process Identifier' equal to 0.

Test Concept: Using any received and supported unsubscribed UnconfirmedCOVNotification, observe the effect of its execution.

Test Steps:

1. TRANSMIT UnconfirmedCOVNotification-Request,  
     'Subscriber Process Identifier' = 0,  
     'Initiating Device Identifier' = TD,  
     'Monitored Object Identifier' = (any object present in TD),  
     'Time Remaining' = 0,  
     'List of Values' = (any valid set of values)
3. CHECK (for any vendor-defined observable actions)

## 9.11 SubscribeCOVProperty Service Execution Tests

### 9.11.1 Positive SubscribeCOVProperty Service Execution Tests

[Change **Clause 9.11.1.1**, p. 354]

[Reviewer Note: Removed the allowance for devices which do not support both confirmed and unconfirmed notifications.]

### 9.11.1.1 Confirmed COV Notifications

Purpose: To verify that the IUT correctly responds to a SubscribeCOVProperty 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 9.11.1.2.~~

Test Steps:

1. TRANSMIT SubscribeCOVProperty-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~~),  
     'Monitored Property Identifier' = (any valid property supporting COV notifications)
2. RECEIVE BACnet-SimpleACK-PDU
3. BEFORE **Notification Fail Time**  
     ~~IF (the IUT supports confirmed notifications) THEN~~  
         ~~RECEIVE BACnetConfirmedCOVNotification-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 property subscribed to, and any other properties the IUT provides with it, such as Status-Flags)~~  
     ~~ELSE~~  
         ~~RECEIVE BACnet-Error-PDU,~~  
         ~~Error Class = SERVICES,~~  
         ~~Error Code = SERVICE\_REQUEST\_DENIED | OTHER~~
4. TRANSMIT BACnet-SimpleACK-PDU

[Change **Clause 9.11.1.2**, p. 355]

[Reviewer Note: Removed the allowance for devices which do not support both confirmed and unconfirmed notifications.]

### 9.11.1.2 Unconfirmed COV Notifications

Purpose: To verify that the IUT correctly responds to a SubscribeCOVProperty 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 9.11.1.1.~~

Test Steps:

1. TRANSMIT SubscribeCOVProperty-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~~),  
     'Monitored Property Identifier' = (any valid property supporting COV notifications)
2. RECEIVE BACnet-SimpleACK-PDU
3. BEFORE **Notification Fail Time**  
     ~~IF (the IUT supports unconfirmed notifications) THEN~~  
         RECEIVE BACnetUnconfirmedCOVNotification-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 property subscribed to, and any other properties the IUT provides with it, such as Status-Flags)  
     ~~ELSE~~  
         ~~RECEIVE BACnet-Error-PDU,~~

~~Error Class = SERVICES,~~

~~Error Code = SERVICE\_REQUEST\_DENIED | OTHER~~

[Change **Clause 9.11.1.4**, p. 356]

[Reviewer Note: Added Configuration Requirements and check at end of test to remove the Notes to Tester requirement.]

#### **9.11.1.4 Canceling COV Subscriptions**

~~Dependencies: Indefinite lifetime COV subscriptions, 9.11.1.1.~~

Purpose: To verify that the IUT correctly responds to a SubscribeCOVProperty request to cancel a COV subscription. This test cancels the subscription made in 9.11.1.1.

*Configuration Requirements: This test should be executed after test 9.11.1.1, while the subscription created in that test still exists in the IUT.*

Test Steps:

1. TRANSMIT SubscribeCOVProperty-Request,  
    'Subscriber Process Identifier' = (the process identifier used in test 9.11.1.1),  
    'Monitored Object Identifier' = (the same object used in test 9.11.1.1),  
    'Monitored Property Identifier' = (the same property used in test 9.11.1.1)
2. RECEIVE BACnet-SimpleACK-PDU
3. WAIT Notification Fail Time
4. MAKE (a change to the monitored object that would cause a COV notification if there were an active subscription)
5. *CHECK(the IUT does not transmit a COV notification)*

~~Notes to Tester: The IUT shall not transmit a COV notification message.~~

[Change **Clause 9.11.1.5**, p. 357]

[Reviewer Note: Added missing verification that the IUT did not send a COV notification, and removed superfluous Notes to Tester.]

#### **9.11.1.5 Canceling Expired or Non-Existing Subscriptions**

Purpose: To verify that the IUT correctly responds to a SubscribeCOVProperty request to cancel a subscription that no longer exists.

Test Steps:

1. TRANSMIT SubscribeCOVProperty-Request,  
    'Subscriber Process Identifier' = (any unused process identifier or an identifier from a previously terminated subscription),  
    'Monitored Object Identifier' = (any unused object or an object from a previously terminated subscription),  
    'Monitored Property Identifier' = (any unused property or a property from a previously terminated subscription)
2. RECEIVE BACnet-SimpleACK-PDU
3. WAIT **Notification Fail Time**
4. MAKE (a change to the monitored object that would cause a COV notification if there were an active subscription)
5. *CHECK(the IUT did not issue a COV notification)*

~~Notes to Tester: The IUT shall not transmit a COV notification message. An error message is not an acceptable response.~~

[Change **Clause 9.11.1.7**, p. 357]

[Reviewer Note: Updated description of 'Time Remaining' and adds validation that this value counts down as expected. Corrected the typo error at step 9 and updated the language as per BACnet 135.1, Clause 6.]

### 9.11.1.7 Finite Lifetime Subscriptions

Purpose: To verify that the IUT correctly responds to a SubscribeCOVProperty 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:

1. TRANSMIT SubscribeCOVProperty-Request,  
     'Subscriber Process Identifier' = *(PID1, any valid process identifier)*,  
     'Monitored Object Identifier' = *(OI, any object supporting COV notifications)*,  
     'Issue Confirmed Notifications' = TRUE | FALSE,  
     'Lifetime' = *(T1, any value between 60 seconds and 300 seconds)*,  
     'Monitored Property Identifier' = *(PI, any valid property supporting COV notifications)*
2. RECEIVE BACnet-SimpleACK-PDU
3. ~~BEFORE Notification Fail Time~~  
     IF (the subscription was for confirmed notifications) THEN  
         ~~BEFORE Notification Fail Time~~  
         RECEIVE BACnetConfirmedCOVNotification-Request,  
         'Subscriber Process Identifier' = *PID1* ~~(the same identifier used in the subscription)~~,  
         'Initiating Device Identifier' = IUT,  
         'Monitored Object Identifier' = *OI* ~~(the same object used in the subscription)~~,  
         'Time Remaining' = ~~(the requested subscription lifetime~~ *A value approximately equal to, but not greater than T1)*,  
         'List of Values' = (values appropriate to the property subscribed to, and any other properties the IUT provides with it, such as Status-Flags)  
         TRANSMIT BACnet-SimpleACK-PDU  
     ELSE  
         ~~BEFORE Notification Fail Time~~  
         RECEIVE BACnetUnconfirmedCOVNotification-Request,  
         'Subscriber Process Identifier' = *PID1* ~~(the same identifier used in the subscription)~~,  
         'Initiating Device Identifier' = IUT,  
         'Monitored Object Identifier' = *OI* ~~(the same object used in the subscription)~~,  
         'Time Remaining' = ~~(the requested lifetime)~~ *(A value approximately equal to, but no greater than T1)*,  
         'List of Values' = (values appropriate to the property subscribed to, and any other properties the IUT provides with it, such as Status-Flags)  
     4. WAIT a period longer than the resolution of the IUT's COV subscription lifetime timer  
     54. MAKE (a change to the monitored object that ~~should~~ causes a COV notification)  
     5. ~~BEFORE Notification Fail Time~~
6. IF (the subscription was for confirmed notifications) THEN  
     ~~BEFORE Notification Fail Time~~  
     RECEIVE BACnetConfirmedCOVNotification-Request,  
     'Subscriber Process Identifier' = *PID1* ~~(the same identifier used in the subscription)~~,  
     'Initiating Device Identifier' = IUT,  
     'Monitored Object Identifier' = *OI* ~~(the same object used in the subscription)~~,  
     'Time Remaining' = *(T2: a value greater than 0 and less than the requested subscription lifetime)*,  
     'List of Values' = (values appropriate to the property subscribed to, and any other properties the IUT provides with it, such as Status-Flags)  
     TRANSMIT BACnet-SimpleACK-PDU  
     ELSE  
         ~~BEFORE Notification Fail Time~~  
         RECEIVE BACnetUnconfirmedCOVNotification-Request,  
         'Subscriber Process Identifier' = *PID1* ~~(the same identifier used in the subscription)~~,  
         'Initiating Device Identifier' = IUT,

'Monitored Object Identifier' = *OI* (the same object used in the subscription),  
 'Time Remaining' = (*T2*: 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 that triggered the notification)

~~6. WAIT (the lifetime of the subscription),~~

7. *WAIT a period longer than the resolution of the IUT's COV subscription lifetime timer*

8. *MAKE (a change to the monitored object that causes a COV notification)*

9. *IF (the subscription was for confirmed notifications) THEN*

*BEFORE Notification Fail Time*

*RECEIVE BACnetConfirmedCOVNotification-Request,*

*'Subscriber Process Identifier' = PIDI,*

*'Initiating Device Identifier' = IUT,*

*'Monitored Object Identifier' = OI,*

*'Time Remaining' = (a value greater than 0 and less than the T2),*

*'List of Values' = (values appropriate to the object type of the monitored object)*

*ELSE*

*BEFORE Notification Fail Time*

*RECEIVE BACnetUnconfirmedCOVNotification-Request,*

*'Subscriber Process Identifier' = PIDI,*

*'Initiating Device Identifier' = IUT,*

*'Monitored Object Identifier' = OI,*

*'Time Remaining' = (a value greater than 0 and less than the T2),*

*'List of Values' = (values appropriate to the object type of the monitored object including the changed value that triggered the notification)*

610. *WAIT (the lifetime of the subscription)*

711. *MAKE (a change to the monitored object that would cause a COV notification if there were an active subscription)*

12. *CHECK (verify that the IUT did not transmit a COV notification message)*

~~Notes to Tester: The IUT shall not transmit a COV notification message addressed to the TD after step 6.~~

[Change **Clause 9.11.1.8**, p. 358]

[Reviewer Note: Modified as per purpose, BACnet Clause 13.14.2, and updated the language as per BACnet 135.1, Clause 6.]

### 9.11.1.8 Updating Existing Subscriptions

Purpose: To verify that the IUT correctly responds to a SubscribeCOVProperty 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:

1. TRANSMIT SubscribeCOVProperty-Request,

'Subscriber Process Identifier' = (*PIDI*, any valid process identifier),

'Monitored Object Identifier' = (*OI*, any object supporting COV notifications),

'Issue Confirmed Notifications' = TRUE | FALSE,

'Lifetime' = 60

'Monitored Property Identifier' = (*PI*, any valid property supporting COV notifications)

2. RECEIVE BACnet-SimpleACK-PDU

~~3. BEFORE Notification Fail Time~~

3. *IF (the subscription was for confirmed notifications) THEN*

*BEFORE Notification Fail Time*

*RECEIVE BACnetConfirmedCOVNotification-Request,*

*'Subscriber Process Identifier' = PIDI (the same identifier used in the subscription),*

```

        'Initiating Device Identifier' = IUT,
        'Monitored Object Identifier' = OI (the same object used in the subscription),
        'Time Remaining' = (60 or less than 60),
        'List of Values' = (values appropriate to the object type and subscribed to property of the monitored object
including the changed value of that triggered the notification)
        TRANSMIT BACnet-SimpleACK-PDU
    ELSE
        BEFORE Notification Fail Time
        RECEIVE BACnetUnconfirmedCOVNotification-Request,
        'Subscriber Process Identifier' = PIDI (the same identifier used in the subscription),
        'Initiating Device Identifier' = IUT,
        'Monitored Object Identifier' = OI (the same object used in the subscription),
        'Time Remaining' = (60 or less than 60),
        'List of Values' = (values appropriate to the object type and subscribed to property of the monitored object
including the changed value of that triggered the notification)
4.  TRANSMIT SubscribeCOVProperty-Request,
    'Subscriber Process Identifier' = PIDI (any valid process identifier),
    'Monitored Object Identifier' = OI (any object supporting COV notifications),
    'Issue Confirmed Notifications' = TRUE | FALSE,
    'Lifetime' = (TI, a value between 180 and 300 seconds)
    'Monitored Property Identifier' = PI (any valid property supporting COV notifications)
5.  RECEIVE BACnet-SimpleACK-PDU
6.  BEFORE Notification Fail Time
6.  IF (the subscription was for confirmed notifications) THEN
    BEFORE Notification Fail Time
    RECEIVE BACnetConfirmedCOVNotification-Request,
    'Subscriber Process Identifier' = PIDI (the same identifier used in the subscription),
    'Initiating Device Identifier' = IUT,
    'Monitored Object Identifier' = OI (the same object used in the subscription),
    'Time Remaining' = (~TI, but not greater than TI the requested subscription lifetime),
    'List of Values' = (values appropriate to the object type and subscribed to property of the monitored object
including the changed value of that triggered the notification)
    TRANSMIT BACnet-SimpleACK-PDU
    ELSE
        BEFORE Notification Fail Time
        RECEIVE BACnetUnconfirmedCOVNotification-Request,
        'Subscriber Process Identifier' = PIDI (the same identifier used in the subscription),
        'Initiating Device Identifier' = IUT,
        'Monitored Object Identifier' = OI (the same object used in the subscription),
        'Time Remaining' = (~TI, but not greater than TI the requested subscription lifetime),
        'List of Values' = (values appropriate to the object type and subscribed to property of the monitored object
including the changed value of that triggered the notification)

```

[Change **Clause 9.11.1.9**, p. 359]

[Reviewer Note: Modified to work with all numeric datatypes.]

### 9.11.1.9 Client-Supplied COV Increment

**Purpose:** To verify that the IUT correctly generates COV notifications when the client supplies the COV increment in the SubscribeCOVProperty request. 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 notification is made for a property of *numeric* datatype ~~REAL~~. The subscription request specifies a COV increment. The monitored property is changed by an amount less than the increment, and the TD waits to ensure that the IUT does not generate a notification. The monitored property is changed by an amount slightly more than is required to cause a COV notification, and the TD waits for the notification.

~~Test~~ *Configuration Requirements:* If the property being subscribed to has a related COV\_Increment property in the object, then the value of the COV\_Increment property should be significantly different than the COV increment provided in the subscription service. *In devices where the 'COV Increment' is always less than the minimal change that the monitored property can make, skip steps 4 and 5.*

Test Steps:

1. TRANSMIT SubscribeCOVProperty-Request,
  - 'Subscriber Process Identifier' = (any valid process identifier),
  - 'Monitored Object Identifier' = (any object supporting COV notifications),
  - 'Issue Confirmed Notifications' = TRUE | FALSE,
  - 'Lifetime' = (any value that will ensure no re-subscription is required to complete the test),
  - 'Monitored Property Identifier' = (any valid property supporting COV notifications),
  - 'COV Increment' = (any valid increment value)
2. RECEIVE BACnet-SimpleACK-PDU
3. BEFORE **Notification Fail Time**
  - IF (the subscription was for confirmed notifications) THEN
    - RECEIVE BACnetConfirmedCOVNotification-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 lifetime),
      - 'List of Values' = (values appropriate to the object type of the monitored object including the value of monitored property)
    - TRANSMIT BACnet-SimpleACK-PDU
  - ELSE
    - RECEIVE BACnetUnconfirmedCOVNotification-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 lifetime),
      - 'List of Values' = (values appropriate to the object type of the monitored object including the value of monitored property)
4. MAKE (the monitored property change by less than the COV increment)
5. CHECK (~~verify~~ that the IUT did not transmit a notification message for the monitored property)
6. MAKE (the monitored property change by slightly more than COV Increment less the amount changed in step 54)
7. BEFORE **Notification Fail Time**
  - IF (the subscription was for confirmed notifications) THEN
    - RECEIVE BACnetConfirmedCOVNotification-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' = ?,
      - '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
    - RECEIVE BACnetUnconfirmedCOVNotification-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' = ?,
      - 'List of Values' = (values appropriate to the object type of the monitored object including the changed value that triggered the notification)

[Add new **Clause 9.11.1.X10**, p. 360]



### 9.11.1.X10 Accepts SubscribeCOVProperty-Requests with 8 Hour Lifetimes

Purpose: To verify that the IUT correctly accepts lifetimes of at least 8 hours.

Test Steps:

1. TRANSMIT SubscribeCOVProperty-Request,  
     'Subscriber Process Identifier' = (any valid process identifier),  
     'Monitored Object Identifier' = (any object supporting COV notifications),  
     'Issue Confirmed Notifications' = TRUE | FALSE,  
     'Lifetime' = 28800  
     'Monitored Property Identifier' = (any valid property supporting COV notifications)
2. RECEIVE BACnet-SimpleACK-PDU
3. BEFORE Notification Fail Time  
     IF (the subscription was for confirmed notifications) THEN  
         RECEIVE BACnetConfirmedCOVNotification-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 lifetime),  
         'List of Values' = (values appropriate to the object type of the monitored object including the value of monitored property)  
         TRANSMIT BACnet-SimpleACK-PDU  
     ELSE  
         RECEIVE BACnetUnconfirmedCOVNotification-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 lifetime),  
         'List of Values' = (values appropriate to the object type of the monitored object including the value of monitored property)
4. TRANSMIT SubscribeCOVProperty-Request,  
     'Subscriber Process Identifier' = (the same identifier used in Step 1),  
     'Monitored Object Identifier' = (the same identifier used in the subscription),  
     'Monitored Property Identifier' = (the same object used in the subscription)
5. RECEIVE BACnet-SimpleACK-PDU

[Add new **Clause 9.11.1.X11**, p. 360]

### 9.11.1.X11 Confirmed Change of Value Notification from Property Value

Purpose: To verify that the IUT can initiate ConfirmedCOVNotification service requests conveying a change of the Property Value.

Test Concept: A property subscription for COV notifications is established, using a Lifetime of L. L shall be set to a value less than 24 hours and large enough to complete the test. The Value of the monitored Property is changed, and a notification shall be received. The subscribed property may be changed using the WriteProperty service or by another means. For implementations where it is not possible to write to these properties at all, the vendor shall provide an alternative trigger mechanism to accomplish this task. All of these methods are equally acceptable.

Test Steps:

1. TRANSMIT SubscribeCOVProperty-Request,  
     'Subscriber Process Identifier' = (any valid process identifier),  
     'Monitored Object Identifier' = X  
     'Issue Confirmed Notifications' = TRUE,  
     'Lifetime' = L  
     'Monitored Property Identifier' = Y (any valid property supporting COV notifications)
2. RECEIVE BACnet-SimpleACK-PDU
3. BEFORE Notification Fail Time

RECEIVE BACnetConfirmedCOVNotification-Request,

'Subscriber Process Identifier' = (the same identifier used in the subscription),

'Initiating Device Identifier' = IUT,

'Monitored Object Identifier' = X

'Time Remaining' = (any value appropriate for the Lifetime selected),

'List of Values' = (values appropriate to the property subscribed to, and any other properties the IUT provides

with it, such as Status-Flags)

4. TRANSMIT BACnet-SimpleACK-PDU

5. MAKE (a change to the monitored object PROPERTY that causes a COV notification)

6. BEFORE Notification Fail Time

RECEIVE BACnetConfirmedCOVNotification-Request,

'Subscriber Process Identifier' = (the same identifier used in the subscription),

'Initiating Device Identifier' = IUT,

'Monitored Object Identifier' = X

'Time Remaining' = (any value appropriate for the Lifetime selected),

'List of Values' = (values appropriate to the property subscribed to, and any other properties the IUT provides

with it, such as Status-Flags)

7. TRANSMIT BACnet-SimpleACK-PDU

8. TRANSMIT SubscribeCOVProperty-Request,

'Subscriber Process Identifier' = (the same identifier used in Step 1),

'Monitored Object Identifier' = X

'Monitored Property Identifier' = Y

9. RECEIVE BACnet-SimpleACK-PDU

[Add new **Clause 9.11.1.X12**, p. 360]

#### **9.11.1.X12 Unconfirmed Change of Value Notification from Property Value**

Purpose: To verify that the IUT can initiate UnconfirmedCOVNotification service requests conveying a change of the Property Value.

Test Steps: The steps for this test case are identical to the test steps in 9.11.1.X11, except that the SubscribeCOVProperty service request in step 1 shall have a value of FALSE for the 'Issue Confirmed Notifications' parameter, all of the ConfirmedCOVNotification requests shall be UnconfirmedCOVNotification requests, and there is no acknowledgment of the unconfirmed services.

[Add new **Clause 9.11.1.X21**, p. 360]

#### **9.11.1.X21 Confirmed Change of Value Notification from Status\_Flags Property**

Purpose: To verify that the IUT can initiate ConfirmedCOVNotification service requests conveying a change of the Status\_Flags Property.

Test Concept: A property subscription for COV notifications is established, using a Lifetime of L. L shall be set to a value less than 24 hours and large enough to complete the test. 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.

Test Steps:

1. TRANSMIT SubscribeCOVProperty-Request,

'Subscriber Process Identifier' = (any valid process identifier),

'Monitored Object Identifier' = X

'Issue Confirmed Notifications' = TRUE,

'Lifetime' = L

'Monitored Property Identifier' = Y (any valid property supporting COV notifications)

2. RECEIVE BACnet-SimpleACK-PDU
3. BEFORE Notification Fail Time
  - RECEIVE BACnetConfirmedCOVNotification-Request,
    - 'Subscriber Process Identifier' = (the same identifier used in the subscription),
    - 'Initiating Device Identifier' = IUT,
    - 'Monitored Object Identifier' = X
    - 'Time Remaining' = (any value appropriate for the Lifetime selected),
    - 'List of Values' = (values appropriate to the property subscribed to and initial Status\_Flags)
4. TRANSMIT BACnet-SimpleACK-PDU
5. MAKE (Status\_Flags = any value that differs from "initial Status\_Flags")
6. BEFORE Notification Fail Time
  - RECEIVE BACnetConfirmedCOVNotification-Request,
    - 'Subscriber Process Identifier' = (the same identifier used in the subscription),
    - 'Initiating Device Identifier' = IUT,
    - 'Monitored Object Identifier' = X
    - 'Time Remaining' = (any value appropriate for the Lifetime selected),
    - 'List of Values' = (initial values appropriate to the property subscribed to and new Status\_Flags)
7. TRANSMIT SubscribeCOVProperty-Request,
  - 'Subscriber Process Identifier' = (the same identifier used in Step 1),
  - 'Monitored Object Identifier' = X
  - 'Monitored Property Identifier' = Y
8. RECEIVE BACnet-SimpleACK-PDU

[Add new **Clause 9.11.1.X22**, p. 360]

#### **9.11.1.X22 Unconfirmed Change of Value Notification from Status\_Flags Property**

Purpose: To verify that the IUT can initiate UnconfirmedCOVNotification service requests conveying a change of the Status\_Flags Property.

Test Steps: The steps for this test case are identical to the test steps in 9.11.1.X21 except that the SubscribeCOVProperty service request in step 1 shall have a value of FALSE for the 'Issue Confirmed Notifications' parameter, all of the ConfirmedCOVNotification requests shall be UnconfirmedCOVNotification requests, and there is no acknowledgment of the unconfirmed services. The MAC address used for the notification message shall be such that the TD is one of the recipients

#### **9.11.2 Negative SubscribeCOVProperty Service Execution Tests**

[Change **Clause 9.11.2.1**, p. 361]

[Reviewer Note: Added additional acceptable error responses.]

##### **9.11.2.1 The Monitored Object Does Not Support COV Notification**

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

Test Steps:

1. TRANSMIT SubscribeCOVProperty-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,
  - 'Monitored Property Identifier' = (any property in the object)
2. IF (Protocol\_Revision < 15) THEN

```

RECEIVE
  (BACnet-Error-PDU,
    'Error Class' = SERVICES,
    'Error Code' = SERVICE_REQUEST_DENIED | OTHER) |
  (BACnet-Error-PDU,
    'Error Class' = OBJECT,
    'Error Code' = OPTIONAL_FUNCTIONALITY_NOT_SUPPORTED) |
  (BACnet-Error-PDU,
    'Error Class' = PROPERTY,
    'Error Code' = NOT_COV_PROPERTY)
ELSE
  RECEIVE
    (BACnet-Error-PDU,
      'Error Class' = OBJECT,
      'Error Code' = OPTIONAL_FUNCTIONALITY_NOT_SUPPORTED) |
    (BACnet-Error-PDU,
      'Error Class' = PROPERTY,
      'Error Code' = NOT_COV_PROPERTY)

```

[Change **Clause 9.11.2.2**, p. 361]

[Reviewer Note: Changed description of Monitored Property Identifier to be chosen for this test.]

### 9.11.2.2 The Monitored Property Does Not Support COV Notification

Purpose: To verify that the IUT correctly responds to a SubscribeCOVProperty request to establish a subscription when the monitored object supports COV notifications but not on the requested property.

Test Steps:

1. TRANSMIT SubscribeCOVProperty-Request,
  - 'Subscriber Process Identifier' = (any valid process identifier),
  - 'Monitored Object Identifier' = (any object that supports COV notifications),
  - 'Issue Confirmed Notifications' = TRUE,
  - 'Lifetime' = 60,
  - 'Monitored Property Identifier' = (any property *of the chosen object* that does not support COV notifications)
2. IF (Protocol\_Revision < 15) THEN
  - RECEIVE
    - (BACnet-Error-PDU,
 'Error Class' = SERVICES,
 'Error Code' = SERVICE\_REQUEST\_DENIED | OTHER) |
 (BACnet-Error-PDU,
 'Error Class' = PROPERTY,
 'Error Code' = NOT\_COV\_PROPERTY)
  - ELSE
    - RECEIVE BACnet-Error-PDU,
 'Error Class' = PROPERTY,
    - 'Error Code' = NOT\_COV\_PROPERTY

[Add new **Clause 9.11.2.X11**, p. 361]

### 9.11.2.X11 Monitored Object Does Not Exist

Purpose: To verify that the IUT correctly responds to a SubscribeCOVProperty request to establish a subscription when the monitored object does not exist.

Test Steps:

1. TRANSMIT SubscribeCOVProperty-Request,

- 'Subscriber Process Identifier' = (any valid process identifier),
  - 'Monitored Object Identifier' = (any object of a type that supports COV and an instance which does not exist in the IUT),
  - 'Issue Confirmed Notifications' = TRUE,
  - 'Lifetime' = 60
  - 'Monitored Property Identifier' = (any valid property supporting COV notifications)
2. RECEIVE BACnet-Error-PDU,
    - 'Error Class' = OBJECT,
    - 'Error Code' = UNKNOWN\_OBJECT

[Add new **Clause 9.11.2.X12**, p. 361]

### 9.11.2.X12 Monitored Property Does Not Exist

Purpose: To verify that the IUT correctly responds to a SubscribeCOVProperty request to establish a subscription when the monitored property does not exist.

Test Steps:

1. TRANSMIT SubscribeCOVProperty-Request,
  - 'Subscriber Process Identifier' = (any valid process identifier),
  - 'Monitored Object Identifier' = (object supporting COV notifications),
  - 'Issue Confirmed Notifications' = TRUE,
  - 'Lifetime' = 60
  - 'Monitored Property Identifier' = (any valid property supporting COV notifications which does not exist for specified object)
2. RECEIVE BACnet-Error-PDU,
  - 'Error Class' = PROPERTY,
  - 'Error Code' = UNKNOWN\_PROPERTY

[Add new **Clause 9.11.2.X13**, p. 361]

### 9.11.2.X13 There Is No Space For Subscription

Purpose: To verify that the IUT correctly responds to a SubscribeCOVProperty request to establish a subscription when there is no space for a subscription.

Test Concept: Repeatedly subscribe to the same object each time with a different Process Identifier until the device runs out of resources and returns the appropriate error. This test only applies to IUTs that claim a Protocol\_Revision of 10 or higher.

Test Conditionality: If the device cannot be configured such that the maximum number of subscriptions the IUT can accept is less than 10000, then this test may be skipped.

Test Steps:

REPEAT PID = (1 through the maximum number of subscriptions the IUT can accept plus 1, or until the IUT returns an Error-PDU) {

1. TRANSMIT SubscribeCOVProperty-Request,
    - 'Subscriber Process Identifier' = PID,
    - 'Monitored Object Identifier' = (object supporting COV notifications),
    - 'Issue Confirmed Notifications' = TRUE,
    - 'Lifetime' = 6000
    - 'Monitored Property Identifier' = (any valid property supporting COV notifications)
  2. RECEIVE BACnet-SimpleACK-PDU
    - | (BACnet-Error-PDU,
    - 'Error Class' = RESOURCES,
    - 'Error Code' = NO\_SPACE\_TO\_ADD\_LIST\_ELEMENT)
- }

[Add new **Clause 9.11.2.X14**, p. 361]

### 9.11.2.X14 The Lifetime Parameter is Out of Range

Purpose: To verify that the IUT correctly responds to a SubscribeCOVProperty request to establish a subscription when the Lifetime parameter is out of range.

Test Steps:

1. TRANSMIT SubscribeCOVProperty-Request,  
     'Subscriber Process Identifier' = (any valid process identifier),  
     'Monitored Object Identifier' = (object supporting COV notifications),  
     'Issue Confirmed Notifications' = TRUE,  
     'Lifetime' = (a value larger than that supported by the IUT),  
     'Monitored Property Identifier' = (any valid property supporting COV notifications)
2. IF (Protocol\_Revision is present and Protocol\_Revision => 15) THEN  
     RECEIVE BACnet-Error-PDU,  
     'Error Class' = SERVICES,  
     'Error Code' = VALUE\_OUT\_OF\_RANGE  
   ELSE  
     RECEIVE BACnet-Error-PDU,  
     'Error Class' = SERVICES,  
     'Error Code' = VALUE\_OUT\_OF\_RANGE | SERVICE\_REQUEST\_DENIED | OTHER  
                   | (RECEIVE BACnet-Reject-PDU,  
                     Reject Reason = PARAMETER\_OUT\_OF\_RANGE)

## 9.12 Atomic ReadFile Service Execution Tests

### 9.12.1 Positive AtomicReadFile Service Execution Tests

#### 9.12.1.2 Reading Stream Based Files

[Change **Clause 9.12.1.2.1**, p. 364]

[Reviewer Note: Change to allow testing of files larger than that which can be returned in a single request.]

#### 9.12.1.2.1 Reading an Entire *Stream-Based* File

Purpose: To verify that the IUT correctly responds to a request to read an entire file.

*Test Concept: The test consists of reading the contents of the file using a sequence of AtomicReadFile requests and verifying that the appropriate known file data is returned.*

*Configuration Requirements: The AtomicReadFile service execution tests require that the TD has knowledge of the exact contents of a known file F1. The test procedures assume that the IUT is already configured with the known file data provided by the manufacturer. In the test procedures "X" will designate the File object identifier and Z the 'File Start Position' initialized at "0". When performing the AtomicReadFile services, a Maximum Requested Octet Count (MROC) shall be calculated before starting the test. These values shall be used during the test. MROC shall be 16 less than the minimum of the TD's Max\_APDU\_Length\_Accepted and the IUT's maximum transmittable APDU length.*

Test Steps:

1. ~~TRANSMIT AtomicReadFile Request,~~  
     ~~'File Identifier' = S,~~

```

'File Start Position' = 0,
'Requested Octet Count' = (the number of octets in the test file)
2. RECEIVE AtomicReadFile ACK,
'End of File' = TRUE,
'File Start Position' = 0,
'File Data' = (the known contents of the test file)
1. VERIFY File Access Method = STREAM_ACCESS
2. WHILE (the last read resulted in an Ack with 'End Of File' == FALSE) DO {
    TRANSMIT AtomicReadFile-Request,
    'Object Identifier' = X,
    'File Start Position' = Z (the next unread octet),
    'Requested Octet Count' = MROC
    RECEIVE AtomicReadFile-ACK,
    'End Of File' = TRUE | FALSE,
    'File Start Position' = Z
    'File Data' = (the known contents of the test file of length MROC if 'End Of File' is
        FALSE or of length MROC or less if 'End Of File' is TRUE)
    }
3. CHECK (that the returned file data is F1)

```

## 9.13 AtomicWriteFile Service Execution Tests

### 9.13.1 Positive AtomicWriteFile Service Execution Tests

#### 9.13.1.2 Writing to Stream-Based Files

[Change **Clause 9.13.1.2.1**, p. 373]

[Reviewer Note: Modified to allow the entire file content to be written in all cases.]

##### 9.13.1.2.1 Writing an Entire *Stream-Based* File

Purpose: To verify that the IUT correctly responds to a request to write an entire file.

*Test Concept: The tests consist of modifying the contents of the files using the AtomicWriteFile service and verifying that the appropriate changes to the file data took place*

*Configuration Requirements: The test data shall contain at least as many octets as the initial data for the file. The manufacturer shall provide appropriate test data to write to these files or sufficient information to permit the tester to construct the test data. The file objects shall be configured with initial data that differs from the test data. In the test procedures, "X" will designate the File object identifier and Z the File Start Position' initialized at "1" at the beginning. When performing the AtomicWriteFile services, a Maximum Write Data Length (MWDL) shall be calculated before starting the test. These values shall be used during the test. MWDL shall be 21 less than the minimum of the TD's maximum transmittable APDU length and the IUT's Max\_APDU\_Length\_Accepted.*

Test Steps:

```

1. TRANSMIT AtomicReadFile Request,
'File Identifier' = S,
'File Start Position' = 0,
'Requested Octet Count' = (any number ≥ the number of octets in the test data)
2. RECEIVE AtomicReadFile ACK,
'End of File' = TRUE,
'File Start Position' = 0,
'File Record Data' = (the initial data)
3. TRANSMIT AtomicWriteFile Request,

```

```

_____'File Identifier' = _____ S,
_____'File Start Position' = _____ 0,
_____'File Data' = _____ (the test data)
4. RECEIVE AtomicWriteFile-ACK,
_____'File Start Position' = _____ 0
5. TRANSMIT AtomicReadFile-Request,
_____'File Identifier' = _____ S,
_____'File Start Position' = _____ 0,
_____'Requested Octet Count' = _____ (any number > the number of octets in the test data)
6. RECEIVE AtomicReadFile-ACK,
_____'End of File' = _____ TRUE,
_____'File Start Position' = _____ 0,
_____'File Data' = _____ (the test data)
7. VERIFY (R), Modification_Date = (the current date and time)
8. VERIFY (R), ARCHIVE = _____ FALSE
9. VERIFY (R), Number_Of_Records = (the number of records in the test data)
1. VERIFY Read_Only = _____ FALSE
2. WRITE Archive = _____ TRUE
3. VERIFY File_Access_Method = _____ STREAM ACCESS
4. IF (File_Size is not equal to the size of the test file) THEN
    WRITE File_Size = _____ 0
5. REPEAT Z = (0 through the file size, in increments of MWDL) DO {
    TRANSMIT AtomicWriteFile-Request
    'File Identifier' = X
    'File Start Position' = Z
    'File Data' = _____ (file contents, the number of octets being the lesser of (file size - Z) and MWDL)
    RECEIVE AtomicWriteFile-ACK
    'File Start Position' = _____ Z
    }
6. VERIFY File_Size = _____ (file size of the test data)
7. VERIFY Modification_Date = _____ (the current date and time)
8. VERIFY ARCHIVE = _____ FALSE

```

[Change **Clause 9.13.1.2.3**, p. 374]

[Reviewer Note: Added configuration requirements, fixed purpose, and removed reliance on AtomicReadFile within the test steps.]

### 9.13.1.2.3 Appending Data to the End of a File

Purpose: To verify that the IUT correctly responds to a request to write to the end of a file. ~~If the IUT does not support files that can not be modified except by replacing the entire file, and this restriction is clearly stated in the PICS, then this test may be ignored.~~

Configuration Requirements: *The manufacturer shall provide appropriate test data to write to these files or sufficient information to permit the tester to construct the test data.* The file objects shall be configured with initial data that differs from the test data. *In the test procedures, "X" will designate the File object identifier and when performing the AtomicWriteFile services, a Maximum Write Data Length (MWDL) shall be calculated before starting the test. These values shall be used during the test. MWDL shall be 21 less than the minimum of the TD's maximum transmittable APDU length and the IUT's Max\_APDU\_Length\_Accepted.*

Test Steps:

1. TRANSMIT ReadProperty-Request,  
    'Object Identifier' = S,  
    'Property Identifier' = File\_Size
2. RECEIVE ReadProperty-ACK,  
    'Object Identifier' = S,  
    'Property Identifier' = File\_Size,



- 'Property Value' = (the current size in octets, designated "InitialNumOctets" below)
3. TRANSMIT AtomicReadFile-Request,  
 'File Identifier' = S,  
 'File Start Position' = 0,  
 'Requested Octet Count' = (any number > InitialNumOctets)
4. RECEIVE AtomicReadFile-ACK,  
 'End of File' = TRUE,  
 'File Start Position' = 0,  
 'File Data' = (the initial data)
3. TRANSMIT AtomicWriteFile-Request,  
 'File Identifier' = S,  
 'File Start Position' = -1,  
 'File Data' = (the test data, the number of octets being the lesser of (file size - Z) and MWDL)
4. RECEIVE AtomicWriteFile-ACK,  
 'File Start Position' = InitialNumOctets,
7. TRANSMIT AtomicReadFile-Request,  
 'File Identifier' = S,  
 'File Start Position' = InitialNumOctets,  
 'Requested Octet Count' = (the number of octets in the test data)
8. RECEIVE AtomicReadFile-ACK,  
 'End of File' = TRUE,  
 'File Start Position' = InitialNumOctets,  
 'File Data' = (the test data)
5. VERIFY (R), Modification\_Date = (the current date and time)
6. VERIFY (R), ARCHIVE = FALSE
7. VERIFY (R), File\_Size = (the number of octets in the test data + InitialNumOctets)

## 9.14 AddListElement Service Execution Tests

### 9.14.2 Negative AddListElement Service Execution Tests

[Change Clause 9.14.2.2, p. 381]

[Reviewer Note: Added the additional error conditions that are now accepted. Added 'Note to Tester' that was missing in 135.1-2013.]

#### 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.

*Notes to Tester: value selected for step 1 is 'inappropriate', not a value which is 'allowed', but not supported by this instance of the property, i.e., it is not one of the datatypes that would ever be supported by an instance of this property in this object type. DATATYPE\_NOT\_SUPPORTED is only correct when the datatype requested is supported, for example, in a CHOICE, by this property in this object type, but not supported by this instance of the property.*

Test Steps:

- TRANSMIT AddListElement-Request,  
 'Object Identifier' = L,  
 'Property Identifier' = ListProp,  
 'List of Elements' = (a single element with a datatype inappropriate for this property)
- RECEIVE AddListElement-Error,  
 'Error Class' = PROPERTY,

```
'Error Code' =          INVALID_DATATYPE,
'First Failed Element' = 1 |
(BACnet-Reject-PDU
  Reject Reason =          INVALID_PARAMETER_DATATYPE) |
(BACnet-Reject-PDU
  Reject Reason =          INVALID_TAG)
```

[Change **Clause 9.14.2.2**, p. 381]

[Reviewer Note: Updated test to include additional error codes. Added 'Notes to Tester' which was missing in 135.1-2013.]

### 9.14.2.3 An AddListElement Failure Part Way Through a List

Purpose: To verify the ability of the IUT to respond to an AddListElement service request to add multiple elements to a list where one of the elements cannot be added. Upon failure, the AddListElement service should leave the list unchanged.

*Notes to Tester: value selected for step 3 is 'inappropriate', not a value which is 'allowed', but not supported by this instance of the property, i.e., it is not one of the datatypes that would ever be supported by an instance of this property in this object type. DATATYPE\_NOT\_SUPPORTED is only correct when the datatype requested is supported, for example, in a CHOICE, by this property in this object type, but not supported by this instance of the property.*

Test Steps:

1. READ InitialList = (L), ListProp2
2. TRANSMIT AddListElement-Request,
 

```
'Object Identifier' =      L,
'Property Identifier' =      ListProp
'List of Elements' =        (two or more elements to be added to the list with the second element
                             having an inappropriate datatype)
```
3. IF (Protocol\_Revision is present and Protocol\_Revision >= 7) THEN
 

```
RECEIVE AddListElement-Error,
'Error Class' =              PROPERTY,
'Error Code' =              INVALID_DATATYPE,
'First Failed Element' =      2
| (RECEIVE BACnet-Reject-PDU,
  Reject Reason = INVALID_TAG | INVALID_PARAMETER_DATA_TYPE)
```
- ELSE
 

```
RECEIVE AddListElement-Error,
'Error Class' =              SERVICES,
'Error Code' =              INVALID_PARAMETER_DATATYPE
'First Failed Element' =      2
| (AddListElement-Error,
  'Error Class' =              PROPERTY,
  'Error Code' =              INVALID_DATATYPE)
'First Failed Element' =      2
| (BACnet-Reject-PDU,
  Reject Reason = INVALID_TAG | INVALID_PARAMETER_DATA_TYPE)
```
4. VERIFY (L), ListProp = InitialList

## 9.15 RemoveListElement Service Execution Tests

### 9.15.2 Negative RemoveListElement Service Execution Tests

[Change **Clause 9.15.2.2**, p. 383]

[Reviewer Note: The test specified an incorrect error code.]

### 9.15.2.2 A RemoveListElement Failure Part Way Through a List

Purpose: To verify the ability of the IUT to respond to a RemoveListElement service request to remove multiple elements from a list where one of the elements cannot be removed. Upon failure, the RemoveListElement service should leave the list unchanged.

Test Steps:

1. READ InitialList = (L), ListProp
2. TRANSMIT RemoveListElement-Request,  
     'Object Identifier' = L,  
     'Property Identifier' = ListProp  
     'List of Elements' = (one element from InitialList, followed by an element of the correct datatype that is not in InitialList, followed by one or more elements from InitialList)
3. If (Protocol\_Revision is present and Protocol\_Revision >= 7) THEN  
     RECEIVE RemoveListElement-Error,  
     'Error Class' = ~~PROPERTY SERVICES~~,  
     'Error Code' = ~~INVALID\_DATA+TYPE~~LIST\_ELEMENT\_NOT\_FOUND  
     'First Failed Element' = 2  
   ELSE  
     RECEIVE RemoveListElement-Error  
     ~~Error Class = SERVICES | PROPERTY,~~  
     ~~Error Code = OTHER~~  
     ~~'First Failed Element' = 2~~
4. VERIFY (L), ListProp = InitialList

## 9.16 CreateObject Service Execution Tests

### 9.16.1 Positive CreateObject Service Execution Tests

[Change **Clause 9.16.1.2**, p. 384]

[Reviewer Note: Added clarification that the IUT can place a restriction on the instance used. This correction is not in any SSPC proposal.]

#### 9.16.1.2 Creating Objects by Specifying the Object Identifier with No Initial Values

Purpose: To verify the correct execution of the CreateObject service request when an Object Identifier is used as the object specifier.

Test Steps:

1. TRANSMIT CreateObject-Request,  
     'Object Specifier' = (any unique object identifier of a type that is creatable *and an instance number that is creatable*)
2. RECEIVE CreateObject-ACK,  
     'Object Identifier' = (the object identifier specified in step 1)
3. VERIFY (the object identifier of the newly created object),  
     (any required property of the specified object) = (any value of the correct datatype for the specified property)
4. VERIFY (the IUT's Device object), Object\_List = (any object list containing the newly created object)

[Change **Clause 9.16.1.4**, p. 384]

[Reviewer Note: Added clarification that the IUT can place restrictions on the instance and initial values allowed for creation. This change is not in any SSPC proposal.]

#### 9.16.1.4 Creating Objects by Specifying the Object Identifier and Providing Initial Values

Purpose: To verify the correct execution of the CreateObject service request when an Object Identifier is used as the object specifier and a list of initial property values is provided.

Test Steps:

1. TRANSMIT CreateObject-Request,  
     'Object Specifier' = (any unique object identifier of a type that is creatable *and an instance number that is creatable*)  
     'List Of Initial Values' = (a list of one or more properties and their initial values, that the IUT will accept)
2. RECEIVE CreateObject-ACK,  
     'Object Identifier' = (the object identifier specified in step 1)
3. REPEAT X = (properties initialized in the CreateObject-Request) DO {  
     VERIFY (the object identifier for the newly created object),  
     X = (the value specified in the 'List Of Initial Values' parameter of the CreateObject-Request)  
   }
4. VERIFY (the IUT's Device object), Object\_List = (any object list containing the newly created object)

#### 9.16.2 Negative CreateObject Service Execution Tests

[Change **Clause 9.16.2.4**, p. 386]

[Reviewer Note: Added Test Concept and Configuration Requirements.]

##### 9.16.2.4 Attempting to Create an Object with an Object ~~Type~~ Specifier and an Error in the Initial Values

Purpose: To verify the correct execution of the CreateObject service request when an object type is used as the object specifier and a list of initial property values containing an invalid value is provided.

*Test Concept: The TD shall attempt to create an object with an object type specifier and the 'List Of Initial Values' parameter containing a value which is out of range. The TD then attempts to create an object with a value of an inappropriate datatype in the 'List Of Initial Values' parameter. The selected datatype is not compliant with the property definition given by the BACnet standard.*

*Configuration Requirements: The value to be written shall not be of a datatype which is compliant with the property definition, but which is not supported for P1 by the IUT. For instance, Schedule\_Default, which is defined to be of Any primitive datatype, would not be used in this test along with BitString datatype, even where the IUT's Schedule object cannot be configured for scheduling BitString values.*

Test Steps:

1. READ X1 = Object\_List
2. TRANSMIT CreateObject-Request,  
     'Object ~~Type~~Specifier' = (any creatable object type),  
     'List Of Initial Values' = (a list of one or more properties and their initial values, that the IUT will accept initial values for, with one of the values being out of range)
3. IF (Protocol\_Revision is present and Protocol\_Revision ≥ 4) THEN

```

RECEIVE CreateObject-Error-PDU,
  'Error Class' =      PROPERTY,
  'Error Code' =      VALUE_OUT_OF_RANGE
  'First Failed Element Number' = (the position in the 'List Of Initial Values' with the offending value)
ELSE
  RECEIVE CreateObject-Error,
    'Error Class' =      PROPERTY,
    'Error Code' =      VALUE_OUT_OF_RANGE |
    OTHER
    'First Failed Element Number' = (the position in the 'List Of Initial Values' with the offending value)
4. CHECK (Verify that the new object was not created)
5. TRANSMIT CreateObject-Request,
  'Object Type Specifier' = (object type of step 2),
  'List Of Initial Values' = (a list of one or more properties and their initial values, that the IUT will
  accept initial values for, with one of the values being an inappropriate datatype)
6. IF (Protocol_Revision is present and Protocol_Revision ≥ 4) THEN
  RECEIVE
  CreateObject-Error,
    'Error Class' =      PROPERTY,
    'Error Code' =      INVALID_DATATYPE
    'First Failed Element Number' = (the position in the 'List Of Initial Values' with the offending value) |
  (BACnet-Reject-PDU
  Reject Reason =      INVALID_PARAMETER_DATATYPE | INVALID_TAG)
ELSE
  RECEIVE CreateObject-Error,
    'Error Class' =      PROPERTY,
    'Error Code' =      VALUE_OUT_OF_RANGE | INVALID_DATATYPE |
    OTHER
    'First Failed Element Number' = (the position in the 'List Of Initial Values' with the offending value) |
  (BACnet-Reject-PDU
  Reject Reason =      INVALID_PARAMETER_DATATYPE | INVALID_TAG)
7. READ X2 = Object_List
8. CHECK (X1=X2)

```

[Change **Clause 9.16.2.5**, p. 386]

[Reviewer Note: Added Test Concept and Configuration Requirements to clarify usage.]

### 9.16.2.5 Attempting to Create an Object with an Object Identifier Object Specifier and an Error in the Initial Values

**Purpose:** To verify the correct execution of the CreateObject service request when an object identifier is used as the object specifier and a list of initial property values containing an invalid value is provided.

*Test Concept: The TD shall attempt to create an object with an object type specifier and the 'List Of Initial Values' parameter containing a value which is out of range. The TD then attempts to create an object with a value of an inappropriate datatype in the 'List Of Initial Values' parameter. The selected datatype is not compliant with the property definition given by the BACnet standard.*

*Configuration Requirements: The value to be written shall not be of a datatype which is compliant with the property definition, but which is not supported for P1 by the IUT. For instance, Schedule\_Default, which is defined to be of Any primitive datatype, would not be used in this test along with BitString datatype, even where the IUT's Schedule object cannot be configured for scheduling BitString values.*

**Test Steps:**

1. TRANSMIT CreateObject-Request,
  - 'Object Identifier Specifier' = (any unique object identifier of a type that is creatable and an instance number that is creatable),
  - 'List Of Initial Values' = (a list of one or more properties and their initial values, that the IUT will

- accept initial values for, with one of the values being out of range)
2. IF (Protocol\_Revision is present and Protocol\_Revision  $\geq$  4) THEN  
RECEIVE CreateObject-Error-PDU,  
'Error Class' = PROPERTY,  
'Error Code' = VALUE\_OUT\_OF\_RANGE  
'First Failed Element Number' = (the position in the 'List Of Initial Values' with the offending value)  
ELSE  
RECEIVE CreateObject-Error,  
'Error Class' = PROPERTY,  
'Error Code' = VALUE\_OUT\_OF\_RANGE | OTHER  
'First Failed Element Number' = (the position in the 'List Of Initial Values' with the offending value)
  3. CHECK(Verify that the new object was not created)
  4. TRANSMIT CreateObject-Request,  
'Object Specifier' = (object identifier from step 1),  
'List Of Initial Values' = (a list of ~~two~~one or more properties and their initial values, that the IUT will accept initial values for, with one of the values being an inappropriate datatype)
  5. IF (Protocol\_Revision is present and Protocol\_Revision  $\geq$  4) THEN  
RECEIVE  
CreateObject-Error,  
'Error Class' = PROPERTY,  
'Error Code' = INVALID\_DATATYPE  
'First Failed Element Number' = (the position in the 'List Of Initial Values' with the offending value) |  
(BACnet-Reject-PDU  
Reject Reason = INVALID\_PARAMETER\_DATATYPE) |  
(BACnet-Reject-PDU  
Reject Reason = INVALID\_TAG)  
ELSE  
RECEIVE  
CreateObject-Error,  
'Error Class' = PROPERTY,  
'Error Code' = VALUE\_OUT\_OF\_RANGE | INVALID\_DATATYPE | OTHER  
'First Failed Element Number' = (the position in the 'List Of Initial Values' with the offendingvalue) |  
(BACnet-Reject-PDU  
Reject Reason = INVALID\_PARAMETER\_DATATYPE | INVALID\_TAG)
  6. TRANSMIT ReadProperty-Request,  
'Object Identifier' = (the 'Object Identifier' used in step 1),  
'Property Identifier' = Object\_Name
  7. IF (Protocol\_Revision is present and Protocol\_Revision  $\geq$  4) THEN  
RECEIVE BACnet-Error-PDU,  
'Error Class' = OBJECT,  
'Error Code' = UNKNOWN\_OBJECT  
ELSE  
RECEIVE BACnet-Error-PDU  
'Error Class' = OBJECT,  
'Error Code' = UNKNOWN\_OBJECT | NO\_OBJECTS\_OF\_SPECIFIED\_TYPE | OTHER

[Change **Clause 9.16.2.6**, p. 387]

[Reviewer Note: Corrected parameter for service request.]

#### 9.16.2.6 Attempting to Create an Object with an instance of 4194303

Purpose: This test case verifies the correct execution of the CreateObject service request when the 'Object Specifier' parameter conveys an object identifier with an instance of 4194303. This test shall be performed only if the Protocol\_Revision property is present in the Device object and has a value greater than or equal to 4.

Test Steps:

1. TRANSMIT CreateObject-Request,  
     'Object Identifier'/'Object Specifier' = (any object identifier representing a creatable object-type with an instance of 4194303)
2. RECEIVE BACnet-Reject-PDU,  
     'Reject Reason' = PARAMETER\_OUT\_OF\_RANGE

[Add new **Clause 9.16.2.X1**, p. 387]

#### 9.16.2.X1 Attempting to Create a non-Supported Object Type (by Object Type)

Purpose: To verify the correct execution of the CreateObject service request when the 'Object Specifier' parameter conveys an object type that is not supported in the IUT.

Test Steps:

1. TRANSMIT CreateObject-Request,  
     'Object Specifier' = (any unsupported object type)
2. IF (Protocol\_Revision >= 10) THEN  
     RECEIVE CreateObject-Error,  
         'Error Class' = OBJECT,  
         'Error Code' = UNSUPPORTED\_OBJECT\_TYPE  
         'First Failed Element Number' = 0.  
     ELSE  
         RECEIVE CreateObject-Error,  
         'Error Class' = (any valid error class),  
         'Error Code' = (any valid error code)  
         'First Failed Element Number' = 0
3. VERIFY (the IUT's Device object),  
     Object\_List = (any object list that does not contain the object specified in step 1)

[Add new **Clause 9.16.2.X2**, p. 387]

#### 9.16.2.X2 Attempting to Create a non-Supported Object Type (by Object Identifier)

Purpose: To verify the correct execution of the CreateObject service request when the 'Object Specifier' parameter conveys an object identifier for an object type that is not supported in the IUT.

Notes to Tester: If the IUT limits the instances that can be created, this shall be taken into account when selecting an object identifier in step 1.

Test Steps:

1. TRANSMIT CreateObject-Request,  
     'Object Specifier' = (any object identifier having an unsupported object type)
2. IF (Protocol\_Revision >= 10) THEN  
     RECEIVE CreateObject-Error,  
         'Error Class' = OBJECT,  
         'Error Code' = UNSUPPORTED\_OBJECT\_TYPE  
         'First Failed Element Number' = 0  
     ELSE  
         RECEIVE CreateObject-Error,  
         'Error Class' = (any valid error class),  
         'Error Code' = (any valid error code)  
         'First Failed Element Number' = 0
3. VERIFY (the IUT's Device object),  
     Object\_List = (any object list that does not contain the object specified in step 1)

## 9.17 DeleteObject Service Execution Tests

### 9.17.2 Negative DeleteObject Service Execution Tests

[Change **Clause 9.17.2.1**, p. 388]

[Reviewer Note: Modified test to remove dependency on EPICS values.]

#### 9.17.2.1 Attempting to Delete an Object That is Not Deletable

Purpose: To verify the correct response to an attempt to delete an object that is not deletable.

Configuration Requirements: The IUT shall be configured with an object X that cannot be deleted.

Test Steps:

1. *READ VI = Object\_Name*
2. TRANSMIT DeleteObject-Request,  
    'Object Identifier' = X
3. RECEIVE BACnet-Error-PDU,  
    'Error Class' = OBJECT,  
    'Error Code' = OBJECT\_DELETION\_NOT\_PERMITTED
4. VERIFY (X), Object\_Name = *VI* (~~the Object\_Name specified in the EPICS~~)
5. VERIFY (X), Object\_List = (any object list that contains X)

## 9.18 ReadProperty Service Execution Tests

### 9.18.1 Positive ReadProperty Service Execution Tests

[Change **Clause 9.18.1.2**, p. 389]

[Reviewer Note: Modified test to remove dependency on EPICS values.]

#### 9.18.1.2 Reading a Single Element of an Array

Purpose: To verify that the IUT can execute ReadProperty service requests when the requested property is an array, and a single element of the array is requested.

Test Steps:

1. *READ V = (Device, X), Object\_List ARRAY INDEX=1*
2. *CHECK (V is of type object-identifier)*
1. ~~VERIFY (Device, X),~~  
    Object\_List = (the first element of the Object\_List array as specified in the EPICS),  
    ~~ARRAY INDEX=1~~

~~Passing Result: The returned value should be of type object identifier.~~

[Add new **Clause 9.18.1.X1**, p. 390]

[Reviewer Note: A general ReadProperty test is not supplied by 135.1 that can be used in a variety of situations. Added a test to ensure that all data types are tested.]

#### 9.18.1.X1 Reading Properties Based on Data Type

Purpose: This test case verifies that the IUT can execute ReadProperty service requests for requested properties of each of the supported base data types.



**Test Concept:** This test is repeated once for each base data type that the IUT supports. For each execution of the test a property, P1, shall be selected that is of the data type being tested and the object containing P1 is designated Object1 in the test description.

**Test Steps:**

1. READ V = (Object1), P1
2. CHECK (V returns any valid value of the correct data type for property P1)

[Add new **Clause 9.18.1.X3**, p. 390]

#### **9.18.1.X3 Respects max-segments-accepted bit pattern**

**Purpose:** To verify that the IUT abides by the 'max-segments-accepted' parameter, when the size of the response does require segmentation.

**Configuration Requirements:** Use a very small 50 octet 'max-APDU-length-accepted' size in the request. The BACnet-Confirmed-Request-PDU shall be one where the response size will exceed 2 times 'max-APDU-length-accepted' and so require at least three segments. If the largest response that the IUT can return is 100 or fewer octets, then this test shall be skipped.

**Test Steps:**

1. TRANSMIT BACnet-Confirmed-Request-PDU,  
    'segmented-response-accepted' = TRUE  
    'max-segments-accepted' = 2
2. RECEIVE BACnet-Abort-PDU,  
    'Abort Reason' = BUFFER\_OVERFLOW

**Hints to Tester:** An attempt to read the whole Object\_List might suffice. Or a ReadRange or ReadPropertyMultiple or AtomicReadFile request, if any of those services are executed.

[Add new **Clause 9.18.1.X4**, p. 390]

#### **9.18.1.X4 Reading Array Properties at Different Array Indexes**

**Purpose:** This test verifies the IUT can execute ReadProperty service requests on a single element in an array property.

**Test Concept:** This test will execute a ReadProperty service request to read a single element from the selected property by specifying the array-index in the request. Another request is made to read an element of an array where the array index is out of range.

**Configuration Requirement:** O1 is any object in the IUT database having array property P1 having size X.

**Test Steps:**

1. VERIFY P1 = X, ARRAY INDEX = 0
2. IF (X>0) THEN  
    READ V = P1, ARRAY INDEX = 1  
    CHECK (V is any valid value of the correct data type for property P1)  
    READ V = P1, Array Index =X  
    CHECK (V is any valid value of the correct data type for property P1)
3. TRANSMIT ReadProperty-Request,  
    'Object Identifier' = O1,  
    'Property Identifier' = P1  
    'Property Array Index' = (X+1)
4. RECEIVE BACnet-Error-PDU,  
    'Error Class' = PROPERTY,  
    'Error Code' = INVALID\_ARRAY\_INDEX

[Add new **Clause 9.18.1.X5**, p. 390]

### **9.18.1.X5 ReadProperty of the Network Port Object using the Unknown Instance**

Purpose: Verify that the IUT selects the correct object when a Network Port is read using the special object instance 4194303.

Test Concept: Execute a ReadProperty service request specifying 'Object Identifier' = (Network Port, 4194303). Verify that the responding BACnet-user selects the local Network Port object representing the network port through which the request was received.

Configuration Requirements: Let X be the instance number of the Network Port object associated with the network port through which the TD will communicate with the IUT.

Test Steps:

1. TRANSMIT ReadProperty-Request,  
    'Object Identifier' = (Network Port, 4194303),  
    'Property Identifier' = Object-Identifier
2. RECEIVE ReadProperty-ACK,  
    'Object Identifier' = (Network Port, X),  
    'Property Identifier' = Object-Identifier,  
    'Property Value' = (Network Port, X)
3. REPEAT P = (each property in the specified Network Port object) {  
    TRANSMIT ReadProperty-Request through the same port as above,  
    'Object Identifier' = (Network Port, 4194303),  
    'Property Identifier' = P  
    RECEIVE ReadProperty-ACK,  
    'Object Identifier' = (Network Port, X),  
    'Property Identifier' = P,  
    'Property Value' = V  
    VERIFY (Network Port, X), P = V  
}

[Add new **Clause 9.18.1.X8**, p. 390]

### **9.18.1.X8 ReadProperty Service when Non-BACnet Device Offline**

Purpose: To verify that the ReadProperty service executes successfully when a non-BACnet device is offline.

Test Concept: Object1 is an object which contains information from a non-BACnet device. The non-BACnet device is verified to be online and recognized by the IUT. It is then made to go offline, and the IUT is made to recognize that the device is offline. A property, P1, from Object1 which contains a dynamic value derived from the data in the non-BACnet device is read from the IUT.

Test Steps:

1. CHECK (any vendor-specified indication, that the non-BACnet device is online)
2. MAKE (the non-BACnet device go offline)
3. MAKE (the IUT notice that the non-BACnet device is offline)
4. TRANSMIT ReadProperty Request,  
    'Object Identifier' = Object1,  
    'Property Identifier' = P1
5. RECEIVE ReadProperty-ACK,  
    'Object Identifier' = Object1,  
    'Property Identifier' = P1,  
    'Property Value' = (V, any valid value)

## 9.20 ReadPropertyMultiple Service Execution Tests

### 9.20.1 Positive ReadPropertyMultiple Service Execution Tests

[Change **Clause 9.20.1.1**, p. 392]

[Reviewer Note: Modified test to remove dependency on EPICS values.]

#### 9.20.1.1 Reading a Single Property from a Single Object

Purpose: To verify the ability to read a single property from a single object.

Test Concept: A single supported property is read from the Device object. The property is selected by the TD and is designated as P1 in the test description.

Test Steps:

1. TRANSMIT ReadPropertyMultiple-Request,  
    'Object Identifier' = Object1 | Object2,  
    'Property Identifier' = P1
2. RECEIVE ReadPropertyMultiple-ACK,  
    'Object Identifier' = (the object selected in step 1),  
    'Property Identifier' = P1,  
    'Property Value' = (*any valid value*~~the value of P1 specified in the EPICS~~)

[Change **Clause 9.20.1.2**, p. 393]

[Reviewer Note: Modified test to remove dependency on EPICS values.]

#### 9.20.1.2 Reading Multiple properties from a Single Object

Purpose: To verify the ability to read multiple properties from a single object.

Test Steps:

1. TRANSMIT ReadPropertyMultiple-Request,  
    'Object Identifier' = Object1 | Object 2,  
    'Property Identifier' = P1,  
    'Property Identifier' = P2  
-- ... (Two properties are required but more may be selected.)
2. RECEIVE ReadPropertyMultiple-ACK,  
    'Object Identifier' = (the object selected in step 1),  
    'Property Identifier' = P1,  
    'Property Value' = (*any valid value for P1*~~the value of P1 specified in the EPICS~~),  
    'Property Identifier' = P2,  
    'Property Value' = (*any valid value for P2*~~the value of P2 specified in the EPICS~~)  
-- ... (An appropriate value must be returned for each property included in the ReadPropertyMultiple-Request.)

[Change **Clause 9.20.1.3**, p. 393]

[Reviewer Note: Modified test to remove dependency on EPICS values.]

#### 9.20.1.3 Reading a Single Property from Multiple Objects

Purpose: To verify the ability to read a single property from multiple objects.

#### Test Steps:

1. TRANSMIT ReadPropertyMultiple-Request,  
     'Object Identifier' = Object1,  
     'Property Identifier' = P1,  
     'Object Identifier' = Object2,  
     'Property Identifier' = P2  
   -- ... (Two properties are required but more may be selected.)
2. RECEIVE ReadPropertyMultiple-ACK,  
     'Object Identifier' = Object1,  
     'Property Identifier' = P1,  
     'Property Value' = (any valid value for P1~~the value of P1 specified in the EPICS~~),  
     'Object Identifier' = Object2,  
     'Property Identifier' = P2,  
     'Property Value' = (any valid value for P2~~the value of P2 specified in the EPICS~~)  
   -- ... (An appropriate value must be returned for each property included in the ReadPropertyMultiple-Request.)

[Change **Clause 9.20.1.4**, p. 393]

[Reviewer Note: Modified test to remove dependency on EPICS values.]

#### 9.20.1.4 Reading Multiple Properties from Multiple Objects

Purpose: To verify the ability to read multiple properties from multiple objects.

#### Test Steps:

1. TRANSMIT ReadPropertyMultiple-Request,  
     'Object Identifier' = Object1,  
     'Property Identifier' = P1,  
     'Property Identifier' = P2,  
     'Property Identifier' = P3,  
     'Object Identifier' = Object2,  
     'Property Identifier' = P4,  
     'Property Identifier' = P5,  
     'Property Identifier' = P6  
   -- ... (Two objects must be included but but more may be selected.)
2. RECEIVE ReadPropertyMultiple-ACK,  
     'Object Identifier' = Object1,  
     'Property Identifier' = P1,  
     'Property Value' = (any valid value for P1~~the value of P1 specified in the EPICS~~),  
     'Property Identifier' = P2,  
     'Property Value' = (any valid value for P2~~the value of P2 specified in the EPICS~~),  
     'Property Identifier' = P3,  
     'Property Value' = (any valid value for P3~~the value of P3 specified in the EPICS~~),  
     'Object Identifier' = Object2,  
     'Property Identifier' = P4,  
     'Property Value' = (any valid value for P4~~the value of P4 specified in the EPICS~~),  
     'Property Identifier' = P5,  
     'Property Value' = (any valid value for P5~~the value of P5 specified in the EPICS~~),  
     'Property Identifier' = P6,  
     'Property Value' = (any valid value for P6~~the value of P6 specified in the EPICS~~)  
   -- ... (An appropriate value must be returned for each property included in the ReadPropertyMultiple-Request.)

[Change **Clause 9.20.1.5**, p. 394]

[Reviewer Note: Modified test to remove dependency on EPICS values.]

### 9.20.1.5 Reading Multiple Properties with a Single Embedded Access Error

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

Test Steps:

1. TRANSMIT ReadPropertyMultiple-Request,
  - 'Object Identifier' = Object1,
  - 'Property Identifier' = P1,
  - 'Property Identifier' = P2,
  - 'Property Identifier' = (any property, P3, not supported in this object),
  - 'Property Identifier' = P4
2. RECEIVE ReadPropertyMultiple-ACK,
  - 'Object Identifier' = Object1,
  - 'Property Identifier' = P1,
  - 'Property Value' = (any valid value for P1~~the value of P1 specified in the EPICS~~),
  - 'Property Identifier' = P2,
  - 'Property Value' = (any valid value for P2~~the value of P2 specified in the EPICS~~),
  - 'Property Identifier' = P3,
  - 'Error Class' = PROPERTY,
  - 'Error Code' = UNKNOWN\_PROPERTY,
  - 'Property Identifier' = P4,
  - 'Property Value' = (any valid value for P4~~the value of P4 specified in the EPICS~~)

[Change **Clause 9.20.1.6**, p. 394]

[Reviewer Note: Modified test to remove dependency on EPICS values.]

### 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' = Object1,
  - '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, Object2, which is of a type supported by the IUT),
  - 'Property Identifier' = P5,
  - 'Property Identifier' = P6
2. RECEIVE ReadPropertyMultiple-ACK,
  - 'Object Identifier' = Object1,
  - 'Property Identifier' = P1,
  - 'Property Value' = (any valid value for P1~~the value of P1 specified in the EPICS~~),
  - 'Property Identifier' = P2,
  - 'Property Value' = (any valid value for P2~~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' = Object2,  
 'Property Identifier' = P5,  
 'Error Class' = OBJECT,  
 'Error Code' = (UNKNOWN\_OBJECT),  
 'Property Identifier' = P6,  
 'Error Class' = OBJECT,  
 'Error Code' = (UNKNOWN\_OBJECT)

[Change **Clause 9.20.1.7**, p. 395]

[Reviewer Note: Modified test to remove dependency on EPICS values.]

### 9.20.1.7 Reading ALL Properties

Purpose: To verify the ability to correctly execute a ReadPropertyMultiple service request that uses the special property identifier ALL. One instance of each object-type supported is tested.

*Notes to Tester: If a property which is not readable using the ReadPropertyMultiple service is in the specified object, and Protocol\_Revision < 7, then either no entry is returned, or an error code is returned. If Protocol\_Revision ≥ 7, then the entry shall contain 'Error Class': PROPERTY and 'Error-Code': READ\_ACCESS\_DENIED for that property. Property\_List (371) shall not appear in the List of Results.*

Test Steps:

```
1. REPEAT ObjectX = (one instance of each supported object type) DO {
    TRANSMIT ReadPropertyMultiple-Request,
        'Object Identifier' = ObjectX,
        'Property Identifier' = ALL
    RECEIVE ReadPropertyMultiple-ACK,
        'Object Identifier' = ObjectX,
        'List Of Results' = (a list of all standard properties
                           documented for ObjectX in the EPICS plus all proprietary
                           properties present in ObjectX, each with a valid
                           value, excluding the Property_List property)
    REPEAT P = (each property supported by Object1) DO {
        'Property Identifier' = P,
        'Property Value' = (the value of P specified in the EPICS)
    }
}
```

~~Notes to Tester: Any proprietary properties that are supported for the object type shall also be returned (see BACnet 15.7.3.1.2). If a property which is not readable using the ReadPropertyMultiple service is in the specified object, and Protocol\_Revision < 7, then either no entry is returned, or an error code is returned. If Protocol\_Revision ≥ 7, then the entry shall contain 'Error Class': PROPERTY and 'Error Code': READ\_ACCESS\_DENIED for that property. Property\_List (371) shall not appear in the List of Results.~~

[Change **Clause 9.20.1.8**, p. 395]

[Reviewer Note: Modified test to remove dependency on EPICS values. Added language to clarify the properties required to be returned.]

### 9.20.1.8 Reading OPTIONAL Properties

Purpose: To verify the ability to correctly execute a ReadPropertyMultiple service request that uses the special property identifier OPTIONAL *by returning all of the object's optional properties. One instance of each object type supported is tested. The property identifier OPTIONAL means that only those standard properties present in the object that have a conformance code "O" shall be returned.*

*Notes to Tester: If no optional properties are supported, then an empty 'List of Results' shall be returned for the specified property. If a property which is not readable using the ReadPropertyMultiple service is in the specified object, and Protocol\_Revision < 7, then either no entry is returned, or an error code is returned. If Protocol\_Revision >= 7, then the entry shall contain Error Class: PROPERTY and 'Error-Code': READ\_ACCESS\_DENIED for that property.*

Test Steps:

```
1. REPEAT ObjectX = (one instance of each supported object type) DO {
    TRANSMIT ReadPropertyMultiple-Request,
        'Object Identifier' = Object1ObjectX,
        'Property Identifier' = OPTIONAL
    RECEIVE ReadPropertyMultiple-ACK,
        'Object Identifier' = Object1ObjectX,
        'List Of Results' = (a list of all standard properties with a conformance code
                           of O documented for ObjectX in the EPICS, each with a valid
                           value)
    REPEAT P = (each optional property supported by Object1ObjectX) DO {
    'Property Identifier' = P,
    'Property Value' = (any valid value for Pthe value of P specified in the EPICS)
}
}
```

~~Notes to Tester: If no optional properties are supported then an empty 'List of Results' shall be returned for the specified property.~~

[Change **Clause 9.20.1.9**, p. 396]

[Reviewer Note: Modified test to remove dependency on EPICS values. Added language to clarify the properties required to be returned.]

### 9.20.1.9 Reading REQUIRED Properties

Purpose: To verify the ability to correctly execute a ReadPropertyMultiple service request that uses the special property identifier REQUIRED *by returning all of the object's required properties. One instance of each object type supported is tested. The property identifier REQUIRED means that only those standard properties having a conformance code of "R" or "W" shall be returned.*

*Notes to Tester: If a property which is not readable using the ReadPropertyMultiple service is in the specified object, and Protocol\_Revision < 7, then either no entry is returned, or an error code is returned. If Protocol\_Revision >= 7, then the entry shall contain 'Error Class': PROPERTY and 'Error-Code': READ\_ACCESS\_DENIED for that property. Property\_List (371) shall not appear in the List of Results.*

Test Steps:

```
1. REPEAT ObjectX = (one instance of each supported object type) DO {
    TRANSMIT ReadPropertyMultiple-Request,
        'Object Identifier' = ObjectX,
        'Property Identifier' = REQUIRED
    RECEIVE ReadPropertyMultiple-ACK,
        'Object Identifier' = ObjectX,
```

*'List Of Results' = (a list of all standard properties with a conformance code of R or W documented for ObjectX in the EPICS, each with a valid value, excluding the Property\_List property)*

```

REPEAT P = (each property supported by Object1) DO {
  'Property Identifier' = P,
  'Property Value' = (the value of P specified in the EPICS)
}
}

```

~~Notes to Tester: If a property which is not readable using the ReadPropertyMultiple service is in the specified object, and Protocol\_Revision < 7, then either no entry is returned, or an error code is returned. If Protocol\_Revision ≥ 7, then the entry shall contain 'Error Class': PROPERTY and 'Error Code': READ\_ACCESS\_DENIED for that property. Property\_List (371) shall not appear in the List of Results.~~

[Add new **Clause 9.20.1.X1**, p. 398]

[Reviewer Note: A general ReadPropertyMultiple test is not supplied by 135.1 that can be used in a variety of situations.]

### 9.20.1.X1 Reading Properties Based on Data Type

Purpose: This test case verifies that the IUT can execute ReadPropertyMultiple service requests for requested properties of each of the supported base data types.

Test Concept: The test 9.18.1.X1 Reading Properties Based on Data Type is repeated using ReadPropertyMultiple instead of ReadProperty.

[Add new **Clause 9.20.1.X2**, p. 398]

### 9.20.1.X2 ReadPropertyMultiple Array Properties

Purpose: To verify that the IUT can execute ReadPropertyMultiple service requests when the requested property is an array, when its size as well as when a single element of the array is requested. Another request is made to read an element of an array where the array index is out of range.

Configuration Requirement: O1 is any object in the IUT database having array property P1 having size X.

Test Steps:

1. VERIFY P1 = X, ARRAY INDEX = 0
2. IF (X>0) THEN
3.     TRANSMIT ReadPropertyMultiple-Request,
  - 'Object Identifier' = O1,
  - 'Property Identifier' = P1,
  - 'Property Array Index' = 1
4.     RECEIVE ReadPropertyMultiple-ACK,
  - 'Object Identifier' = O1,
  - 'Property Identifier' = P1,
  - 'Property Array Index' = 1,
  - 'Property Value' = (V, any valid value of the correct data type for property P1)
5.     TRANSMIT ReadPropertyMultiple-Request,
  - 'Object Identifier' = O1,
  - 'Property Identifier' = P1,
  - 'Property Array Index' = X,
6.     RECEIVE ReadPropertyMultiple-ACK,
  - 'Object Identifier' = O1,
  - 'Property Identifier' = P1,
  - 'Property Array Index' = X,
  - 'Property Value' = (V, any valid value of the correct data type for property P1)



7. CHECK (V is any valid value of the correct data type for property P1)
8. TRANSMIT ReadPropertyMultiple-Request,  
     'Object Identifier' = O1,  
     'Property Identifier' = P1,  
     'Property Array Index' = (X+1)
9. RECEIVE ReadPropertyMultiple-Error,  
     'Error Class' = PROPERTY,  
     'Error Code' = INVALID\_ARRAY\_INDEX

[Add new **Clause 9.20.1.X3**, p. 398]

### **9.20.1.X3 ReadPropertyMultiple of the Network Port Object using the Unknown Instance**

Purpose: Verify that the IUT selects the correct object when a Network Port is read using the special object instance 4194303.

Test Concept: Execute a ReadPropertyMultiple service request specifying 'Object Identifier' = (Network Port, 4194303). The responding BACnet-user selects the local Network Port object representing the network port through which the request was received.

Configuration Requirements: Let X be the instance number of the Network Port object associated with the network port through which the TD will communicate with the IUT.

Test Steps:

1. TRANSMIT ReadPropertyMultiple-Request,  
     'Object Identifier' = (Network Port, 4194303),  
     'Property Identifier' = Object-Identifier
2. RECEIVE ReadPropertyMultiple-ACK,  
     'Object Identifier' = (Network Port, X),  
     'Property Identifier' = Object-Identifier,  
     'Property Value' = (Network Port, X)
3. REPEAT P = (each property in the specified Network Port object) {  
     TRANSMIT ReadPropertyMultiple-Request through the same port as above,  
     'Object Identifier' = (Network Port, 4194303),  
     'Property Identifier' = P  
     RECEIVE ReadPropertyMultiple-ACK,  
     'Object Identifier' = (Network Port, X),  
     'Property Identifier' = P,  
     'Property Value' = V  
     VERIFY (Network Port, X), P = V  
   }

[Add new **Clause 9.20.1.X9**, p. 398]

### **9.20.1.X9 ReadPropertyMultiple Service when Non-BACnet Device Offline**

Purpose: To verify that the ReadPropertyMultiple service executes successfully when a non-BACnet device is offline.

Test Concept: Object1 is an object which contains information from a non-BACnet device. The non-BACnet device is verified to be online and recognized by the IUT. It is then made to go offline, and the IUT is made to recognize that the device is offline. A property, P1, from Object1 which contains a dynamic value derived from the data in the non-BACnet device is read from the IUT.

Test Steps:

1. CHECK (any vendor-specified indication, that the non-BACnet device is online)
2. MAKE (the non-BACnet device go offline)
3. MAKE (the IUT notice that the non-BACnet device is offline)

4. TRANSMIT ReadPropertyMultiple-Request,  
'Object Identifier' = Object1,  
'Property Identifier' = P1
5. RECEIVE ReadPropertyMultiple-ACK,  
'Object Identifier' = Object1,  
'Property Identifier' = P1,  
'Property Value' = (any valid value)

[Change **Clause 9.21**, p. 400]

## 9.21 ReadRange Service Execution Tests

This clause defines the tests necessary to demonstrate support for executing ReadRange service requests.

~~Dependencies: None.~~

~~BACnet Reference Clause: 15.8.~~

Configuration Requirements: *When testing a Log\_Buffer property, the IUT shall be configured with a Trend Log logging object that contains a set of known log records.* ~~The IUT shall be configured with a Trend Log logging object that contains a set of known log records.~~ ~~The TD must have exact knowledge of the trend data log records in order to evaluate the results of the tests. The value of the Enable property shall be FALSE so that the Log\_Buffer does not change during the tests.~~

*When testing a property other than the Log\_Buffer, steps shall be taken to ensure that the value of the property does not change outside the control of the tester during the execution of the test.*

The following sample log buffer is used as explanation for the tests in this section.

**Sample Log\_Buffer, (Trend Log, Instance 1)**

Arbitrary Record Designation	Position (index)	Implied Sequence #	Timestamp (Date excluded for clarity)	LogDatum
a	1	16	13:01:00.00	log-status, buffer-purged
b	2	17	13:02:00.00	log-status, log-disabled = FALSE
c	3	18	13:05:00.00	real-value = 5.0
d	4	19	13:10:00.00	real-value = 10.0
e	5	20	13:15:00.00	real-value = 15.0
f	6	21	13:16:00.00	log-status, log-disabled = TRUE
g	7	22	13:21:00.00	log-status, log-disabled = FALSE
h	8	23	13:25:00.00	real-value = 25.0
i	9	24	13:30:00.00	real-value = 30.0
j	10	25	13:35:00.00	real-value = 35.0
k	11	26	13:36:00.00	log-status, log-disabled = TRUE

### 9.21.1 Positive ReadRange Service Execution Tests

[Change **Clause 9.21.1.1**, p. 401]

[Reviewer Note: Made the test applicable to object types other than trends.]

### 9.21.1.1 Reading All Items in the List

Purpose: To verify that the IUT correctly responds to a ReadRange service request to return all of the available data items.

*Test Concept: A list property, P is read using ReadRange by position with no range specified. It is verified that the complete list is returned.*

*Configuration Requirements: Property P is configured with a value that is small enough to be returned in a single ReadRange response. If the IUT cannot be configured in this manner, see the Notes to Tester.*

*Notes to Tester: The property P may have more items than can be returned in a single message. Under these circumstances 'Result Flags' will have the value {TRUE, FALSE, TRUE} and the 'Item Count' and 'Item Data' parameters would reflect the actual number of items that were able to be returned.*

Test Steps:

1. TRANSMIT ReadRange-Request,
  - 'Object Identifier' = (the log object configured for this test),
  - 'Property Identifier' = ~~Log\_Buffer~~ P
2. RECEIVE Read-Range-ACK,
  - 'Object Identifier' = (the log object configured for this test),
  - 'Property Identifier' = Log\_Buffer,
  - 'Result Flags' = {TRUE, TRUE, FALSE},
  - 'Item Count' = (the number of ~~entries in P~~ trend records in the test object),
  - 'Item Data' = (all of the ~~entries in P~~ trend records in the test object)

~~Notes to Tester: The trend data may have more items than can be returned in a single message. Under these circumstances 'Result Flags' will have the value {TRUE, FALSE, TRUE} and the 'Item Count' and 'Item Data' parameters would reflect the actual number of items that were able to be returned.~~

[Change **Clause 9.21.1.2**, p. 401]

[Reviewer Note: Make the test applicable to object types other than trends.]

### 9.21.1.2 Reading Items by Position with Positive Count

Purpose: To verify that the IUT correctly responds to a ReadRange service request to return items specified by indicating a position and the number of items after that position to return.

~~Test Concept: A ReadRange request is transmitted by the TD requesting a range of items known to be in the list property P the Log\_Buffer. This range is specified using the 'By Position' option and a positive value for 'Count'. The 'Reference Index' and 'Count' are selected so that the results can be conveyed in a single acknowledgment.~~

*Configuration Requirements: A list property, P, is configured with N items.*

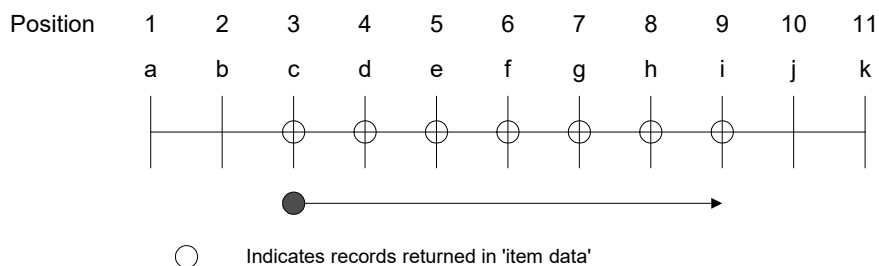
Test Steps:

1. TRANSMIT ReadRange-Request,
  - 'Object Identifier' = (the log object configured for this test),
  - 'Property Identifier' = ~~Log\_Buffer~~ P,
  - 'Reference Index' = (any value x:  $1 \leq x \leq \text{Record\_Count } N$ ),
  - 'Count' = (any value y ~~x~~:  $0 < y \leq \text{Record\_Count } N - x + 1$ )
2. RECEIVE Read-Range-ACK,
  - 'Object Identifier' = (the log object configured for this test),
  - 'Property Identifier' = ~~Log\_Buffer~~ P,
  - 'Result Flags' = {?, ?, FALSE},
  - 'Item Count' = y (the same value used in the 'Count' parameter in step 1),
  - 'Item Data' = (all of the items specified trend records in order of increasing position. The items specified include

the item at the index specified by x, plus (y-1) items following.)

Test Example (using the sample buffer at beginning of section):

1. TRANSMIT ReadRange-Request,
  - 'Object Identifier' = (Trend Log, Instance 1),
  - 'Property Identifier' = Log\_Buffer,
  - 'Reference Index' = 3,
  - 'Count' = 7
2. RECEIVE ReadRange-ACK,
  - 'Object Identifier' = (Trend Log, Instance 1),
  - 'Property Identifier' = Log\_Buffer,
  - 'Result Flags' = {FALSE, FALSE, FALSE},
  - 'Item Count' = 7,
  - 'Item Data' = Records < c, d, e, f, g, h, i > in that order.



[Change **Clause 9.21.1.3**, p. 402]

[Reviewer Note: Make the test applicable to object types other than trends.]

### 9.21.1.3 Reading Items by Position with Negative Count

Purpose: To verify that the IUT correctly responds to a ReadRange service request to return items specified by indicating a position and the number of items before that position to return.

Test Concept: A ReadRange request is transmitted by the TD requesting a range of items known to be in *the list property P* the ~~Log\_Buffer~~. This range is specified using the 'By Position' option and a negative value for 'Count'. The 'Reference Index' and 'Count' are selected so that the results can be conveyed in a single acknowledgement.

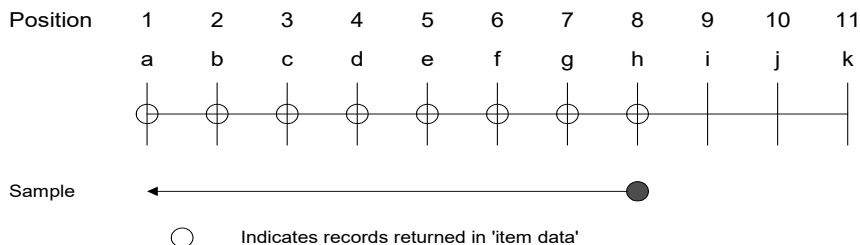
*Configuration Requirements: A list property, P, is configured with N items.*

Test Steps:

1. TRANSMIT ReadRange-Request,
  - 'Object Identifier' = (the ~~log~~ object configured for this test),
  - 'Property Identifier' = ~~Log\_Buffer~~ P,
  - 'Reference Index' = (any value x:  $1 \leq x \leq N$  ~~Record\_Count~~),
  - 'Count' = (any value y:  $y < 0$  AND  $|y| \leq x$ )
2. RECEIVE ReadRange-ACK,
  - 'Object Identifier' = (the ~~log~~ object configured for this test),
  - 'Property Identifier' = ~~Log\_Buffer~~ P,
  - 'Result Flags' = {?, ?, FALSE},
  - 'Item Count' =  $|y|$ ,
  - 'Item Data' = (all of the *items* specified ~~trend records~~ in order of increasing position. The items specified include the item at the index specified by x, plus  $|y|-1$  items preceding.)

Test Example (using the sample buffer at beginning of section):

1. TRANSMIT ReadRange-Request,  
     'Object Identifier' = (Trend Log, Instance 1),  
     'Property Identifier' = Log\_Buffer,  
     'Reference Index' = 8,  
     'Count' = -8
2. RECEIVE ReadRange-ACK,  
     'Object Identifier' = (Trend Log, Instance 1),  
     'Property Identifier' = Log\_Buffer,  
     'Result Flags' = {TRUE, FALSE, FALSE},  
     'Item Count' = 8  
     'Item Data' = Records < a, b, c, d, e, f, g, h > in that order.



[Remove **Clause 9.21.1.5**, p. 405]

#### **9.21.1.5 Reading Items by Time Range**

**Purpose:** To verify that the IUT correctly responds to a ReadRange service request to return items specified by indicating a range of times that are to be included.

**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 the "Time Range" option. The 'Beginning Time' and 'Ending Time' are selected so that the results can be conveyed in a single acknowledgement.

**Test Steps:**

1. TRANSMIT ReadRange Request,  
     'Object Identifier' = (the Trend Log object configured for this test),  
     'Property Identifier' = Log\_Buffer,  
     'Beginning Time' = (any value before the last time in the buffer),  
     'Ending Time' = (any value > 'Beginning Time')
2. RECEIVE Read Range ACK,  
     'Object Identifier' = (the Trend Log object configured for this test),  
     'Property Identifier' = Log\_Buffer,  
     'Result flags' = {TRUE, TRUE, FALSE},  
     'Item Count' = (the number of trend records meeting the specified criteria),  
     'Item Data' = (all of the specified trend records)

**Notes to Tester:** The first item returned shall be the first one in the buffer that has a timestamp newer (later time) than the time specified by the 'Beginning Time' parameter. The last item returned shall be the one with a timestamp older (earlier time) than or equal to the one specified by the 'Ending Time' parameter

[Change and renumber **Clause 9.21.1.6**, p. 405]

[Reviewer Note: Make the test applicable to object types other than trends.]

#### **9.21.1.6 9.21.1.5 Reading a Range of Items that do not Exist by Position**

**Purpose:** To verify that the IUT correctly responds to a ReadRange service request when there are no items within the specified by position range.

Test Concept: A ReadRange request is transmitted by the TD requesting a range of items known not to be in the *list property P Log\_Buffer*. The IUT shall respond by returning an empty list.

*Configuration Requirements: The list property, P, is configured with N items.*

Test Steps:

1. TRANSMIT ReadRange-Request,
  - 'Object Identifier' = (the ~~Trend Log~~ log object configured for this test),
  - 'Property Identifier' = ~~P Log\_Buffer~~,
  - 'Reference Index' = (any value  $x$ :  $x > N$ ),
  - 'Count' = (any value  $y$ :  $y > 0$ )
  - ~~'Beginning Time' = (any value that will result in a time interval for which there are no items present),~~
  - ~~'Ending Time' = (any value that will result in a time interval for which there are no items present)~~
2. RECEIVE Read-Range-ACK,
  - 'Object Identifier' = (the ~~Trend Log~~ log object configured for this test),
  - 'Property Identifier' = ~~P Log\_Buffer~~,
  - 'Result flags' = {TRUE, TRUE, FALSE},
  - 'Item Count' = 0,
  - 'Item Data' = (an empty list)

[Renumber **Clause 9.21.1.7**, p. 406]

#### **9.21.1.7.9.21.1.6 Reading a Range of Items that do not Exist (Using by Sequence)**

[Renumber **Clause 9.21.1.8**, p. 406]

#### **9.21.1.8.9.21.1.7 Reading a Range of Items that do Not Exist (Using by Time)**

[Change and renumber **Clause 9.21.1.9**, p. 407]

[Reviewer Note: Make the test applicable to object types other than trends.]

#### **9.21.1.9.9.21.1.8 Reading Items by Sequence with Positive Count**

Purpose: To verify that the IUT correctly responds to a ReadRange service request to return items specified by indicating a sequence number and the number of items after that sequence to return.

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 the 'By Sequence' option and a positive value for 'Count'. The 'Reference Sequence Number' and 'Count' are selected so that the results can be conveyed in a single acknowledgment.

Test Steps:

1. TRANSMIT ReadRange-Request,
  - 'Object Identifier' = (the log object configured for this test),
  - 'Property Identifier' = Log\_Buffer,
  - 'Reference Sequence Number' = (any value  $x$ :  $(\text{Total\_Record\_Count} - \text{Record\_Count} + 1) \leq x \leq (\text{Total\_Record\_Count} - y + 1)$ ),
  - 'Count' = (any value  $y$ :  $0 < y \leq \text{Record\_Count}$ )
2. RECEIVE ReadRange-ACK,
  - 'Object Identifier' = (the log object configured for this test),
  - 'Property Identifier' = Log\_Buffer,
  - 'Result Flags' = {?, ?, FALSE},
  - 'Item Count' =  $y$ ,

'Item Data' = (All of the specified ~~trend~~ records in the order of increasing sequence number. The items specified are all items with the sequence number in the range of x through (x+y-1) in that order).

'First Sequence Number' = x

Test Example (using sample buffer at beginning of section):

1. TRANSMIT ReadRange-Request,
  - 'Object Identifier' = 20:1,
  - 'Property Identifier' = Log\_Buffer,
  - 'Reference Sequence Number' = 16,
  - 'Count' = 11
2. RECEIVE ReadRange-ACK,
  - 'Object Identifier' = 20:1,
  - 'Property Identifier' = Log\_Buffer,
  - 'Result Flags' = {TRUE, TRUE, FALSE},
  - 'Item Count' = 11,
  - 'Item Data' = Records < a, b, c, d, e, f, g, h, i, j, k > in that order.
  - 'First Sequence Number' = 16

[Change and renumber **Clause 9.21.1.10**, p. 408]

[Reviewer Note: Usage of Record\_Count was incorrect and changed to Total\_Record\_Count.]

#### **9.21.1.10.21.1.9 Reading Items by Sequence with Negative Count**

Purpose: To verify that the IUT correctly responds to a ReadRange service request to return items specified by indicating a sequence number and the number of items after that sequence to return.

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 the 'By Sequence' option and a negative value for 'Count'. The 'Reference Sequence Number' and 'Count' are selected so that the results can be conveyed in a single acknowledgment.

Test Steps:

1. TRANSMIT ReadRange-Request,
  - 'Object Identifier' = (the log object configured for this test),
  - 'Property Identifier' = Log\_Buffer,
  - 'Reference Sequence Number' = (any value x:  $(Total\_Record\_Count - Record\_Count + 2) < x \leq Total\_Record\_Count$ ),
  - 'Count' = (any value y:  $0 < |y| < (Record\_Count - (Total\_Record\_Count - x) + 1)$ )
2. RECEIVE ReadRange-ACK,
  - 'Object Identifier' = (the log object configured for this test),
  - 'Property Identifier' = Log\_Buffer,
  - 'Result Flags' = {?, ?, FALSE},
  - 'Item Count' = y,
  - 'Item Data' = (All of the specified records in order of increasing sequence number. The items specified are all items in the range of (x-|y|+1) through x in that order.)
  - 'First Sequence Number' = (x - |y| + 1)

Test Example (using sample buffer at beginning of section):

1. TRANSMIT ReadRange-Request,
  - 'Object Identifier' = (Trend Log, 1),
  - 'Property Identifier' = Log\_Buffer,
  - 'Reference Sequence Number' = 24,
  - 'Count' = -6
2. RECEIVE ReadRange-ACK,

'Object Identifier' = (Trend Log, 1),  
 'Property Identifier' = Log\_Buffer,  
 'Result Flags' = {FALSE, FALSE, FALSE},  
 'Item Count' = 6,  
 'Item Data' = Records < d, e, f, g, h, i > in that order.  
 'First Sequence Number' = 19

[Remove **Clause 9.21.1.11**, p. 409]

#### **9.21.1.11 Data Type Verification Test**

[Renumber **Clause 9.21.1.13**, p. 409]

[Reviewer Note: Clause 9.21.1.12 was moved to Clause 7 and renumbered as 7.3.2.24.X1]

#### **9.21.1.13 9.21.1.10 Reading Items with Negative Count and MOREITEMS**

[Add new **Clause 9.21.1.X1**, p. 410]

[Reviewer Note: Added a ReadRange test for non-Log\_Buffer list properties.]

#### **9.21.1.X1 ReadRange Support for All List Properties**

Purpose: To verify that all list properties of all objects can be read using the 3 by position forms of the ReadRange service.

Configuration Requirements: The IUT must be configured with at least one non-empty list property.

Test Steps:

1. REPEAT X = (all objects in the IUT's database) DO {
 REPEAT Y = (all list properties in object X) DO {
 TRANSMIT ReadRange-Request  
     'Object Identifier' = X,  
     'Property Identifier' = Y,  
 RECEIVE (ReadRange-ACK  
     'Object Identifier' = X,  
     'Property Identifier' = Y,  
     'Result Flags' = (?, ?, ?),  
     'Item Count' = (C: up to number of items in Y)  
     'Item Data' = (the first C elements of Y) )|  
 (ReadRange-ACK  
     'Object Identifier' = X,  
     'Property Identifier' = Y,  
     'Result Flags' = (FALSE, FALSE, FALSE),  
     'Item Count' = (C = 0)  
     'Item Data' = ())  
 IF (C <> 0) THEN  
     TRANSMIT ReadRange-Request  
         'Object Identifier' = X,  
         'Property Identifier' = Y,  
         'Reference Index' = 1,  
         'Count' = (C: any valid positive value)  
     RECEIVE ReadRange-ACK  
         'Object Identifier' = X,  
         'Property Identifier' = Y,  
         'Result Flags' = (TRUE, ?, ?),  
         'Item Count' = (C2: up to C)



```

        'Item Data' = (the first C2 elements of Y)
    TRANSMIT ReadRange-Request
        'Object Identifier' = X,
        'Property Identifier' = Y,
        'Reference Index' = (the number of elements in Y),
        'Count' = (C: any valid negative value)
    RECEIVE ReadRange-ACK
        'Object Identifier' = X,
        'Property Identifier' = Y,
        'Result Flags' = (?, TRUE, ?),
        'Item Count' = (C2: up to abs(C))
        'Item Data' = (the last C2 elements of Y)
    }
}

```

[Add new **Clause 9.21.1.X10**, p. 410]

### 9.21.1.X10 ReadRange Service when Non-BACnet Device Offline

Purpose: To verify that the ReadRange Service executes successfully when a non-BACnet device is offline.

Test Concept: Object1 is an object which contains information from a non-BACnet device. The non-BACnet device is verified to be online and recognized by the IUT. It is then made to go offline, and the IUT is made to recognize that the device is offline. A property, P1, from Object1 which contains a dynamic value derived from the data in the non-BACnet device is read from the IUT.

Test Steps:

1. CHECK (any vendor-specified indication, that the non-BACnet device is online)
2. MAKE (the non-BACnet device go offline)
3. MAKE (the IUT notice that the non-BACnet device is offline)
4. TRANSMIT ReadRange-Request,
  - 'Object Identifier' = Object1,
  - 'Property Identifier' = P1
5. RECEIVE ReadRange-ACK,
  - 'Object Identifier' = Object1,
  - 'Property Identifier' = P1,
  - 'Property Value' = (any valid value)

[Add new **Clause 9.21.2**, p. 410]

## 9.21.2 Negative ReadRange Service Execution Tests

### 9.21.2.1 Attempting to Read a Property That Does not Exist

Purpose: To verify the correct execution of the ReadRange service request when the requested property does not exist. This test is only applied to devices with a Protocol\_Revision of 10 or higher.

Configuration Requirements: If all the list properties applicable for the object under testing are supported, then this test shall be skipped.

Test Steps:

1. TRANSMIT ReadRange-Request,
  - 'Object Identifier' = (any object that exists in the IUT),
  - 'Property Identifier' = (any list property applicable for that object but not supported by the IUT),
2. RECEIVE BACnet-Error-PDU,

'Error Class' = PROPERTY,  
'Error Code' = UNKNOWN\_PROPERTY

### 9.21.2.2 Attempting to Read a Property That is not a List

Purpose: To verify the correct execution of the ReadRange service request when the requested property is not a list. This test is only applied to devices with a Protocol\_Revision of 10 or higher.

Test Steps:

1. TRANSMIT ReadRange-Request,  
    'Object Identifier' = (any object that exists in the IUT),  
    'Property Identifier' = (any non-list property supported by and present in the IUT),
2. RECEIVE BACnet-Error-PDU,  
    'Error Class' = SERVICES,  
    'Error Code' = PROPERTY\_IS\_NOT\_A\_LIST

### 9.21.2.3 Attempting to Read a non-Array Property with an Array Index

Purpose: To verify the correct execution of the ReadRange service request when the requested property is not an array of lists. This test is only applied to devices with a Protocol\_Revision of 10 or higher.

Test Steps:

1. TRANSMIT ReadRange-Request,  
    'Object Identifier' = (any object that exists in the IUT),  
    'Property Identifier' = (any non-array list property supported by and present in the IUT),  
    'Property Array Index' = (any valid value)
2. RECEIVE BACnet-Error-PDU,  
    'Error Class' = PROPERTY,  
    'Error Code' = PROPERTY\_IS\_NOT\_AN\_ARRAY

### 9.21.2.4 Reading a Range of Items that do not Exist (Using by Position)

Purpose: To verify that the IUT correctly responds to a ReadRange service request when there are no items within the specified criteria.

Test Concept: A ReadRange request is transmitted by the TD requesting a range of items known not to be in the Log\_Buffer. The IUT shall respond by returning an empty list.

Test Steps:

1. TRANSMIT ReadRange-Request,  
    'Object Identifier' = (the log object configured for this test),  
    'Property Identifier' = Log\_Buffer,  
    'Reference Index' = (any value that will result in no items being present)  
    'Count' = (any non-zero number)
2. RECEIVE ReadRange-ACK,  
    'Object Identifier' = (the log object configured for this test),  
    'Property Identifier' = Log\_Buffer,  
    'Result flags' = {FALSE, FALSE, FALSE},  
    'Item Count' = 0,  
    'Item Data' = (an empty list)  
    'First Sequence Number' = (should be absent)

Test Example 1 (using index that does not exist):

1. TRANSMIT ReadRange-Request,
  - 'Object Identifier' = (the log object configured for this test),
  - 'Property Identifier' = Log\_Buffer,
  - 'Reference Index' = 0
  - 'Count' = 5
2. RECEIVE ReadRange-ACK,
  - 'Object Identifier' = (the log object configured for this test),
  - 'Property Identifier' = Log\_Buffer,
  - 'Result flags' = {FALSE, FALSE, FALSE},
  - 'Item Count' = 0,
  - 'Item Data' = (an empty list)

Test Example 2 (using index that does not exist):

1. TRANSMIT ReadRange-Request,
  - 'Object Identifier' = (the log object configured for this test),
  - 'Property Identifier' = Log\_Buffer,
  - 'Reference Index' = Index= Buffer\_Size + 1
  - 'Count' = -20
2. RECEIVE ReadRange-ACK,
  - 'Object Identifier' = (the log object configured for this test),
  - 'Property Identifier' = Log\_Buffer,
  - 'Result flags' = {FALSE, FALSE, FALSE},
  - 'Item Count' = 0,
  - 'Item Data' = (an empty list)

## 9.22 WriteProperty Service Execution Tests

### 9.22.1 Positive WriteProperty Service Execution Tests

[Change **Clause 9.22.1.1**, p. 410]

[Reviewer Note: Modified test to remove dependency on EPICS values.]

#### 9.22.1.1 Writing a Single Element of an Array

Purpose: To verify that the IUT can execute WriteProperty service requests when the property is an array, and a single array element is written.

Test Concept: The TD shall select an object in the IUT that contains a writable array property. This property is designated P1. If no suitable object can be found, then this test shall be omitted.

Configuration Requirements: If the IUT supports writing array values it shall be configured with at least one writable property that can be used for this test.

Test Steps:

1. *READ*  $X = (\text{Object1})$ ,  $P1 \text{ ARRAY INDEX} = (\text{any value } N: 1 \leq N \leq \text{the size of the array})$
1. ~~VERIFY (Object1), P1 = (the value defined for this property in the EPICS)~~
2. TRANSMIT WriteProperty-Request,
  - 'Object Identifier' = Object1,
  - 'Property Identifier' = P1,
  - 'Property Array Index' = ~~N~~ *(any value N: 1 ≤ N ≤ the size of the array)*
  - 'Property Value' = (any valid value of the correct datatype *subject to the restrictions specified in the EPICS as defined in 4.4.2* for this array, except the value  $X$  read

for this element in step 1)

3. RECEIVE ~~Simple-ACK-PDU~~BACnet-SimpleACK-PDU
4. VERIFY (Object1), P1 = (the value used in step 2), ARRAY INDEX = N

[Change **Clause 9.22.1.2**, p. 411]

[Reviewer Note: Modified test to remove dependency on EPICS values.]

### 9.22.1.2 Writing a Commandable Property Without a Priority

Purpose: To verify that the IUT can execute WriteProperty service requests when the property is commandable, but a priority is not specified.

Test Concept: The TD shall select an object in the IUT that contains a writable property that is commandable and has no internal algorithm writing to it at priority 16. If no suitable object can be found, then this test shall be omitted.

Configuration Requirements: If the IUT supports commandable properties that have no internal algorithm writing at priority 16, it shall be configured with at least one such property that can be used for this test.

Test Steps:

1. READ  $X = (Object1), Priority\_Array, ARRAY\ INDEX = 16$
- ~~1. VERIFY (Object1), Priority\_Array = (the value defined for this property in the EPICS), ARRAY INDEX = 16~~
2. TRANSMIT WriteProperty-Request,  
     'Object Identifier' = Object1,  
     'Property Identifier' = Present\_Value,  
     'Property Value' = (any *valid* value of the correct datatype for this property subject to the restrictions specified in the EPICS as defined in 4.4.2, except the value  $X$  read in step 1)
3. RECEIVE ~~Simple-ACK-PDU~~BACnet-SimpleACK-PDU
4. VERIFY (Object1), Priority\_Array = (the value used in step 2), ARRAY INDEX = 16

[Change **Clause 9.22.1.3**, p. 411]

[Reviewer Note: Modified test to remove dependency on EPICS values.]

### 9.22.1.3 Writing a Non-Commandable Property with a Priority

Purpose: To verify that the IUT can execute WriteProperty service requests when the property is not commandable but a priority is specified.

Test Concept: The TD shall select an object in the IUT that contains a writable property that is not commandable and has no internal algorithm writing to it. If no suitable property can be found, then this test shall be omitted.

Configuration Requirements: If the IUT supports non-commandable properties that have no internal algorithm writing to them, it shall be configured with at least one such property that can be used for this test.

Test Steps:

1. READ  $X = (Object1), P1$
- ~~1. VERIFY (Object1), P1 = (the value defined for this property in the EPICS)~~
2. TRANSMIT WriteProperty-Request,  
     'Object Identifier' = Object1,  
     'Property Identifier' = P1,  
     'Priority' = (any valid priority)  
     'Property Value' = (any valid value defined for this property subject to the restrictions specified in the EPICS as defined in 4.4.2, except the value  $X$  read in step 1)
3. RECEIVE BACnet-BACnet-SimpleACK-PDU
4. VERIFY (Object1), P1 = (the value used in step 2)

[Add new **Clause 9.22.1.X1**, p. 411]

### **9.22.1.X1 Writing an Array Size**

Purpose: This test case verifies that the IUT can execute WriteProperty service requests to the array size of a writable, non-fixed size array property.

Test Concept: The TD shall select an object in the IUT that contains a writable array property of a non-fixed size. This property is designated P1. If no suitable object can be found, then this test shall be omitted.

Configuration Requirements: If the IUT supports writable non-fixed size array properties, it shall be configured with at least one writable non-fixed size array property that can be used for this test.

Test Steps:

1. READ X = (Object1), P1 ARRAY INDEX = 0
2. TRANSMIT WriteProperty-Request,  
    'Object Identifier' = Object1,  
    'Property Identifier' = P1,  
    'Property Array Index' = 0  
    'Property Value' = (any valid array size defined for this property subject to the  
                          restrictions specified in the EPICS as defined in 4.4.2,  
                          except the value verified in step 1)
3. RECEIVE BACnet-SimpleACK-PDU
4. VERIFY (Object1), P1[0] = (the value used in step 2)

[Add new **Clause 9.22.1.X2**, p. 411]

[Reviewer Note: Added a general WriteProperty test that can be used in a variety of situations.]

### **9.22.1.X2 Writing to Properties Based on Data Type**

Purpose: This test case verifies that the IUT can execute WriteProperty service requests to specific data types supported by the IUT.

Test Concept: For the specified base data type, the TD shall select an object in the IUT that contains a writable property of that data type. This property is designated P1.

Configuration Requirements: The IUT shall be configured with at least one writable property of the specified data type to be used for this test.

Test Steps:

1. X = READ (Object1), P1
2. TRANSMIT WriteProperty-Request,  
    'Object Identifier' = Object1,  
    'Property Identifier' = P1,  
    'Property Value' = (any valid value defined for this property subject to the  
                          restrictions specified in the EPICS as defined in 4.4.2,  
                          except the value X determined in step 1)
3. RECEIVE BACnet-SimpleACK-PDU
4. VERIFY (Object1), P1 = (the value used in step 2)

### 9.22.2 Negative WriteProperty Service Execution Tests

[Change **Clause 9.22.2.1**, p. 411]

[Reviewer Note: Modified test to remove dependency on EPICS values.]

#### 9.22.2.1 Writing Non-Array Properties with an Array Index

Purpose: To verify that the IUT can execute WriteProperty service requests when the property value is not an array but an ~~ARRAY INDEX~~ *array index* is included in the service request.

Test Concept: The TD shall select an object in the IUT that contains a writable scalar property designated P1. An attempt will be made to write to this property using an array index. If no suitable object can be found, then this test shall be omitted.

Configuration Requirements: If the IUT supports any writable properties that are scalars, it shall be configured with at least one such property that can be used for this test.

Test Steps:

1. *READ X = (Object1), P1*
- ~~1. VERIFY (Object1), P1 = (the value defined for this property in the EPICS)~~
2. TRANSMIT WriteProperty-Request,  
    'Object Identifier' = Object1,  
    'Property Identifier' = P1,  
    'Property Value' = (any *valid* value of the correct datatype for this property subject to the restrictions specified in the EPICS as defined in 4.4.2, except the value *X* read in step 1),  
    'Property Array Index' = (any positive integer)
3. IF (Protocol\_Revision is present and Protocol\_Revision >= 4) THEN  
    RECEIVE BACnet-Error PDU,  
    'Error Class' = PROPERTY,  
    'Error Code' = PROPERTY\_IS\_NOT\_AN\_ARRAY  
ELSE  
    RECEIVE BACnet-Error PDU,  
    'Error Class' = SERVICES,  
    'Error Code' = INCONSISTENT\_PARAMETERS
4. ~~VERIFY (Object1), P1 = X(the value defined for this property in the EPICS)~~

[Change **Clause 9.22.2.2**, p. 412]

[Reviewer Note: Modified test to remove dependency on EPICS values. Fixed array index description per BTL-CR-0373.]

#### 9.22.2.2 Writing Array Properties with an Array Index that is Out of Range

Purpose: To verify that the IUT can execute WriteProperty service requests when the requested property value is an array but the ~~ARRAY INDEX~~ *array index* is out of range.

Test Concept: The TD shall select an object in the IUT that contains a writable array property designated P1. An attempt will be made to write to this property using an array index that is out of range. If no suitable object can be found, then this test shall be omitted.

Configuration Requirements: If the IUT supports any writable properties that are arrays, it shall be configured with at least one such property that can be used for this test.

Test Steps:

1. *READ X = (Object1), P1*
- ~~1. VERIFY (Object1), P1 = (the value defined for this property in the EPICS)~~

2. TRANSMIT WriteProperty-Request,  
     'Object Identifier' = Object1,  
     'Property Identifier' = P1,  
     'Property Value' = (any value of the correct datatype for this property subject to the restrictions specified in the EPICS as defined in 4.4.2, except the value *X* read in step 1),  
     'Property Array Index' = (any ~~value positive integer~~ that is larger than ~~that the current supported size of~~ the array)
3. RECEIVE BACnet-Error PDU,  
     'Error Class' = PROPERTY,  
     'Error Code' = INVALID\_ARRAY\_INDEX
4. VERIFY (Object1), P1 = ~~*X* (the value defined for this property in the EPICS)~~

[Change **Clause 9.22.2.3**, p. 412]

[Reviewer Note: Updated Test Concept and Added Configuration Requirements.]

### 9.22.2.3 Writing with a Property Value Having the Wrong Datatype

Purpose: To verify that the IUT correctly responds to an attempt to write a property value that has an invalid datatype.

Test Concept: The TD shall select an object in the IUT that contains a writable property designated P1. An attempt will be made to write to this property using a datatype that the IUT supports but ~~which is invalid for the property~~ which is not compliant with the property definition given by the BACnet standard.

Configuration Requirements: The value to be written shall not be of a datatype which is compliant with the property definition. If no object supports writable properties, then this test shall be omitted.

Test Steps:

1. READ *X* = (Object1), P1
2. TRANSMIT WriteProperty-Request,  
     'Object Identifier' = Object1,  
     'Property Identifier' = P1,  
     'Property Value' = (any value with an invalid datatype)
3. RECEIVE  
     (BACnet-Error PDU,  
         'Error Class' = PROPERTY,  
         'Error Code' = INVALID\_DATATYPE) |  
     (BACnet-Reject-PDU  
         Reject Reason = INVALID\_PARAMETER\_DATATYPE) |  
     (BACnet-Reject-PDU  
         Reject Reason = INVALID\_TAG)
4. VERIFY (Object1), P1 = ~~*X*~~

[Change **Clause 9.22.2.4**, p. 413]

[Reviewer Note: Modified test to remove dependency on EPICS values. Modified to allow OPTIONAL\_FUNCTIONALITY\_NOT\_SUPPORTED in some specific cases.]

### 9.22.2.4 Writing with a Property Value that is Out of Range

Purpose: To verify that the IUT can execute WriteProperty service requests when an attempt is made to write a value that is outside of the supported range.

Test Concept: The TD attempts to write to a property using a value that is outside of the supported range. If the IUT does not contain any writable properties that have restricted ranges, then this test shall be skipped.

Notes to Tester: The value used in step 2 shall be of the correct datatype. For bit string types, the bit count shall be correct, for Date and Time values, the value shall be within the range defined by the standard for the datatype, for constructed values, the

*constructed value shall match the structure defined by the ASN.1, and all field values shall be within the ranges defined by the standard for those field values.*

*In this test, the error code OPTIONAL\_FUNCTIONALITY\_NOT\_SUPPORTED is permitted only if the value written would require the device to exhibit non-supported optional functionality.*

Test Steps:

1. READ X = (Object1), P1  
~~1. VERIFY (Object1), P1 = (the value defined for this property in the EPICS),~~
2. TRANSMIT WriteProperty-Request,  
    'Object Identifier' = (Object1, any object with writable properties),  
    'Property Identifier' = (P1, any writable property with a restricted range of values),  
    'Property Value' = (any value, of the correct datatype, that is outside the supported range)
3. IF (Protocol\_Revision is present and Protocol\_Revision >= 4) THEN  
    RECEIVE  
    (BACnet-Error PDU,  
    'Error Class' = PROPERTY,  
    'Error Code' = VALUE\_OUT\_OF\_RANGE) |  
    (BACnet-Error-PDU,  
    'Error Class' = PROPERTY,  
    'Error Code' = OPTIONAL\_FUNCTIONALITY\_NOT\_SUPPORTED)  
ELSE  
    RECEIVE  
    (BACnet-Error-PDU,  
    'Error Class' = PROPERTY,  
    'Error Code' = VALUE\_OUT\_OF\_RANGE) |  
    (BACnet-Error-PDU,  
    'Error Class' = PROPERTY,  
    'Error Code' = OPTIONAL\_FUNCTIONALITY\_NOT\_SUPPORTED) |  
    (BACnet-Reject-PDU,  
    Reject Reason = PARAMETER\_OUT\_OF\_RANGE)
4. VERIFY (Object1), P1 = X~~(the value defined for this property in the EPICS)~~

~~Notes to Tester: The value used in step 2 shall be of the correct datatype. For bit string types, the bit count shall be correct, for Date and Time values, the value shall be within the range defined by the standard for the datatype, for constructed values, the constructed value shall match the structure defined by the ASN.1 and all field values shall be within the ranges defined by the standard for those field values.~~

[Add new **Clause 9.22.2.X1**, p. 415]

[Reviewer Note: Existing test 9.22.2.1 forbids the testing of a read-only property, to observe the response when an array index is included in the service request.]

### 9.22.2.X1 Writing Non-Array Read-only Property with an Array Index

Purpose: This test case verifies that the IUT correctly responds to an attempt to write a property value when the property value is not an array, but an array index is included in the service request, and the property specified in the service request is not writable.

Test Concept: Select an object, designated Object1, in the IUT that contains a non-writable scalar property designated P1. An attempt will be made to write to this property with an array index included. If no object supports non-writable scalar properties, then this test shall be omitted.

Test Steps:

1. TRANSMIT WriteProperty-Request,  
    'Object Identifier' = Object1,  
    'Property Identifier' = P1,  
    'Property Value' = (any value of the correct datatype for this property)



'Property Array Index' = (any positive integer)

2. IF (Protocol\_Revision is present and Protocol\_Revision  $\geq$  4) THEN  
RECEIVE BACnet-Error PDU,  
'Error Class' = PROPERTY,  
'Error Code' = WRITE\_ACCESS\_DENIED | PROPERTY\_IS\_NOT\_AN\_ARRAY  
ELSE  
RECEIVE (BACnet-Error PDU,  
'Error Class' = SERVICES,  
'Error Code' = INCONSISTENT\_PARAMETERS) |  
(BACnet-Error PDU,  
'Error Class' = PROPERTY,  
'Error Code' = WRITE\_ACCESS\_DENIED | PROPERTY\_IS\_NOT\_AN\_ARRAY)

[Add new **Clause 9.22.2.X2**, p. 415]

### 9.22.2.X2 Resizing a writable fixed size array property

Purpose: This test case verifies that the IUT correctly responds to an attempt to resize a writable fixed size array property using WriteProperty service.

Test Concept: Select an object (O1) in the IUT that contains a writable array property of a fixed size. This property is designated P1. If no suitable object can be found, then this test shall be omitted.

Test Steps:

1. READ X = (O1), P1 ARRAY INDEX = 0
2. WRITE P1= (Entire Array with any valid value greater than Array Size X)
3. RECEIVE BACnet-Error-PDU,  
'Error Class' = PROPERTY,  
'Error Code' = INVALID\_ARRAY\_INDEX | VALUE\_OUT\_OF\_RANGE
4. VERIFY (O1), P1= X, ARRAY INDEX = 0
5. WRITE P1= (Entire Array with any valid value less than Array Size X)
6. RECEIVE BACnet-Error PDU,  
'Error Class' = PROPERTY,  
'Error Code' = INVALID\_ARRAY\_INDEX | VALUE\_OUT\_OF\_RANGE
7. VERIFY (O1), P1= X, ARRAY INDEX = 0
8. WRITE P1 = (any valid value greater than Array Size X), ARRAY INDEX=0
9. RECEIVE BACnet-Error PDU,  
'Error Class' = PROPERTY,  
'Error Code' = INVALID\_ARRAY\_INDEX | VALUE\_OUT\_OF\_RANGE | WRITE\_ACCESS\_DENIED
10. VERIFY (O1), P1= X, ARRAY INDEX = 0,
11. WRITE P1 = (any valid value less than Array Size X), ARRAY INDEX=0
12. RECEIVE BACnet-Error PDU,  
'Error Class' = PROPERTY,  
'Error Code' = INVALID\_ARRAY\_INDEX | VALUE\_OUT\_OF\_RANGE | WRITE\_ACCESS\_DENIED
13. VERIFY (O1), P1= X, ARRAY INDEX = 0

[Add new **Clause 9.22.2.X4**, p. 415]

[Reviewer Note: No test exists in the actual test plan when a strong indication by the explicit mention of PROPERTY OPTIONAL\_FUNCTIONALITY\_NOT\_SUPPORTED is mentioned in Clause 12.]

### 9.22.2.X4 Writing a Property Value Related to Non-supported Optional Functionality

Purpose: To verify that the IUT responds correctly to WriteProperty service requests when the value provided would require the device to exhibit non-supported optional functionality.

Test Concept: A writable property, P1, is selected for which the standard defines optional behavior which the IUT does not support, and the writing of specific values to the property would require the device to exhibit the non-supported behavior. The TD attempts to write to the property using a selected value, V1, which would require the IUT to exhibit non-supported optional functionality.

Test Steps:

1. READ X = (Object1), P1
2. TRANSMIT WriteProperty-Request,  
     'Object Identifier' = (Object1, the object containing a writable P1),  
     'Property Identifier' = (P1, any writable property with a restricted range of values),  
     'Property Value' = (V1, a value that would require a non-supported optional functionality)
3. RECEIVE BACnet-Error-PDU  
     'Error Class' = PROPERTY,  
     'Error Code' = OPTIONAL\_FUNCTIONALITY\_NOT\_SUPPORTED
4. VERIFY (Object1), P1 = X

## 9.23 WritePropertyMultiple Service Execution Tests

### 9.23.1 Positive WritePropertyMultiple Service Execution Tests

[Change **Clause 9.23.1.1**, p. 415]

[Reviewer Note: Modified test to remove dependency on EPICS values.]

#### 9.23.1.1 Writing a Single Property to a Single Object

Purpose: To verify the ability to write a single property to a single object.

Test Concept: This test case attempts to write to a single scalar property, P1, that is not commandable. If no such writable property exists, the test can be modified to write to an array property or to a commandable property with a write priority high enough to ensure that the commandable property's value will change.

Configuration Requirements: If the IUT supports any writable scalar properties that are not commandable it shall be configured with one for use in this test. If no such properties are supported the IUT shall be configured with a writable array or commandable property and the test steps modified to account for this variation.

Test Steps:

1. READ X = (Object1), P1
- ~~1. VERIFY (Object1), P1 = (the value specified for this property in the EPICS)~~
2. TRANSMIT WritePropertyMultiple-Request,  
     'Object Identifier' = Object1,  
     'Property Identifier' = P1,  
     'Property Value' = (any valid value of the appropriate datatype for this property subject to the restrictions specified in the EPICS as defined in 4.4.2, except the value X ~~except for the one~~ read in step 1)
3. RECEIVE BACnet-Simple-ACK-PDU
4. VERIFY (Object1), P1 = (the value specified in step 2)

[Change **Clause 9.23.1.2**, p. 416]

[Reviewer Note: Modified test to remove dependency on EPICS values.]

### 9.23.1.2 Writing Multiple properties to a Single Object

Purpose: To verify the ability to write multiple properties to a single object.

Test Concept: This test case attempts to write to multiple scalar properties, P1 and P2, that are not commandable. If two such writable properties don't exist, the test can be modified to write to an array property or to a commandable property with a write priority high enough to ensure that the commandable property's value will change.

Configuration Requirements: If the IUT supports any object that has two writable scalar properties that are not commandable it shall be configured with one for use in this test. If no such properties are supported the IUT shall be configured, if possible, with writable array or commandable properties and the test steps modified to account for this variation. If no object type is supported that has two or more writable properties this test may be omitted. The IUT must support either the configuration required for this test or a configuration required for test 9.23.1.3

Test Steps:

1. READ  $X = (Object1), P1$
2. READ  $Y = (Object1), P2$
1. VERIFY (Object1), P1 = ~~(the value specified for this property in the EPICS)~~
2. VERIFY (Object1), P2 = ~~(the value specified for this property in the EPICS)~~
3. TRANSMIT WritePropertyMultiple-Request,  
     'Object Identifier' = Object1,  
     'Property Identifier' = P1,  
     'Property Value' = (any valid value of the appropriate datatype for this property subject to the restrictions specified in the EPICS as defined in 4.4.2, except the value  $X$  ~~except for the one~~ read in step 1),  
     'Property Identifier' = P2,  
     'Property Value' = (any valid value of the appropriate datatype for this property subject to the restrictions specified in the EPICS as defined in 4.4.2, except the value  $Y$  ~~except for the one~~ read in step 2)
4. RECEIVE ~~BACnet-Simple-ACK-PDU~~ BACnet-SimpleACK-PDU
5. VERIFY (Object1), P1 = (the value specified for P1 in step 23)
6. VERIFY (Object1), P2 = (the value specified for P2 in step 23)

[Change **Clause 9.23.1.3**, p. 416]

[Reviewer Note: Modified test to remove dependency on EPICS values.]

### 9.23.1.3 Writing a Single Property to Multiple Objects

Purpose: To verify the ability to write a single property from multiple objects.

Test Concept: This test case attempts to write to single scalar properties, P1 and P2, that reside in different objects but are not commandable. If two such writable properties don't exist the test can be modified to write to an array property or to a commandable property with a write priority high enough to ensure that the commandable property's value will change.

Test Steps:

1. READ  $X = (Object1), P1$
2. READ  $Y = (Object2), P2$
1. VERIFY (Object1), P1 = ~~(the value specified for this property in the EPICS)~~
2. VERIFY (Object2), P2 = ~~(the value specified for this property in the EPICS)~~
3. TRANSMIT WritePropertyMultiple-Request,  
     'Object Identifier' = Object1,  
     'Property Identifier' = P1,  
     'Property Value' = (any valid value of the appropriate datatype for this property subject to the restrictions specified in the EPICS as defined in 4.4.2, except the value  $X$  ~~except for the one~~ read in step 1),  
     'Object Identifier' = Object2,  
     'Property Identifier' = P2,  
     'Property Value' = (any valid value of the appropriate datatype for this property subject to the restrictions specified in the EPICS as defined in 4.4.2, except the value  $Y$  ~~except for the one~~ read in step 2)
4. RECEIVE ~~BACnet-Simple-ACK-PDU~~ BACnet-SimpleACK-PDU

5. VERIFY (Object1), P1 = (the value specified for P1 in step 3)
6. VERIFY (Object2), P2 = (the value specified for P2 in step 3)

[Change **Clause 9.23.1.4**, p. 416]

[Reviewer Note: Modified test to remove dependency on EPICS values.]

#### 9.23.1.4 Writing Multiple Properties to Multiple Objects

Purpose: To verify the ability to write multiple properties to multiple objects.

Test Concept: This test case attempts to write properties, P1 and P2, that reside in Object1, and properties P3 and P4 that reside in Object2. P1, P2, P3 and P4 are not commandable properties. If four such writable properties do not exist, the test can be modified to write to an array property or to a commandable property with a write priority high enough to ensure that the commandable property's value will change.

Test Steps:

1. READ X = (Object1), P1
2. READ Y = (Object1), P2
3. READ Z = (Object2), P3
4. READ A = (Object2), P4
1. ~~VERIFY (Object1), P1 = (the value specified for this property in the EPICS)~~
2. ~~VERIFY (Object1), P2 = (the value specified for this property in the EPICS)~~
3. ~~VERIFY (Object2), P3 = (the value specified for this property in the EPICS)~~
4. ~~VERIFY (Object2), P4 = (the value specified for this property in the EPICS)~~
5. TRANSMIT WritePropertyMultiple-Request,  
     'Object Identifier' = Object1,  
     'Property Identifier' = P1,  
     'Property Value' = (any valid value of the appropriate datatype for this property subject to the restrictions specified in the EPICS as defined in 4.4.2, except the value X ~~except for the one~~ read in step 1),  
     'Property Identifier' = P2,  
     'Property Value' = (any valid value of the appropriate datatype for this property subject to the restrictions specified in the EPICS as defined in 4.4.2, except the value Y ~~except for the one~~ read in step 2),  
     'Object Identifier' = Object2,  
     'Property Value' = (any valid value of the appropriate datatype for this property subject to the restrictions specified in the EPICS as defined in 4.4.2, except the value Z ~~except for the one~~ read in step 3),  
     'Object Identifier' = Object2,  
     'Property Identifier' = P4,  
     'Property Value' = (any valid value of the appropriate datatype for this property subject to the restrictions specified in the EPICS as defined in 4.4.2, except the value A ~~except for the one~~ read in step 4)
6. RECEIVE BACnet-BACnet-SimpleACK-PDU
7. VERIFY (Object1), P1 = (the value specified for P1 in step 5)
8. VERIFY (Object1), P2 = (the value specified for P2 in step 5)
9. VERIFY (Object2), P3 = (the value specified for P3 in step 5)
10. VERIFY (Object2), P4 = (the value specified for P4 in step 5)

[Change **Clause 9.23.1.7**, p. 417]

[Reviewer Note: the test should not mandate that the writable property be in the device object.]

#### 9.23.1.7 Writing Maximum Multiple Properties

Purpose: This test case verifies that IUT does not arbitrarily restrict the number of properties that can be written to it using a single WritePropertyMultiple request, *to object 01*.

Test Concept: A writable property is written to an object in the IUT as many times as can be conveyed in the largest request accepted by the IUT. The calculation of the maximum request size shall be based on the IUT's Max\_APDU\_Length\_Accepted and maximum segments per request.

The procedure to determine the number of values to use is:

MaxAPDU = IUT's Max\_APDU\_Length\_Accepted  
 MaxRxSegs = IUT's maximum segments accepted per request  
 MaxTxSegs = IUT's maximum segments generated per response

NonSegRqstHdrSize = size of (non-segmented BACnetConfirmed-RequestPDU header) = 4  
 SegRqstHdrSize = size of (segmented BACnetConfirmed-RequestPDU header) = 6  
 ObjIdSize = size of (an Object-Identifier) = 5  
 TagsSize = size of (an open and a close tag) = 2

PropIdSize = size of (chosen property Id) = depends on property ID and includes ARRAY INDEX size if required  
 ValueSize = size of (chosen property value) = depends on property and value chosen

If the IUT does not support receiving segmented requests:

NumPropertiesToWrite =  

$$\frac{(\text{MaxAPDU} - \text{NonSegRqstHdrSize} - \text{ObjIdSize} - \text{TagsSize})}{(\text{PropIdSize} + \text{TagsSize} + \text{ValueSize})} =$$

$$(\text{MaxAPDU} - 11) / (\text{PropIdSize} + 2 + \text{ValueSize})$$

If the IUT does support receiving segmented requests:

NumPropertiesToWrite =  

$$\frac{((\text{MaxAPDU} - \text{SegRqstHdrSize}) * \text{MaxRxSegs} - \text{ObjIdSize} - \text{TagsSize})}{\text{PropIdSize}} =$$

$$((\text{MaxAPDU} - 6) * \text{MaxRxSegs} - 7) / 2$$

Test Steps:

1. TRANSMIT WritePropertyMultiple-Request,  
     'Object Identifier' = ~~(Device, XO1)~~,  
     'Property Identifier' = P1,  
     'Priority Array Index' = A1, -- only if required  
     'Property Value' = V1,  
     ...  
     'Property Identifier' = P1,  
     'Priority Array Index' = A1, -- only if required  
     'Property Value' = V1
2. RECEIVE ~~Simple-ACK~~ BACnet-SimpleACK-PDU
3. VERIFY ~~(P1=V1)~~ OI, P1 = V1

[Add new **Clause 9.23.1.X4**, p. 418]

### 9.23.1.X4 Writing an Array Size

Purpose: This test case verifies that the IUT can execute WritePropertyMultiple service requests to the array size of a writable, non-fixed size array property.

Test Concept: Repeat test 9.22.1.X1 Writing an Array Size using WritePropertyMultiple instead of WriteProperty.

### 9.23.2 Negative WritePropertyMultiple Service Execution Tests

[Change **Clause 9.23.2.1**, p. 419]

[Reviewer Note: Modified test to remove dependency on EPICS values.]

### 9.23.2.1 Writing Multiple Properties with a Property Access Error

Purpose: To verify the ability to correctly execute a WritePropertyMultiple service request for which the 'List of Write Access Specifications' contains a specification for an unsupported property.

Test Concept: An attempt is made to write to two properties in a single object. The first property is supported and writable. The second property is not supported for this object. The objective is to verify that an appropriate error response is returned and that all writes up to the first failed write attempt take place.

Configuration Requirements: If the IUT supports any writable scalar properties that are not commandable it shall be configured with one for use in this test. If no such properties are supported the IUT shall be configured with a writable array or commandable property and the test steps modified to account for this variation. In the test description Object1 will be used to designate the object, P1 the writable property, and P2 the unsupported property used for this test.

Test Steps:

1. *READ X = (Object1), P1*
- ~~1. VERIFY (Object1), P1 = (the value specified for this property in the EPICS)~~
2. TRANSMIT WritePropertyMultiple-Request,  
'Object Identifier' = Object1,  
'Property Identifier' = P1,  
'Property Value' = (any *valid* value of the appropriate datatype for this property subject to the restrictions specified in the EPICS as defined in 4.4.2, except the value X ~~except for the one~~ read in step 1),  
'Property Identifier' = P2,  
'Property Value' = (any *valid* value of the appropriate datatype for this property subject to the restrictions specified in the EPICS as defined in 4.4.2)
3. RECEIVE WritePropertyMultiple-Error,  
'Error Class' = PROPERTY,  
'Error Code' = UNKNOWN\_PROPERTY,  
'Object Identifier' = Object1,  
'Property Identifier' = P2
4. VERIFY (Object1), P1 = (the value specified for P1 in step 2)

[Change **Clause 9.23.2.2**, p. 419]

[Reviewer Note: Modified test to remove dependency on EPICS values.]

### 9.23.2.2 Writing Multiple Properties with an Object Access Error

Purpose: To verify the ability to correctly execute a WritePropertyMultiple service request for which the 'List of Write Access Specifications' contains a specification for an unsupported object.

Test Concept: An attempt is made to write to a single property in two different objects. The first object is supported and the property is writable. The second object is not supported. The objective is to verify that an appropriate error response is returned and that all writes up to the first failed write attempt take place.

Configuration Requirements: If the IUT supports any writable scalar properties that are not commandable it shall be configured with one for use in this test. If no such properties are supported the IUT shall be configured with a writable array or commandable property and the test steps modified to account for this variation. In the test description Object1 and P1 will be used to designate the writable object and property used for this test. The designation BadObject will be used to indicate an object that is not supported.

Test Steps:

1. *READ X = (Object1), P1*
- ~~1. VERIFY (Object1), P1 = (the value specified for this property in the EPICS)~~

2. TRANSMIT WritePropertyMultiple-Request,  
     'Object Identifier' = Object1,  
     'Property Identifier' = P1,  
     'Property Value' = (any *valid* value of the appropriate datatype for this property subject to the restrictions specified in the EPICS as defined in 4.4.2, except the value ~~X except for the one~~ read in step 1),  
     'Object Identifier' = BadObject,  
     'Property Identifier' = P2,  
     'Property Value' = (any *valid* value of the appropriate datatype for this property subject to the restrictions specified in the EPICS as defined in 4.4.2)
3. RECEIVE WritePropertyMultiple-Error,  
     'Error Class' = OBJECT,  
     'Error Code' = UNKNOWN\_OBJECT,  
     'Object Identifier' = BadObject,  
     'Property Identifier' = P2
4. VERIFY (Object1), P1 = (the value specified for P1 in step 2)

[Change **Clause 9.23.2.3**, p. 420]

[Reviewer Note: Modified test to remove dependency on EPICS values.]

### 9.23.2.3 Writing Multiple Properties with a Write Access Error

Purpose: To verify the ability to correctly execute a WritePropertyMultiple service request for which the 'List of Write Access Specifications' contains a specification for a read only property.

Test Concept: An attempt is made to write to two properties in a single object. The first property is supported and writable. The second property is supported but read only. The objective is to verify that an appropriate error response is returned and that all writes up to the first failed write attempt take place.

Configuration Requirements: If the IUT supports any writable scalar properties that are not commandable it shall be configured with one for use in this test. If no such properties are supported the IUT shall be configured with a writable array or commandable property and the test steps modified to account for this variation. In the test description Object1 will be used to designate the object, P1 the writable property, and P2 the read only property used for this test.

Test Steps:

1. READ X = (Object1), P1
2. READ Y = (Object1), P2
- ~~1. VERIFY (Object1), P1 = (the value specified for this property in the EPICS)~~
- ~~2. VERIFY (Object1), P2 = (the value specified for this property in the EPICS)~~
3. TRANSMIT WritePropertyMultiple-Request,  
     'Object Identifier' = Object1,  
     'Property Identifier' = P1,  
     'Property Value' = (any *valid* value of the appropriate datatype for this property subject to the restrictions specified in the EPICS as defined in 4.4.2, except the value ~~X except for the one~~ read in step 1),  
     'Property Identifier' = P2,  
     'Property Value' = (any *valid* value of the appropriate datatype for this property subject to the restrictions specified in the EPICS as defined in 4.4.2, except the value ~~Y except for the one~~ read in step 1)
4. RECEIVE WritePropertyMultiple-Error,  
     'Error Class' = PROPERTY,  
     'Error Code' = WRITE\_ACCESS\_DENIED,  
     'Object Identifier' = Object1,  
     'Property Identifier' = P2
5. VERIFY (Object1), P1 = (the value specified for P1 in step 3)
6. VERIFY (Object1), P2 = Y~~(the value specified for this property in the EPICS)~~

[Change **Clause 9.23.2.4**, p. 420]

[Reviewer Note: Modified test to remove dependency on EPICS values. Added Property Array Index to error received.]

#### 9.23.2.4 Writing Non-Array Properties with an Array Index

**Purpose:** This test case verifies that the IUT can execute WritePropertyMultiple service requests when the 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 Concept:** The TD shall select an object, designated Object1, in the IUT that contains a writable scalar property designated P1 *having a value X*. An attempt will be made to write to this property using an array index Y. If no suitable object can be found, then this test shall be omitted.

**Configuration Requirements:** If the IUT supports any writable properties that are scalars, it shall be configured with at least one such property that can be used for this test.

**Test Steps:**

1. *READ X = (Object1), P1*
1. ~~VERIFY (Object1), P1 = (the value defined for this property in the EPICS)~~
2. TRANSMIT WritePropertyMultiple-Request,
  - 'Object Identifier' = Object1,
  - 'Property Identifier' = P1,
  - 'Property Value' = (any *valid* value of the correct datatype for this property subject to the restrictions specified in the EPICS as defined in 4.4.2, except the value X read in step 1),
  - 'Property Array Index' = (Y, any positive integer)
3. RECEIVE WritePropertyMultiple-Error,
  - 'Error Class' = PROPERTY,
  - 'Error Code' = PROPERTY\_IS\_NOT\_AN\_ARRAY,
  - 'Object Identifier' = Object1,
  - 'Property Identifier' = P1,
  - 'Property Array Index' = Y
4. ~~VERIFY (Object1), P1 = X (the value defined for this property in the EPICS)~~

[Change **Clause 9.23.2.5**, p. 421]

[Reviewer Note: Modified test to remove dependency on EPICS values. Fixed array index description per BTL-CR-0373.]

#### 9.23.2.5 Writing Array Properties with an Array Index that is Out of Range

**Purpose:** This test case verifies that the IUT can execute WritePropertyMultiple service requests when the requested property value is an array, but the array index is out of range. This test shall only be performed if Protocol\_Revision is present and has a value greater than or equal to 4.

**Test Concept:** The TD shall select an object, designated Object1, in the IUT that contains a writable array property designated P1 *having value X*. An attempt will be made to write to this property using an array index, Y, that is out of range. If no suitable object can be found, then this test shall be omitted.

**Configuration Requirements:** If the IUT supports any writable properties that are arrays, it shall be configured with at least one such property that can be used for this test.

**Test Steps:**

1. *READ X = (Object1), P1*
1. ~~VERIFY (Object1), P1 = (the value defined for this property in the EPICS)~~
2. TRANSMIT WritePropertyMultiple-Request,
  - 'Object Identifier' = Object1,
  - 'Property Identifier' = P1,
  - 'Property Value' = (any *valid* value of the correct datatype for this property subject to the restrictions specified in the EPICS as defined in 4.4.2, except the value X read in step 1),



- 'Property Array Index' = (Y, any ~~value~~ positive integer that is larger than that the ~~current~~ supported size of the array)
3. RECEIVE WritePropertyMultiple-Error,  
 'Error Class' = PROPERTY,  
 'Error Code' = INVALID\_ARRAY\_INDEX,  
 'Object Identifier' = Object1,  
 'Property Identifier' = P1,  
 'Property Array Index' = Y
  4. VERIFY (Object1), P1 = X(~~the value defined for this property in the EPICS~~)

[Change **Clause 9.23.2.6**, p. 421]

[Reviewer Note: Added configuration requirements to clarify usage.]

### 9.23.2.6 Writing with a Property Value Having the Wrong Datatype

Purpose: This test case verifies that the IUT correctly responds to an attempt to write a property value that has an invalid datatype.

Test Concept: The TD shall select an object, designated Object1, in the IUT that contains a writable property designated P1. An attempt will be made to write to this property using a datatype that the IUT supports but ~~which is invalid for the property~~ which is not compliant with the property definition given by the BACnet standard.

Configuration Requirements: The value to be written shall not be of a datatype which is compliant with the property definition. If no object supports writable properties, then this test shall be omitted.

Test Steps:

1. READ X = (Object1), P1
2. TRANSMIT WritePropertyMultiple-Request,  
 'Object Identifier' = Object1,  
 'Property Identifier' = P1,  
 'Property Value' = (any value with an invalid datatype)
3. RECEIVE WritePropertyMultiple-Error,  
 'Error Class' = PROPERTY,  
 'Error Code' = INVALID\_DATATYPE,  
 'Object Identifier' = Object1,  
 'Property Identifier' = P1  
 | ( BACnet-Reject-PDU  
   'Reject Reason' =           INVALID\_PARAMETER\_DATATYPE)  
 | (BACnet-Reject-PDU  
   'Reject Reason' =           INVALID\_TAG)
4. VERIFY (Object1), P1 = X(~~the value defined for this property in the EPICS~~)

[Change **Clause 9.23.2.7**, p. 422]

[Reviewer Note: Modified test to remove dependency on EPICS values. Modified to allow this test to be used on all protocol revisions. Modified to allow OPTIONAL\_FUNCTIONALITY\_NOT\_SUPPORTED in some specific cases.]

### 9.23.2.7 Writing with a Property Value that is Out of Range

Purpose: This test case verifies that the IUT can execute WritePropertyMultiple service requests when an attempt is made to write a value that is outside of the supported range. ~~This test shall only be performed if Protocol\_Revision is present and has a value greater than or equal to 4.~~

Test Concept: The TD shall select an object, designated Object1, in the IUT that contains a writable property designated P1. The TD attempts to write to a property using a value that is outside of the supported range.

*Notes to Tester: In this test, the error code `OPTIONAL_FUNCTIONALITY_NOT_SUPPORTED` is permitted only if the value written would require the device to exhibit non-supported optional functionality.*

Test Steps:

1. *READ*  $X = (\text{Object1}), P1$
1. ~~VERIFY (Object1), P1 = (the value defined for this property in the EPICS),~~
2. TRANSMIT WritePropertyMultiple-Request,  
     'Object Identifier' = (Object1, any object with writable properties),  
     'Property Identifier' = (P1, any property with a restricted range of values),  
     'Property Value' = (any value that is outside the supported range)
3. IF (Protocol\_Revision < 4) THEN  
     RECEIVE  
         (WritePropertyMultiple-Error,  
         'Error Class' = PROPERTY,  
         'Error Code' = VALUE\_OUT\_OF\_RANGE,  
         'Object Identifier' = Object1,  
         'Property Identifier' = P1) |  
         (WritePropertyMultiple-Error,  
         'Error Class' = PREOPROPERTY,  
         'Error Code' = OPTIONAL\_FUNCTIONALITY\_NOT\_SUPPORTED,  
         'Object Identifier' = Object1,  
         'Property Identifier' = P1) |  
         (BACnet-Reject-PDU,  
         'Reject Reason' = PARAMETER\_OUT\_OF\_RANGE)  
     ELSE  
         RECEIVE  
             (WritePropertyMultiple-Error,  
             'Error Class' = PROPERTY,  
             'Error Code' = VALUE\_OUT\_OF\_RANGE,  
             'Object Identifier' = Object1,  
             'Property Identifier' = P1) |  
             (WritePropertyMultiple-Error,  
             'Error Class' = PROPERTY,  
             'Error Code' = OPTIONAL\_FUNCTIONALITY\_NOT\_SUPPORTED,  
             'Object Identifier' = Object1,  
             'Property Identifier' = P1)
4. ~~VERIFY (OBJECT1), P1 = X(the value defined for this property in the EPICS)~~

[Add new **Clause 9.23.2.X1**, p. 424]

### 9.23.2.X1 WritePropertyMultiple Reject Test

Purpose: This test case verifies that the IUT does not send a Reject-PDU after applying part of a WritePropertyMultiple.

Test Concept: Two writable properties, P1 and P2, are written to the IUT, but the portion of the WritePropertyMultiple specifying P2 is made invalid by omitting the 'Property Value' parameter. If the IUT returns a Reject, then the value of the first property is checked to ensure it has not changed.

Test Steps:

1. READ OldValue = O1, P1
2. TRANSMIT WritePropertyMultiple-Request,  
     'Object Identifier' = O1,  
     'Property Identifier' = P1,  
     'Property Value' = (NewValue: any value other than OldValue that would be accepted by the IUT for P1)  
     'Object Identifier' = O2,

'Property Identifier' = P2

- ```

3.  RECEIVE WritePropertyMultiple-Error,
    'Error Class' = SERVICES,
    'Error Code' = INVALID_TAG
    'Object Identifier' = O2
    'Property Identifier' = P2) |
RECEIVE BACnet-Reject-PDU,
    'Reject Reason' = INVALID_TAG | MISSING_REQUIRED_PARAMETER
                        | INCONSISTENT_PARAMETERS | INVALID_PARAMETER_DATA_TYPE
                        | TOO_MANY_ARGUMENTS)
4.  IF (a WritePropertyMultiple-Error was received in step 3) THEN
    VERIFY (O1), P1 = NewValue
ELSE -- a Reject-PDU was received
    VERIFY (O1), P1 = OldValue

```

[Add new **Clause 9.23.2.X2**, p. 424]

### 9.23.2.X2 Resizing a writable fixed size array property using WritePropertyMultiple service

Purpose: This test case verifies that the IUT correctly responds to an attempt to resize a writable fixed size array property using WritePropertyMultiple service.

Test Concept: Select an object(O1) in the IUT that contains a writable array property of a fixed size. This property is designated P1. If no suitable object can be found, then this test shall be omitted.

### Test Steps:

1. READ X = (O1), P1, ARRAY INDEX = 0
2. TRANSMIT WritePropertyMultiple-Request,  
    'Object Identifier' = O1,  
    'Property Identifier' = P1,  
    'Property Value' = (Entire Array with any valid value greater than Array Size X)
3. RECEIVE WritePropertyMultiple-Error,  
    'Error Class' = PROPERTY,  
    'Error Code' = INVALID\_ARRAY\_INDEX | VALUE\_OUT\_OF\_RANGE,  
    'ObjectIdentifier' = O1,  
    'PropertyIdentifier' = P1
4. VERIFY (O1), P1= X, ARRAY INDEX = 0
5. TRANSMIT WritePropertyMultiple-Request,  
    'Object Identifier' = O1,  
    'Property Identifier' = P1,  
    'Property Value' = (Entire Array with any valid value less than Array Size X)
6. RECEIVE WritePropertyMultiple-Error,  
    'Error Class' = PROPERTY,  
    'Error Code' = INVALID\_ARRAY\_INDEX | VALUE\_OUT\_OF\_RANGE,  
    'ObjectIdentifier' = O1,  
    'PropertyIdentifier' = P1
7. VERIFY (O1), P1= X, ARRAY INDEX = 0
8. TRANSMIT WritePropertyMultiple-Request,  
    'Object Identifier' = O1,  
    'Property Identifier' = P1,  
    'Property Value' = (any valid value greater than Array Size X),  
    'Property Array Index' = 0
9. RECEIVE WritePropertyMultiple-Error,  
    'Error Class' = PROPERTY,  
    'Error Code' = INVALID\_ARRAY\_INDEX | VALUE\_OUT\_OF\_RANGE,  
    'ObjectIdentifier' = O1,  
    'PropertyIdentifier' = P1

'Property Array Index'=0

10. VERIFY (O1), P1= X, ARRAY INDEX = 0
11. TRANSMIT WritePropertyMultiple-Request,  
     'Object Identifier' = O1,  
     'Property Identifier' = P1,  
     'Property Value' = (any valid value less than Array Size X),  
     'Property Array Index' = 0
12. RECEIVE WritePropertyMultiple-Error,  
     'Error Class' = PROPERTY,  
     'Error Code' = INVALID\_ARRAY\_INDEX | VALUE\_OUT\_OF\_RANGE,  
     'ObjectIdentifier' = O1,  
     'PropertyIdentifier' = P1  
     'Property Array Index' = 0
13. VERIFY (O1), P1= X, ARRAY INDEX = 0

[Add new **Clause 9.23.2.X3**, p. 424]

### 9.23.2.X3 Writing first element of 'List of Write Access Specifications' with Object Access Error

Purpose: To verify the ability to correctly execute a WritePropertyMultiple service request for which the first element of the 'List of Write Access Specifications' contains a specification for an unsupported object and all writes after the first failed write attempt do not take place.

Test Concept: An attempt is made to write to a single property in two different objects. The first object is not supported. The second object is supported, and the property is writable. The objective is to verify that an appropriate error response is returned and that all writes after the first failed write attempt do not take place.

Configuration Requirements: If the IUT supports any writable scalar properties that are not commandable it shall be configured with one for use in this test. If no such properties are supported the IUT shall be configured with a writable array or commandable property and the test steps modified to account for this variation. In the test description O2 and P2 will be used to designate the writable object and property having value X used for this test. The designation Bad Object will be used to indicate an object that is not supported or not present in IUT database P1 is any valid Property Identifier.

Test Steps:

1. VERIFY (O2), P2 = X
2. TRANSMIT WritePropertyMultiple-Request,  
     'Object Identifier' = BadObject,  
     'Property Identifier' = P1,  
     'Property Value' = (any valid value of the appropriate datatype for this property subject to the restrictions specified in the EPICS as defined in 4.4.2)  
     'Object Identifier' = O2,  
     'Property Identifier' = P2,  
     'Property Value' = (any valid value not equal to X),
3. RECEIVE WritePropertyMultiple-Error,  
     'Error Class' = OBJECT,  
     'Error Code' = UNKNOWN\_OBJECT,  
     'Object Identifier' = BadObject,  
     'Property Identifier' = P1 |  
     (RECEIVE WritePropertyMultiple-Error,  
         'Error Class' = OBJECT,  
         'Error Code' = UNSUPPORTED\_OBJECT\_TYPE,  
         'Object Identifier' = BadObject,  
         'Property Identifier' = P1)
4. VERIFY (O2), P2 = X

[Add new **Clause 9.23.2.X4**, p. 424]

### 9.23.2.X4 Writing first element of 'List of Write Access Specifications' with a Write Access Error

Purpose: To verify the ability to correctly execute a WritePropertyMultiple service request for which the first element of the 'List of Write Access Specifications' contains a specification for a read only property and all writes after the first failed write attempt do not take place.

Test Concept: An attempt is made to write to two properties in a single object. The first property is supported but read only. The second property is supported and writable. The objective is to verify that an appropriate error response is returned and that all writes after the first failed write attempt do not take place.

Configuration Requirements: If the IUT supports any writable scalar properties that are not commandable, it shall be configured with one for use in this test. If no such properties are supported, the IUT shall be configured with a writable array or commandable property and the test steps modified to account for this variation. In the test description, O1 will be used to designate the object, P1 the read only property having value X, P2 the writable property having value Y used for this test.

Test Steps:

1. VERIFY (O1), P1= X
2. VERIFY (O1), P2=Y
3. TRANSMIT WritePropertyMultiple-Request,  
'Object Identifier' = O1,  
'Property Identifier' = P1,  
'Property Value' = X,  
'Property Identifier' = P2,  
'Property Value' = (any valid value not equal to Y)
4. RECEIVE WritePropertyMultiple-Error,  
'Error Class' = PROPERTY,  
'Error Code' = WRITE\_ACCESS\_DENIED,  
'Object Identifier' = O1,  
'Property Identifier' = P1
5. VERIFY (O1), P2 = Y

[Add new **Clause 9.23.2.X5**, p. 424]

### 9.23.2.X5 WritePropertyMultiple Reject Test for first element of 'List of Write Access Specifications'

Purpose: This test case verifies that if IUT does sends a Reject-PDU or Error-PDU, then the write attempt for the remaining element of 'List of Write Access Specifications' do not take place.

Test Concept: Two writable properties, P1 having value X and P2 having value Y, are written to the IUT, but the portion of the WritePropertyMultiple specifying P1 is made invalid by omitting the 'Property Value' parameter. The value of the properties are checked to ensure that it has not changed.

Test Steps:

1. VERIFY (O1), P1= X
2. VERIFY (O2), P2=Y
3. TRANSMIT WritePropertyMultiple-Request,  
'Object Identifier' = O1,  
'Property Identifier' = P1,  
'Object Identifier' = O2,  
'Property Identifier' = P2  
'Property Value' = (Any valid value not equal to Y))
4. RECEIVE WritePropertyMultiple-Error,  
'Error Class' = SERVICES,  
'Error Code' = INVALID\_TAG  
'Object Identifier' = O1  
'Property Identifier' = P1) |

(RECEIVE BACnet-Reject-PDU,

'Reject Reason' = INVALID\_TAG | MISSING\_REQUIRED\_PARAMETER  
| INCONSISTENT\_PARAMETERS | INVALID\_PARAMETER\_DATA\_TYPE  
| TOO\_MANY\_ARGUMENTS)

4. VERIFY (O1), P1 = X
5. VERIFY (O2), P2 = Y

[Add new **Clause 9.23.2.X6**, p. 424]

### 9.23.2.X6 Writing first element of 'List of Write Access Specifications' with a Property Access Error

Purpose: To verify the ability to correctly execute a WritePropertyMultiple service request for which the first element of the 'List of Write Access Specifications' contains a specification for an unsupported property and all writes after the first failed write attempt do not take place.

Test Concept: An attempt is made to write to two properties in a single object. The first property is not supported for this object. The second property is supported for this object and writable. The objective is to verify that an appropriate error response is returned and that all writes after the first failed write attempt do not take place.

Configuration Requirements: If the IUT supports any writable scalar properties that are not commandable, it shall be configured with one for use in this test. If no such properties are supported, the IUT shall be configured with a writable array or commandable property and the test steps modified to account for this variation. In the test description, O1 will be used to designate the object, P1 the unsupported property, and P2 the writable property having value X used.

Test Steps:

1. VERIFY (O1), P2 = X
2. TRANSMIT WritePropertyMultiple-Request,  
'Object Identifier' = O1,  
'Property Identifier' = P1,  
'Property Value' = (any valid value of the appropriate datatype for this property subject to the restrictions specified in the EPICS as defined in 4.4.2)  
  
'Property Identifier' = P2,  
'Property Value' = (any valid value not equal to X),
3. RECEIVE WritePropertyMultiple-Error,  
'Error Class' = PROPERTY,  
'Error Code' = UNKNOWN\_PROPERTY,  
'Object Identifier' = O1,  
'Property Identifier' = P1
4. VERIFY (O1), P2 = X

[Add new **Clause 9.23.2.X7**, p. 424]

### 9.23.2.X7 Writing a Property Value to a Non-supported Optional Functionality

Purpose: To verify that the IUT can execute WritePropertyMultiple service requests when an attempt is made to write a value that would require a non-supported optional functionality

Test Concept: The TD attempts to write to a property using a value that would require a non-supported optional functionality.

Test Steps:

1. READ X = (Object1), P1
2. TRANSMIT WritePropertyMultiple-Request,  
'Object Identifier' = (Object1, any object with writable properties),  
'Property Identifier' = (P1, any property with a restricted range of values),  
'Property Value' = (a value that would require a non-supported optional functionality)
3. RECEIVE WritePropertyMultiple-Error,

'Error Class' = PROPERTY,  
 'Error Code' = OPTIONAL\_FUNCTIONALITY\_NOT\_SUPPORTED  
 'Object Identifier' = Object1,  
 'Property Identifier' = P1

4. VERIFY (Object1), P1 = X

[Add new **Clause 9.23.2.X9**, p. 424]

### 9.23.2.X9 Date Non-Pattern Properties Test using WritePropertyMultiple Service

Purpose: To verify that the property being tested does not accept special date field values.

Test Concept: The property being tested, P1, is written with each of the special date field values to ensure that the property does not accept them. A date is selected which is within the date range that the IUT will accept for the property. The value, V1, written to the property is the date D1 with one of its fields replaced with one of the date special values. If the property is a complex datatype, the other fields in the value shall be set within the range accepted by the IUT. This test shall only be applied to devices claiming Protocol\_Revision 11 or higher.

Notes to Tester: if P1 is an array, then a non-zero array index may be provided in the TRANSMIT and the same array index observed in the WritePropertyMultiple-Error.

Test Steps:

1. REPEAT SV = (year unspecified, month unspecified, day of month unspecified,  
                   day of week unspecified, odd months, even months, last day of month,  
                   even days, odd days) DO {
2. TRANSMIT WritePropertyMultiple-Request  
     'Object Identifier' = O1,  
     'Property Identifier' = P1,  
     'Property Value' = (V1 updated with the special value SV)
3. RECEIVE WritePropertyMultiple-Error,  
     'Error Class' = PROPERTY,  
     'Error Code' = VALUE\_OUT\_OF\_RANGE,  
     'Object Identifier' = Object1,  
     'Property Identifier' = P1)  
   | (BACnet-Reject-PDU  
     'Reject Reason' = INVALID\_PARAMETER\_DATATYPE)  
   | (BACnet-Reject-PDU  
     'Reject Reason' = INVALID\_TAG)  
   }

[Add new **Clause 9.23.2.X10**, p. 424]

### 9.23.2.X10 Time Non-Pattern Properties Test using WritePropertyMultiple Service

Purpose: To verify that the property being tested does not accept special time field values.

Test Concept: The property being tested, P1, is written with each of the special time field values to ensure that the property does not accept them. A time is selected which is within the time range that the IUT will accept for the property. The value, V1, written to the property is the time T1 with one of its fields replaced with one of the time special values. If the property is a complex datatype, the other fields in the value shall be set within the range accepted by the IUT. This test shall only be applied to devices claiming Protocol\_Revision 11 or higher.

Notes to Tester: if P1 is an array, then a non-zero array index may be provided in the TRANSMIT and the same array index observed in the WritePropertyMultiple-Error.

Test Steps:

1. REPEAT SV = (hour unspecified, minute unspecified, second unspecified, hundredths unspecified) Do {
2. TRANSMIT WritePropertyMultiple-Request  
'Object Identifier' = O1,  
'Property Identifier' = P1,  
'Property Value' = (V1 updated with the special value SV)
3. RECEIVE WritePropertyMultiple-Error,  
'Error Class' = PROPERTY,  
'Error Code' = VALUE\_OUT\_OF\_RANGE,  
'Object Identifier' = Object1,  
'Property Identifier' = P1  
| (BACnet-Reject-PDU  
'Reject Reason' = INVALID\_PARAMETER\_DATATYPE)  
| (BACnet-Reject-PDU  
'Reject Reason' = INVALID\_TAG)  
}

[Add new **Clause 9.23.2.X11**, p. 424]

### 9.23.2.X11 DateTime Non-Pattern Properties Test using WritePropertyMultiple Service

Purpose: To verify that the property being tested does not accept special date field values.

Test Concept: The property being tested, P<sub>1</sub>, is written with each of the special datetime field values to ensure that the property does not accept them. A datetime DT<sub>1</sub> is selected which is within the range that the IUT will accept for the property. The value, V<sub>1</sub>, written to the property is the datetime DT<sub>1</sub> with one of its fields replaced with one of the date or time special values. If the property is a complex datatype, the other fields in the value shall be set within the range accepted by the IUT. This test shall only be applied to devices claiming Protocol\_Revision 11 or higher.

Notes to Tester: if P<sub>1</sub> is an array, then a non-zero array index may be provided in the TRANSMIT and the same array index observed in the WritePropertyMultiple-Error.

Test Steps:

1. REPEAT SV = (year unspecified, month unspecified, day of month unspecified, odd months, even months, last day of month, even days, odd days, hour unspecified, minute unspecified, second unspecified, hundredths unspecified) DO {
2. TRANSMIT WritePropertyMultiple-Request,  
'Object Identifier' = O1,  
'Property Identifier' = P1,  
'Property Value' = (DT<sub>1</sub> updated with the special value SV)
3. RECEIVE WritePropertyMultiple-Error,  
'Error Class' = PROPERTY,  
'Error Code' = VALUE\_OUT\_OF\_RANGE,  
'Object Identifier' = Object1,  
'Property Identifier' = P1)  
| (BACnet-Reject-PDU  
'Reject Reason' = INVALID\_PARAMETER\_DATATYPE)  
| (BACnet-Reject-PDU  
'Reject Reason' = INVALID\_TAG)  
}

[Add new **Clause 9.23.2.X12**, p. 424]

### 9.23.2.X12 BACnetDateRange Non-Pattern Properties Test using WritePropertyMultiple Service

Purpose: To verify that the property being tested does not accept special date field values, except for fully unspecified start of the range or fully unspecified end of the range, or both.



Test Concept: A BACnetDateRange property, or property that is a complex datatype containing BACnetDateRange P1 is written with each of the special field values to ensure that the property does not accept them. Each half of the dateRange DR1 is selected so it is within the range that the IUT will accept for the property. The value, V1, written to the property is the dateRange DR1 with one of its fields replaced with one of the date special values. If the property is a complex datatype, the other fields in the value shall be set within the range accepted by the IUT. This test shall only be applied to devices claiming Protocol\_Revision 11 or higher.

Notes to Tester: if P1 is an array, then a non-zero array index may be provided in the TRANSMIT and the same array index observed in the WritePropertyMultiple-Error.

Test Steps:

5. REPEAT SV = (year unspecified, month unspecified, day of month unspecified, day of week unspecified, odd months, even months, last day of month, even days, odd days) DO {
6. TRANSMIT WritePropertyMultiple-Request,  
     'Object Identifier' = O1,  
     'Property Identifier' = P1,  
     'Property Value' = (DR1 with startDate updated with special value SV)
7. RECEIVE WritePropertyMultiple-Error,  
     'Error Class' = PROPERTY,  
     'Error Code' = VALUE\_OUT\_OF\_RANGE,  
     'Object Identifier' = Object1,  
     'Property Identifier' = P1  
     | (BACnet-Reject-PDU  
         'Reject Reason' = INVALID\_PARAMETER\_DATATYPE)  
     | (BACnet-Reject-PDU  
         'Reject Reason' = INVALID\_TAG)
8. TRANSMIT WritePropertyMultiple-Request,  
     'Object Identifier' = O1,  
     'Property Identifier' = P1,  
     'Property Value' = (DR1 with endDate updated with special value SV)
9. RECEIVE WritePropertyMultiple-Error,  
     'Error Class' = PROPERTY,  
     'Error Code' = VALUE\_OUT\_OF\_RANGE,  
     'Object Identifier' = Object1,  
     'Property Identifier' = P1)  
     | (BACnet-Reject-PDU  
         'Reject Reason' = INVALID\_PARAMETER\_DATATYPE)  
     | (BACnet-Reject-PDU  
         'Reject Reason' = INVALID\_TAG)
- }

## 9.24 DeviceCommunicationControl Service Execution Test

### 9.24.1 Positive DeviceCommunicationControl Service Execution Tests

[Change **Clause 9.24.1.5**, p. 426]

[Reviewer Note: Modified test to remove dependency on EPICS values.]

#### 9.24.1.5 Finite Time Duration Restored by ReinitializeDevice

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

#### Test Steps:

1. *READ Y = (Device, X), Object\_Name*
2. 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)
3. RECEIVE BACnet-SimpleACK-PDU
4. WAIT **Internal Processing Fail Time**
5. TRANSMIT ReadProperty-Request,  
     'Object Identifier' = (Device, X),  
     'Property Identifier' = (any required non-array property of the Device object)
6. WAIT (an arbitrary time  $> \text{Internal Processing Fail Time}$  selected by the tester, and  $< T$  as specified in the DeviceCommunicationControl-Request)
7. CHECK (Verify that the IUT has not transmitted any messages since the acknowledgment in step 2.)
8. TRANSMIT ReinitializeDevice-Request,  
     'Reinitialize State of Device' = WARMSTART,  
     'Password' = (any appropriate password as described in the Configuration Requirements)
9. RECEIVE ~~BACnet-Simple-ACK-PDU~~ BACnet-SimpleACK-PDU
10. CHECK (Did the IUT perform a WARMSTART reboot?)
11. VERIFY (Device, X), *Object\_Name = Y* (~~any required non-array property~~) = (~~the value for this property as described in the EPICS~~)

[Add new **Clause 9.24.1.11**, p. 429]

[Reviewer Note: This new test addressed the requirement of 135-2016bi-2 and BTL-CR-0492].

#### 9.24.1.11 Ensure that DISABLE option is not supported by IUT claiming PR $\geq 20$

Purpose: To verify that IUT claiming Protocol Revision (PR) greater than or equal to 20, does not accept 'Enable/Disable' parameter equal to DISABLE in the DeviceCommunicationControl request.

Test Concept: Send DeviceCommunicationControl request with 'Enable/Disable' parameter equal to DISABLE to IUT. Then IUT is verified that it correctly responds with a Result(-).

Configuration Requirements: If the IUT does not support an internal clock this test shall be tested with indefinite time duration.

#### Test Steps:

1. TRANSMIT DeviceCommunicationControl-Request  
     'Time Duration' = (a value  $T > 1$ , in minutes) | (no value)  
     'Enable/Disable' = DISABLE,  
     'Password' = Any appropriate password if required
2. RECEIVE BACnet-Error-PDU  
     Error Class = SERVICES,  
     Error Code = SERVICE\_REQUEST\_DENIED
3. VERIFY (Device, X), System\_Status = (any valid value)

[Add new **Clause 9.24.1.12**, p. 429]

#### 9.24.1.12 Disable of Service Initiation Restored by ReinitializeDevice

Purpose: To verify the correct execution of the DeviceCommunicationControl service when DISABLE\_INITIATION is requested with a finite time duration. Communication is restored using the ReinitializeDevice service.

Configuration Requirements: The IUT shall be configured to initiate client requests.

#### Test Steps:

1. MAKE (a condition that would normally cause the IUT to initiate requests)
2. CHECK (that the IUT is initiating requests)
3. TRANSMIT DeviceCommunicationControl-Request,  
     'Time Duration' = (a value in minutes > time required to execute all test steps),  
     'Enable/Disable' = DISABLE\_INITIATION,  
     'Password' = (any appropriate password if required)
4. RECEIVE BACnet-SimpleACK-PDU
5. MAKE (a condition that would normally cause IUT to initiate requests)
6. CHECK (that the IUT has stopped initiating requests)
7. VERIFY (any supported property) = (any valid value)
8. TRANSMIT Who-Is-Request
9. RECEIVE I-Am-Request
10. TRANSMIT ReinitializeDevice-Request,  
     'Reinitialized State of Device' = WARMSTART,  
     'Password' = (any appropriate password)
11. RECEIVE BACnet-SimpleACK-PDU
12. CHECK (Did the IUT perform a WARMSTART reboot?)
13. MAKE (a condition that would normally cause the IUT to initiate requests)
14. CHECK (that the IUT is initiating requests)

### 9.24.2 Negative DeviceCommunicationControl Service Execution Tests

[Change **Clause 9.24.2.1**, p. 429]

[Reviewer Note: Modified the parameter value from DISABLE to DISABLE\_INITIATION in step 1.]

#### 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\_INITIATION*, ~~DISABLE~~  
     'Password' = (any invalid password)
2. RECEIVE BACnet-Error-PDU,  
     Error Class = SECURITY,  
     Error Code = PASSWORD\_FAILURE
3. VERIFY (Device, X), System\_Status = (any valid value)

[Change **Clause 9.24.2.2**, p. 429]

[Reviewer Note: Modified the parameter value from DISABLE to DISABLE\_INITIATION in step 1.]

#### 9.24.2.2 Missing Password

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

#### Test Steps:

1. TRANSMIT DeviceCommunicationControl-Request,  
'Enable/Disable' = *DISABLE\_INITIATION* ~~DISABLE~~
2. IF (Protocol\_Revision >= 7) THEN  
RECEIVE BACnet-Error-PDU,  
Error Class = SECURITY,  
Error Code = PASSWORD\_FAILURE  
ELSE  
RECEIVE BACnet-Error-PDU,  
Error Class = SECURITY,  
Error Code = PASSWORD\_FAILURE |  
(RECEIVE BACnet-Error-PDU,  
Error Class = SERVICES,  
Error Code = MISSING\_REQUIRED\_PARAMETER)
3. VERIFY (Device, X), System\_Status = (any valid value)

[Change **Clause 9.24.2.3**, p. 430]

[Reviewer Note: Added support for additional error codes.]

### 9.24.2.3 Restore by ReinitializeDevice with Invalid 'Reinitialized State of Device'

Purpose: To verify the communications are not restored when a ReinitializeDevice request is received that contains one of the backup or restore related values for service parameter 'Reinitialized State of Device'.

Test Concept: Disable the IUT's communications for a period time, T, longer than it will take to complete the test. Verify that, while communications are disabled, the IUT correctly responds with a Result(-) when it receives a ReinitializeDevice request containing a backup or restore related values.

Test Steps:

1. TRANSMIT DeviceCommunicationControl-Request,  
'Enable/Disable' = DISABLE  
'Password' = (any appropriate password),  
'Time Duration' = (a value T >= 1, in minutes) | (no value)
2. ~~RECEIVE BACnet-Simple-ACK-PDU~~ *BACnet-SimpleACK-PDU*
3. WAIT **Internal Processing Fail Time**
4. TRANSMIT ReinitializeDevice-Request,  
'Reinitialized State of Device' = STARTBACKUP | ENDBACKUP |  
STARTRESTORE | ENDRESTORE | ABORTRESTORE,  
'Password' = (any appropriate password)
5. IF (Protocol\_Revision is present and Protocol\_Revision >= 7) THEN  
*IF (Device supports DM-BR-B) THEN*  
RECEIVE BACnet-Error-PDU,  
'Error Class' = SERVICES,  
'Error Code' = COMMUNICATION\_DISABLED  
ELSE  
*RECEIVE BACnet-Error-PDU,*  
*'Error Class' = SERVICES,*  
*'Error Code' = COMMUNICATION\_DISABLED |*  
*OPTIONAL\_FUNCTIONALITY\_NOT\_SUPPORTED*  
ELSE  
CHECK(that the IUT responded with BACnet-Error-PDU with an Error Class of SERVICES and *any appropriate*  
~~Error Code of COMMUNICATION\_DISABLED~~, or *that* the IUT did not respond at all)
6. TRANSMIT DeviceCommunicationControl-Request,  
'Enable/Disable' = ENABLE  
'Password' = (any appropriate password),
7. RECEIVE BACnet-Simple-ACK-PDU

## 9.25 ConfirmedPrivateTransfer Service Execution Tests

[Add new **Clause 9.25.1**, p. 430]

[Reviewer Note: New Clause number and title added so negative tests could be separated from positive tests.]

### 9.25.1 Positive ConfirmedPrivateTransfer Service Execute Tests

[Add new **Clause 9.25.1.1**, p. 430]

[Reviewer Note: Only the clause number and title were added. Except for the removal of the Dependencies and BACnet Reference Clause sections, the content of 9.25.1.1 is identical to the original 9.25 test.]

#### 9.25.1.1 Correctly Executes a Supported ConfirmedPrivateTransfer Service

~~Dependencies: None.~~

~~BACnet Reference Clause: 16.2.~~

Purpose: To verify the ability to correctly execute a ConfirmedPrivateTransfer service request.

Test Concept: The service procedure implied by a particular private transfer service is defined by the vendor. This test simply verifies that an appropriate acknowledgment is returned and that any externally visible actions defined by the vendor are observed.

Configuration Requirements: The IUT shall be configured to execute at least one ConfirmedPrivateTransfer service. The service parameters that are to be provided in the request and a list of any externally visible actions that should be apparent to the tester shall also be provided.

Test Steps:

1. TRANSMIT ConfirmedPrivateTransfer-Request,  
    'Vendor ID' = (the Vendor\_Identifier specified in the Device object of the  
                  EPICS),  
    'Service Number' = (any service number provided by the vendor),  
    'Service Parameters' = (the service parameters provided for this service)
2. RECEIVE ConfirmedPrivateTransfer-ACK,  
    'Vendor ID' = (the Vendor\_Identifier specified in the Device object of the  
                  EPICS),  
    'Service Number' = (the service number used in step 1),  
    'Result Block' = (the expected results provided by the vendor)
3. CHECK (Did the externally visible actions take place?)

[Add new **Clause 9.25.2**, p. 431]

[Reviewer Note: New Clause number and title added so negative tests could be separated from positive tests.]

### 9.25.2 Negative ConfirmedPrivateTransfer Service Execute Tests

[Add new **Clause 9.25.2.1**, p. 431]

[Reviewer Note: There is no existing test verifying correct error responses for unknown ConfirmedPrivateTransfer services.]

### 9.25.2.1 Correctly Executes a Non-Supported ConfirmedPrivateTransfer Service

Purpose: To verify that the IUT correctly responds with an error when an unsupported ConfirmedPrivateTransfer is received.

Test Concept: Send a non-supported ConfirmedPrivateTransfer request to the IUT and verify that it responds with an error or rejects the service.

Test Steps:

1. TRANSMIT ConfirmedPrivateTransfer-Request,  
     'Vendor ID' = (VID: any valid value not supported by the IUT),  
     'Service Number' = (SID: any valid value, that when combined with Vendor ID is not supported by the IUT),  
     'Service Parameters' = (any valid values)
2. RECEIVE  
     (ConfirmedPrivateTransfer-Error,  
     'Error Class' = any  
     'Error Code' = any  
     'Vendor ID' = (VID),  
     'Service Number' = (SID),  
     'Result Block' = (the expected results provided by the vendor)) |  
     (BACnet-Reject-PDU,  
     'Reject Reason' = any reason) |  
     (BACnet-Abort-PDU  
     'Abort Reason' = any reason)
3. CHECK (that the IUT exhibits the vendor defined results)

## 9.27 ReinitializeDevice Service Execution Tests

### 9.27.2 Negative ReinitializeDevice Service Execution Tests

[Remove Clause 9.27.2.1, p. 432]

#### ~~9.27.2.1 COLDSTART with an Invalid Password~~

~~Purpose: To verify the correct execution of the ReinitializeDevice request service procedure when a COLDSTART is attempted and an invalid password is provided. If the IUT does not provide password protection this test case shall be omitted.~~

~~Test Steps:~~

- ~~1. TRANSMIT ReinitializeDevice Request,  
     'Reinitialized State of Device' = COLDSTART,  
     'Password' = (any invalid password)~~
- ~~2. RECEIVE BACnet Error PDU,  
     Error Class = SECURITY,  
     Error Code = PASSWORD\_FAILURE~~
- ~~3. CHECK (Did the IUT reboot?)~~

~~Notes to Tester: The IUT shall not reinitialize.~~

[Remove Clause 9.27.2.2, p. 433]

#### ~~9.27.2.2 WARMSTART with an Invalid Password~~

~~Purpose: To verify the correct execution of the ReinitializeDevice request service procedure when a WARMSTART is attempted and an invalid password is provided. If the IUT does not provide password protection this test case shall be omitted.~~

#### Test Steps:

1. ~~TRANSMIT ReinitializeDevice-Request,~~  
~~'Reinitialized State of Device' = WARMSTART,~~  
~~'Password' = (any invalid password)~~
2. ~~RECEIVE BACnet-Error-PDU,~~  
~~Error Class = SECURITY,~~  
~~Error Code = PASSWORD\_FAILURE~~
3. ~~CHECK (Did the IUT reboot?)~~

Notes to Tester: The IUT shall not reinitialize.

[Change and renumber **Clause 9.27.2.3**, p. 433]

[Reviewer Note: Updated test to also test invalid password usage.]

#### ~~9.27.2.3~~9.27.2.1 COLDSTART with Missing or Invalid Password

Purpose: To verify that the correct BACnet Error PDU is returned when a COLDSTART is attempted, and *the password is invalid or* a password is required but no password is provided.

Configuration Requirements: The IUT shall be configured to require a password for ReinitializeDevice.

#### Test Steps:

1. TRANSMIT ReinitializeDevice-Request,  
'Reinitialized State of Device' = COLDSTART;
2. IF (Protocol\_Revision is present and Protocol\_Revision >= 7) THEN  
RECEIVE BACnet-Error-PDU,  
'Error Class' = SECURITY,  
'Error Code' = PASSWORD\_FAILURE  
ELSE  
(RECEIVE BACnet-Error-PDU,  
'Error Class' = SECURITY,  
'Error Code' = PASSWORD\_FAILURE) |  
(RECEIVE BACnet-Error-PDU,  
'Error Class' = SERVICES,  
'Error Code' = SERVICE\_REQUEST\_DENIED) |  
(BACnet-Error-PDU,  
'Error Class' = SERVICES,  
'Error Code' = MISSING\_REQUIRED\_PARAMETER)  
3. CHECK (The IUT did NOT perform a COLDSTART reboot)
4. TRANSMIT ReinitializeDevice-Request,  
'Reinitialized State of Device' = COLDSTART,  
'Password' = (any invalid password)
5. IF (Protocol\_Revision is present and Protocol\_Revision >= 7) THEN  
RECEIVE BACnet-Error-PDU,  
'Error Class' = SECURITY,  
'Error Code' = PASSWORD\_FAILURE  
ELSE  
(RECEIVE BACnet-Error-PDU,  
'Error Class' = SECURITY,  
'Error Code' = PASSWORD\_FAILURE) |  
(RECEIVE BACnet-Error-PDU,  
'Error Class' = SERVICES,  
'Error Code' = SERVICE\_REQUEST\_DENIED) |  
(BACnet-Error-PDU,  
'Error Class' = SERVICES,

'Error Code' = MISSING\_REQUIRED\_PARAMETER)

6. CHECK (The IUT did NOT perform a COLDSTART reboot)

[Change and renumber **Clause 9.27.2.4**, p. 433]

[Reviewer Note: Updated test to also test invalid password usage.]

#### **9.27.2.49.27.2.2 WARMSTART with Missing or Invalid Password**

Purpose: To verify that the correct BACnet Error PDU is returned when a WARMSTART is attempted and *the password is invalid or* a password is required but no password is provided.

*Configuration Requirements: The IUT shall be configured to require a password for ReinitializeDevice.*

*Notes to Tester: External indications that the IUT has reinitialized, such as LEDs or startup message traffic, shall be used to confirm reinitialization whenever possible.*

Test Steps:

1. TRANSMIT ReinitializeDevice-Request,  
'Reinitialized State of Device' = WARMSTART;
2. IF (Protocol\_Revision is present and Protocol\_Revision >= 7) THEN  
RECEIVE BACnet-Error-PDU,  
'Error Class' = SECURITY,  
'Error Code' = PASSWORD\_FAILURE  
ELSE  
RECEIVE BACnet-Error-PDU,  
'Error Class' = SECURITY,  
'Error Code' = PASSWORD\_FAILURE |  
(RECEIVE BACnet-Error-PDU,  
'Error Class' = SERVICES,  
'Error Code' = SERVICE\_REQUEST\_DENIED) |  
(BACnet-Error-PDU,  
'Error Class' = SERVICES,  
'Error Code' = MISSING\_REQUIRED\_PARAMETER)
3. CHECK (The IUT did NOT perform a WARMSTART reboot)
4. TRANSMIT ReinitializeDevice-Request,  
'Reinitialized State of Device' = WARMSTART,  
'Password' = (any invalid password)
5. IF (Protocol\_Revision is present and Protocol\_Revision >= 7) THEN  
RECEIVE BACnet-Error-PDU,  
'Error Class' = SECURITY,  
'Error Code' = PASSWORD\_FAILURE  
ELSE  
(RECEIVE BACnet-Error-PDU,  
'Error Class' = SECURITY,  
'Error Code' = PASSWORD\_FAILURE) |  
(RECEIVE BACnet-Error-PDU,  
'Error Class' = SERVICES,  
'Error Code' = SERVICE\_REQUEST\_DENIED) |  
(BACnet-Error-PDU,  
'Error Class' = SERVICES,  
'Error Code' = MISSING\_REQUIRED\_PARAMETER)
6. CHECK (The IUT did NOT perform a WARMSTART reboot)

~~Notes to Tester: External indications that the IUT has reinitialized, such as LEDs or startup message traffic, shall be used to confirm reinitialization whenever possible.~~



[Add new **Clause 9.27.2.X**, p. 433]

[Reviewer Note: Added to ensure that non-supported 'Reinitialized State of Device' are properly rejected.]

### **9.27.2.X Rejects Unsupported Reinitialize Types**

Purpose: Verify that IUT correctly rejects unsupported 'Reinitialized State of Device' values.

Test Concept: Send each unsupported 'Reinitialized State of Device' value to the device and ensure that it correctly rejects the value.

Test Steps:

1. REPEAT S = (each unsupported 'Reinitialized State of Device' value) {  
    TRANSMIT ReinitializeDevice  
        'Reinitialized State of Device' = S,  
        'Password' = (any valid value)  
    RECEIVE BACnet-Error-PDU  
        'Error Class' = SERVICES,  
        'Error Code' = VALUE\_OUT\_OF\_RANGE |  
                            OPTIONAL\_FUNCTIONALITY\_NOT\_SUPPORTED  
}

[Change **Clause 9.29**, p. 435]

[Reviewer Note: Add test support for the UnconfirmedTextMessage service.]

## **9.29 UnconfirmedTextMessage Service Execution Tests**

~~BACnet does not define a service procedure for executing the UnconfirmedTextMessage service and thus no tests are needed.~~

[Add new **Clause 9.29.1**, p. 435]

### **9.29.1 UnconfirmedTextMessage With No Message Class**

Purpose: To verify the correct execution of the UnconfirmedTextMessage service request when no 'Message Class' is provided.

Notes to Tester: The IUT shall respond with the indicated message and perform any vendor-specified action that is appropriate.

Test Steps:

1. TRANSMIT UnconfirmedTextMessage-Request,  
    'Text Message Source Device' = TD,  
    'Message Priority' = NORMAL,  
    'Message' = (any CharacterString)
2. CHECK (Did any vendor specified action for these circumstances occur?)

[Add new **Clause 9.29.2**, p. 435]

### **9.29.2 UnconfirmedTextMessage With an Unsigned Message Class**

Purpose: To verify the correct execution of the UnconfirmedTextMessage service request when the Unsigned form of the 'Message Class' is used.

Configuration Requirements: The vendor shall provide a list of supported Unsigned message classes.

Notes to Tester: The IUT shall respond with the indicated message and perform any vendor-specified action that is appropriate.

Test Steps:

1. TRANSMIT UnconfirmedTextMessage-Request,  
    'Text Message Source Device' = TD,  
    'Message Class' = (any Unsigned value from the list provided by the vendor),  
    'Message Priority' = NORMAL,  
    'Message' = (any CharacterString)
2. CHECK (Did any vendor specified action for these circumstances occur?)

[Add new **Clause 9.29.3**, p. 435]

### **9.29.3 UnconfirmedTextMessage With a CharacterString Message Class**

Purpose: To verify the correct execution of the UnconfirmedTextMessage service request when the CharacterString form of the 'Message Class' is used.

Configuration Requirements: The vendor shall provide a list of supported CharacterString message classes.

Notes to Tester: The IUT shall respond with the indicated message and perform any vendor-specified action that is.

Test Steps:

1. TRANSMIT UnconfirmedTextMessage-Request,  
    'Text Message Source Device' = TD,  
    'Message Class' = (any CharacterString value from the list provided by the vendor),  
    'Message Priority' = NORMAL,  
    'Message' = (any CharacterString)
2. CHECK(Did any vendor specified action for these circumstances occur?)

[Change **Clause 9.30**, p. 435]

## **9.30 TimeSynchronization Service Execution Tests**

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

~~BACnet Reference Clause: 16.7.~~

### **9.30.1 Positive TimeSynchronization Service Execution Tests**

[Change **Clause 9.30.1.1**, p. 435]

[Reviewer Note: UTC\_Offset and Daylight\_Savings\_Status are optional properties that are only required for the UTCTimeSynchronization service.]

#### **9.30.1.1 TimeSynchronization Local Broadcast**

Purpose: To verify that the IUT resets its local time and date in response to a local broadcast TimeSynchronization service request.

*Notes to Tester: Select date and time such that either one or both of them is different from initial date and time.*

Test Steps:

1. *READ InitialDate* = *Local\_Date*
2. *READ InitialTime* = *Local\_Time*
3. *TRANSMIT*  
*DA* = *LOCAL BROADCAST*,  
*SA* = *TD*,  
*BACnet-Unconfirmed-Request-PDU*,  
*'Service Choice'* = *TimeSynchronization-Request*,  
*date* = *NewDate: combined with NewTime is different than the InitialDate/InitialTime pair*  
*time* = *NewTime; combined with NewDate is different than the InitialDate/InitialTime pair*
4. *VERIFY Local\_Date* = *NewDate*
5. *VERIFY Local\_Time* ~ = *NewTime*
1. ~~*TRANSMIT ReadProperty Request*,~~  
~~*'Object Identifier'* = *(the IUT's Device object)*,~~  
~~*'Property Identifier'* = *Local\_Date*~~
2. ~~*RECEIVE ReadProperty ACK*,~~  
~~*'Object Identifier'* = *(the IUT's Device object)*,~~  
~~*'Property Identifier'* = *Local\_Date*,~~  
~~*'Property Value'* = *(any valid date referred to as "InitialDate" below)*~~
3. ~~*TRANSMIT ReadProperty Request*,~~  
~~*'Object Identifier'* = *(the IUT's Device object)*,~~  
~~*'Property Identifier'* = *Local\_Time*~~
4. ~~*RECEIVE ReadProperty ACK*,~~  
~~*'Object Identifier'* = *(the IUT's Device object)*,~~  
~~*'Property Identifier'* = *Local\_Time*,~~  
~~*'Property Value'* = *(any valid time referred to as "InitialTime" below)*~~
5. ~~*TRANSMIT ReadProperty Request*,~~  
~~*'Object Identifier'* = *(the IUT's Device object)*,~~  
~~*'Property Identifier'* = *UTC\_Offset*~~
6. ~~*RECEIVE ReadProperty ACK*,~~  
~~*'Object Identifier'* = *(the IUT's Device object)*,~~  
~~*'Property Identifier'* = *UTC\_Offset*,~~  
~~*'Property Value'* = *(any valid offset referred to as "InitialUTC\_Offset" below)*~~
7. ~~*TRANSMIT ReadProperty Request*,~~  
~~*'Object Identifier'* = *(the IUT's Device object)*,~~  
~~*'Property Identifier'* = *Daylight\_Savings\_Status*~~
8. ~~*RECEIVE ReadProperty ACK*,~~  
~~*'Object Identifier'* = *(the IUT's Device object)*,~~  
~~*'Property Identifier'* = *Daylight\_Savings\_Status*,~~  
~~*'Property Value'* = *(any valid status referred to as "InitialDaylight\_Savings\_Status" below)*~~
5. ~~*TRANSMIT*~~  
~~*DA* = *LOCAL BROADCAST*,~~  
~~*SA* = *TD*,~~  
~~*BACnet Unconfirmed Request PDU*,~~  
~~*'Service Choice'* = *TimeSynchronization Request*,~~  
~~*date* = *(any date other than InitialDate)*,~~  
~~*time* = *(any time that does not correspond to InitialTime)*~~
6. ~~*TRANSMIT ReadProperty Request*,~~  
~~*'Object Identifier'* = *(the IUT's Device object)*,~~  
~~*'Property Identifier'* = *Local\_Date*~~
7. ~~*RECEIVE ReadProperty ACK*,~~  
~~*'Object Identifier'* = *(the IUT's Device object)*,~~  
~~*'Property Identifier'* = *Local\_Date*,~~  
~~*'Property Value'* = *(the date specified in step 5)*~~
8. ~~*TRANSMIT ReadProperty Request*,~~  
~~*'Object Identifier'* = *(the IUT's Device object)*,~~  
~~*'Property Identifier'* = *Local\_Time*~~
9. ~~*RECEIVE ReadProperty ACK*,~~

~~'Object Identifier' = (the IUT's Device object),~~  
~~'Property Identifier' = Local\_Time,~~  
~~'Property Value' = (the time specified in step 5)~~  
 10. TRANSMIT ReadProperty Request,  
~~'Object Identifier' = (the IUT's Device object),~~  
~~'Property Identifier' = Local\_Date~~  
 11. RECEIVE ReadProperty ACK,  
~~'Object Identifier' = (the IUT's Device object),~~  
~~'Property Identifier' = Local\_Date,~~  
~~'Property Value' = (the date specified in step 9)~~  
 12. TRANSMIT ReadProperty Request,  
~~'Object Identifier' = (the IUT's Device object),~~  
~~'Property Identifier' = Local\_Time~~  
 13. RECEIVE ReadProperty ACK,  
~~'Object Identifier' = (the IUT's Device object),~~  
~~'Property Identifier' = Local\_Time,~~  
~~'Property Value' = (the time specified in step 9)~~

~~Notes to Tester: The time value returned by the IUT in step 9 shall agree with the time specified in step 5 within the resolution for time specified in the EPICS. If the time returned by the IUT indicates that a small amount of time has passed (< 1 second) since the TimeSynchronization request was received the result shall be considered to be a pass. If the time indicates that the day of week is unspecified but all other fields are correct the result shall be considered to be a pass.~~

[Change **Clause 9.30.1.2**, p. 436]

[Reviewer Note: UTC\_Offset and Daylight\_Savings\_Status are optional properties that are only required for the UTCTimeSynchronization service.]

### 9.30.1.2 TimeSynchronization Directed to the IUT

Purpose: To verify that the IUT resets its local time and date in response to a TimeSynchronization service request directed to the IUT's MAC address.

*Notes to Tester: The passing results are identical to 9.30.1.1.*

Test Steps: This test is identical to 9.30.1.1 except that the TimeSynchronization-Request in ~~step 9~~ *step 5* shall be transmitted using the IUT's MAC address as the destination.

~~Notes to Tester: The passing results are identical to 9.30.1.1.~~

[Change **Clause 9.31**, p. 437]

## 9.31 UTCTimeSynchronization Service Execution Tests

~~BACnet Reference Clause: 16.8.~~

### 9.31.1 Positive UTCTimeSynchronization Service Execution Tests

[Change **Clause 9.31.1.1**, p. 437]

[Reviewer Note: UTC\_Offset and Daylight\_Savings\_Status are needed here, as these optional properties are required for the UTCTimeSynchronization service.]

### 9.31.1.1 Local Broadcast

**Purpose:** To verify that the IUT resets its local time and date in response to a local broadcast UTCTimeSynchronization service request.

*Notes to Tester:* Select date and time such that either one or both of them is different from initial date and time. The IUT may update the Daylight\_Savings\_Status during the execution of the UTCTimeSynchronization request.

*Test Steps:*

~~Test Steps: The test steps are identical to the steps in 9.30.1.1 except that in step 9 the UTCTimeSynchronization request is used and the date and time conveyed represent UTC.~~

~~Passing Results: The passing results are identical to 9.30.1.1 except that the date in step 9 shall be corrected for InitialUTC\_Offset, and the time in step 13 shall be corrected for both Initial\_UTC\_Offset and Daylight\_Savings\_Status (as defined in BACnet 16.7.2).~~

1. READ InitialDate = Local\_Date
2. READ InitialTime = Local\_Time
3. TRANSMIT
  - DA = LOCAL BROADCAST,
  - SA = TD,
  - BACnet-Unconfirmed-Request-PDU,
  - 'Service Choice' = UTCTimeSynchronization-Request,
  - date = NewUtcDate: combined with NewUtcTime and converted to local time is different than the InitialDate/InitialTime pair
  - time = NewUtcTime: combined with NewUtcDate and converted to local time is different than the InitialDate/InitialTime pair
4. VERIFY Local\_Date = (NewUtcDate converted to local date/time using UTC\_Offset and Daylight\_Saving\_Status)
5. VERIFY Local\_Time ~ = (NewUtcTime converted to local date/time using UTC\_Offset and Daylight\_Saving\_Status)

[Change **Clause 9.32**, p. 437]

### 9.32 Who-Has Service Execution Tests

The purpose of this test group is to verify the correct execution of the Who-Has service request.

~~Dependencies: None.~~

~~BACnet Reference Clause: 16.9.~~

#### 9.32.1 Execution of Who-Has Service Requests Originating from the Local Network

[Change **Clause 9.32.1.1**, p. 437]

[Reviewer Note: Modified test to remove dependency on EPICS values. Added the allowance for Unicast I-Have.]

##### 9.32.1.1 Object ID Version with No Device Range

**Purpose:** To verify that the IUT can correctly respond to a local broadcast Who-Has service request that utilizes the object identifier form and does not restrict device ranges.

*Configuration Requirements:* Choose any object (Object1) that exists within the IUT.

*Test Steps:*

1. *READ VI = (Object1), Object\_Name*
2. TRANSMIT
  - DA = LOCAL BROADCAST,
  - SA = TD,
  - Who-Has-Request,
  - 'Object Identifier' = *Object1*(~~any object identifier specified in the EPICS~~)
3. ~~WAIT Internal Processing Fail Time BEFORE Unconfirmed Response Fail Time~~
4. RECEIVE
  - DA = LOCAL BROADCAST | GLOBAL BROADCAST | TD,
  - ~~SA = IUT,~~
  - I-Have-Request,
  - 'Device Identifier' = (the IUT's Device object),
  - 'Object Identifier' = *Object1*(~~the object identifier specified in step 1~~),
  - 'Object Name' = *VI*(~~the object name specified in the EPICS for this object~~)

[Change **Clause 9.32.1.2**, p. 438]

[Reviewer Note: Modified test to remove dependency on EPICS values. Added the allowance for Unicast I-Have.]

### 9.32.1.2 Object Name Version with no Device Range

Purpose: To verify that the IUT can correctly respond to a local broadcast Who-Has service request that utilizes the object name form and does not restrict device ranges.

*Configuration Requirements: Choose any object (Object1) that exists within the IUT.*

Test Steps:

1. *READ VI =(Object1), Object\_Name*
2. TRANSMIT
  - DA = LOCAL BROADCAST,
  - SA = TD,
  - Who-Has-Request,
  - 'Object Name' = *VI*(~~any object name specified in the EPICS~~)
3. ~~WAIT Internal Processing Fail Time BEFORE Unconfirmed Response Fail Time~~
4. RECEIVE
  - DA = LOCAL BROADCAST | GLOBAL BROADCAST | TD,
  - ~~SA = IUT,~~
  - I-Have-Request,
  - 'Device Identifier' = (the IUT's Device object),
  - 'Object Identifier' = *Object1*(~~the object identifier specified in the EPICS for this object~~),
  - 'Object Name' = *VI*(~~the object name specified in step 1~~)

[Change **Clause 9.32.1.3**, p. 438]

[Reviewer Note: Modified test to remove dependency on EPICS values. Added the allowance for Unicast I-Have.]

### 9.32.1.3 Object ID Version with IUT Inside of the Device Range

Purpose: To verify that the IUT can correctly respond to a local broadcast Who-Has service request that utilizes the object identifier form and specifies a device range restriction that includes the IUT.

*Configuration Requirements: Choose any object (Object1) that exists within the IUT.*

Test Steps:

1. *READ VI =(Object1), Object\_Name*

2. TRANSMIT
  - DA = LOCAL BROADCAST,
  - SA = TD,
  - Who-Has-Request,
  - 'Device Instance Low Limit' = (any value L:  $0 \leq L <$  the Device object instance number of the IUT),
  - 'Device Instance High Limit' = (any value H:  $H >$  the Device object instance number of the IUT),
  - 'Object Identifier' = *Object1* (any object identifier specified in the EPICS),
3. ~~WAIT Internal Processing Fail Time BEFORE Unconfirmed Response Fail Time~~
4. RECEIVE
  - DA = LOCAL BROADCAST | GLOBAL BROADCAST | TD,
  - ~~SA = IUT,~~
  - I-Have-Request,
  - 'Device Identifier' = (the IUT's Device object),
  - 'Object Identifier' = *Object1* (the object identifier specified in step 1),
  - 'Object Name' = *VI* (the object name specified in the EPICS for this object)

[Change **Clause 9.32.1.4**, p. 439]

[Reviewer Note: Modified test to remove dependency on EPICS values.]

#### 9.32.1.4 Object ID Version with IUT Outside of the Device Range

Purpose: To verify that the IUT ignores a local broadcast Who-Has service request that utilizes the object identifier form and specifies a device range restriction that does not include the IUT.

*Configuration Requirements: Choose any object (Object1) that exists within the IUT.*

Test Steps:

1. TRANSMIT
  - DA = LOCAL BROADCAST,
  - SA = TD,
  - Who-Has-Request,
  - 'Device Instance Low Limit' = (any value  $> 0$ : the Device object instance number does not fall in the range between Device Instance Low Limit and Device Instance High Limit),
  - 'Device Instance High Limit' = (any value  $>$  Device Instance Low Limit: the Device object instance number does not fall in the range between Device Instance Low Limit and Device Instance High Limit),
  - 'Object Identifier' = *Object1* (any object identifier specified in the EPICS)
2. ~~WAIT Internal Processing Fail Time~~ *Unconfirmed Response Fail Time*
3. CHECK (verify that the IUT does not respond)

[Change **Clause 9.32.1.4**, p. 439]

[Reviewer Note: Modified test to remove dependency on EPICS values. Added the allowance for Unicast I-Have.]

#### 9.32.1.5 Object Name Version with IUT Inside of the Device Range

Purpose: To verify that the IUT can correctly respond to a local broadcast Who-Has service request that utilizes the object name form and specifies a device range restriction that includes the IUT.

*Configuration Requirements: Choose any object (Object1) that exists within the IUT.*

Test Steps:

1. READ *VI* = (*Object1*), *Object\_Name*
2. TRANSMIT
  - DA = LOCAL BROADCAST,

SA = TD,  
 Who-Has-Request,  
 'Device Instance Low Limit' = (any value L:  $0 \leq L < \text{the Device object instance number of the IUT}$ ),  
 'Device Instance High Limit' = (any value H:  $H > \text{the Device object instance number of the IUT}$ ),  
 'Object Name' = *VI*(~~any object name specified in the EPICS~~)

3. ~~WAIT Internal Processing Fail Time BEFORE Unconfirmed Response Fail Time~~

4. RECEIVE  
 DA = LOCAL BROADCAST | GLOBAL BROADCAST | TD,  
 ——— SA = IUT,  
 I-Have-Request,  
 'Device Identifier' = (the IUT's Device object),  
 'Object Identifier' = *Object1*(~~the object identifier specified in the EPICS for this object~~),  
 'Object Name' = *VI*(~~the object name specified in step 1~~)

[Change **Clause 9.32.1.7**, p. 440]

[Reviewer Note: Modified test to remove dependency on EPICS values. Added the allowance for Unicast I-Have.]

### 9.32.1.7 Object ID Version with IUT Device Instance Equal to the High Limit of the Device Range

Purpose: To verify that the IUT correctly recognizes the high limit of the specified device range for Who-Has service requests that utilize the object identifier form.

*Configuration Requirements: Choose any object (Object1) that exists within the IUT.*

Test Steps:

1. READ *VI* = (*Object1*), *Object\_Name*

2. TRANSMIT  
 DA = LOCAL BROADCAST,  
 SA = TD,  
 Who-Has-Request,  
 'Device Instance Low Limit' = (any value L:  $0 \leq L < \text{the Device object instance number of the IUT}$ ),  
 'Device Instance High Limit' = (The Device object instance number of the IUT),  
 'Object Identifier' = *Object1*(~~any object identifier specified in the EPICS~~)

3. ~~WAIT Internal Processing Fail Time BEFORE Unconfirmed Response Fail Time~~

4. RECEIVE  
 DA = LOCAL BROADCAST | GLOBAL BROADCAST | TD,  
 ——— SA = IUT,  
 I-Have-Request,  
 'Device Identifier' = (the IUT's Device object),  
 'Object Identifier' = *Object1*(~~the object identifier specified in step 1~~),  
 'Object Name' = *VI*(~~the object name specified in the EPICS for this object~~)

[Change **Clause 9.32.1.8**, p. 440]

[Reviewer Note: Modified test to remove dependency on EPICS values. Added the allowance for Unicast I-Have.]

### 9.32.1.8 Object ID Version with IUT Device Instance Equal to the Low Limit of the Device Range

Purpose: To verify that the IUT correctly recognizes the low limit of the specified device range for Who-Has service requests that utilize the object identifier form.

*Configuration Requirements: Choose any object (Object1) that exists within the IUT.*

Test Steps:

1. READ *VI* = (*Object1*), *Object\_Name*



2. TRANSMIT
  - DA = LOCAL BROADCAST,
  - SA = TD,
  - Who-Has-Request,
  - 'Device Instance Low Limit' = (The Device object instance number of the IUT),
  - 'Device Instance High Limit' = (any value H:  $H >$  the Device object instance number of the IUT),
  - 'Object Identifier' = *Object1*(~~any object identifier specified in the EPICS~~)
3. ~~WAIT Internal Processing Fail Time BEFORE Unconfirmed Response Fail Time~~
4. RECEIVE
  - DA = LOCAL BROADCAST | GLOBAL BROADCAST | TD,
  - ~~SA = IUT,~~
  - I-Have-Request,
  - 'Device Identifier' = (the IUT's Device object),
  - 'Object Identifier' = *Object1*(~~the object identifier specified in step 1~~),
  - 'Object Name' = *V1*(~~the object name specified in the EPICS for this object~~)

[Change **Clause 9.32.1.9**, p. 440]

[Reviewer Note: Modified test to remove dependency on EPICS values. Added the allowance for Unicast I-Have.]

### 9.32.1.9 Object Name Version with IUT Device Instance Equal to the High Limit of the Device Range

Purpose: To verify that the IUT correctly recognizes the high limit of the specified device range for Who-Has service requests that utilize the object name form.

*Configuration Requirements: Choose any object (Object1) that exists within the IUT.*

Test Steps:

1. READ *V1* = (*Object1*), *Object\_Name*
2. TRANSMIT
  - Who-Has-Request,
  - 'Device Instance Low Limit' = (any value L:  $0 \leq L <$  the Device object instance number of the IUT),
  - 'Device Instance High Limit' = (The Device object instance number of the IUT),
  - 'Object Name' = *V1*(~~any object name specified in the EPICS~~)
3. ~~WAIT Internal Processing Fail Time BEFORE Unconfirmed Response Fail Time~~
4. RECEIVE
  - DA = LOCAL BROADCAST | GLOBAL BROADCAST | TD,
  - ~~SA = IUT,~~
  - I-Have-Request,
  - 'Device Identifier' = (the IUT's Device object),
  - 'Object Identifier' = *Object1*(~~the object identifier specified in the EPICS for this object~~),
  - 'Object Name' = *V1*(~~the object name specified in step 1~~)

[Change **Clause 9.32.1.10**, p. 441]

[Reviewer Note: Modified test to remove dependency on EPICS values. Added the allowance for Unicast I-Have.]

### 9.32.1.10 Object Name Version with IUT Device Instance Equal to the Low Limit of the Device Range

Purpose: To verify that the IUT correctly recognizes the low limit of the specified device range for Who-Has service requests that utilize the object name form.

*Configuration Requirements: Choose any object (Object1) that exists within the IUT.*

Test Steps:

1. READ *V1* = (*Object1*), *Object\_Name*

2. TRANSMIT
  - DA = LOCAL BROADCAST,
  - SA = TD,
  - Who-Has-Request,
  - 'Device Instance Low Limit' = (The Device object instance number of the IUT),
  - 'Device Instance High Limit' = (any value H: H > the Device object instance number of the IUT),
  - 'Object Name' = *VI*(~~any object name specified in the EPICS~~)
3. ~~WAIT Internal Processing Fail Time BEFORE Unconfirmed Response Fail Time~~
4. RECEIVE
  - DA = LOCAL BROADCAST | GLOBAL BROADCAST | TD,
  - ~~SA = IUT,~~
  - I-Have-Request,
  - 'Device Identifier' = (the IUT's Device object),
  - 'Object Identifier' = *Object1*(~~the object identifier specified in step 1~~),
  - 'Object Name' = *VI*(~~the object name specified in the EPICS for this object~~)

[Change **Clause 9.32.1.11**, p. 441]

[Reviewer Note: Modified test to remove dependency on EPICS values. Added the allowance for Unicast I-Have.]

### 9.32.1.11 Object Name Version, Directed to a Specific MAC Address

Purpose: To verify that the IUT responds with a broadcast I-Have service request even if the Who-Has service requests was not transmitted with a broadcast address.

*Configuration Requirements: Choose any object (Object1) that exists within the IUT.*

Test Steps:

1. READ *VI* = (*Object1*), *Object\_Name*
2. TRANSMIT Who-Has-Request,
  - 'Object Name' = *VI*(~~any object name specified in the EPICS~~),
3. ~~WAIT Internal Processing Fail Time BEFORE Unconfirmed Response Fail Time~~
4. RECEIVE
  - DA = LOCAL BROADCAST | GLOBAL BROADCAST | TD,
  - ~~SA = IUT,~~
  - I-Have-Request,
  - 'Device Identifier' = (the IUT's Device object),
  - 'Object Identifier' = *Object1*(~~the object identifier specified in the EPICS for this object~~),
  - 'Object Name' = *VI*(~~the object name specified in step 1~~)

[Change **Clause 9.32.1.12**, p. 442]

[Reviewer Note: Addendum 135-2012ar-5 (Protocol Revision 15) allows the IUT to send a unicast response.]

### 9.32.1.12 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~~ *Unconfirmed Response 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. *BEFORE Unconfirmed Response Fail Time*  
    RECEIVE ~~DESTINATION~~ *DESTINATION* = LOCAL BROADCAST | GLOBAL BROADCAST | TD,  
    I-Have-Request,  
    'Device Identifier' = (the IUT's Device object),  
    'Object Identifier' = O1,  
    'Object Name' = V2

[Change **Clause 9.32.1.13**, p. 442]

[Reviewer Note: Addendum 135-2012ar-5 (Protocol Revision 15) allows the IUT to send a unicast response.]

### 9.32.1.13 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~~ **Unconfirmed Response 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. ~~BEFORE~~ **Unconfirmed Response Fail Time**  
RECEIVE ~~DESTINATION~~ **DA = LOCAL BROADCAST | GLOBAL BROADCAST | TD,**  
I-Have-Request,  
'Device Identifier' = (the IUT's Device object),  
'Object Identifier' = O2  
'Object Name' = *VI*(~~the object name specified in the EPICS for this object~~)

### 9.32.2 Execution of Who-Has Service Requests Originating from a Remote Network

[Change **Clause 9.32.2.1**, p. 443]

[Reviewer Note: Modified test to remove dependency on EPICS values. Added the allowance for Unicast I-Have.]

#### 9.32.2.1 Object ID Version, Global Broadcast from a Remote Network

Purpose: To verify the ability of the IUT to recognize the origin of a globally broadcast Who-Has service request and to respond such that the device originating the request receives the response.

*Configuration Requirements: Choose any object (Object1) that exists within the IUT.*

Test Steps:

1. *READ VI =(Object1), Object\_Name*
2. TRANSMIT  
~~DESTINATION = LOCAL BROADCAST,~~  
~~SA = TD,~~  
DNET = GLOBAL BROADCAST,  
SNET = (X: any remote network number),  
SADR = (Y: any MAC address valid for the specified network),  
Who-Has-Request,  
'Object Identifier' = *Object1*(~~any object identifier specified in the EPICS~~)
3. ~~WAIT Internal Processing Fail Time~~ **BEFORE Unconfirmed Response Fail Time**
4. RECEIVE  
DESTINATION = GLOBAL BROADCAST | REMOTE BROADCAST (to the network *X*~~specified in step 1~~) |  
*TD (DNET = X, DADR = Y),*  
I-Have-Request,  
'Device Identifier' = (the IUT's Device object),  
'Object Identifier' = *Object1*(~~the object identifier specified in step 1~~),  
'Object Name' = *VI*(~~the object name specified in the EPICS for this object~~)

[Change **Clause 9.32.2.2**, p. 443]

[Reviewer Note: Modified test to remove dependency on EPICS values. Added the allowance for Unicast I-Have.]

### 9.32.2.2 Object ID Version, Remote Broadcast

Purpose: To verify the ability of the IUT to recognize the origin of a remotely broadcast Who-Has service request and to respond such that the device originating the request receives the response.

*Configuration Requirements: Choose any object (Object1) that exists within the IUT.*

Test Steps:

1. READ V1=(Object1), Object\_Name
2. TRANSMIT  
~~DESTINATION = LOCAL BROADCAST,~~  
~~SA = TD,~~  
 SNET = (any remote network number),  
 SADR = (any MAC address valid for the specified network),  
 Who-Has-Request,  
 'Object Identifier' = Object1(~~any object identifier specified in the EPICS~~)
3. ~~WAIT Internal Processing Fail Time BEFORE Unconfirmed Response Fail Time~~
4. RECEIVE  
 DESTINATION = GLOBAL BROADCAST | REMOTE BROADCAST (to the network X~~specified in step 1~~) |  
 TD (DNET = X, DADR = Y),  
 I-Have-Request,  
 'Device Identifier' = (the IUT's Device object),  
 'Object Identifier' = Object1(~~the object identifier specified in step 1~~),  
 'Object Name' = V1(~~the object name specified in the EPICS for this object~~)

[Add new **Clause 9.32.2.X3**, p. 444]

### 9.32.2.X3 Who-Has for Non-existent Object\_Name

Purpose: Verifies correct responses to Who-Has service requests by 'Object Name' when the object does not exist in the IUT.

Test Concept: The test verifies the correct non-response to Who-Has service request with 'Object Name' when that named object does not exist in the IUT.

Configuration Requirements: Choose any character string value V1, which is not the Object\_Name of any object in the IUT. The IUT shall be placed in a state where it is not producing I-Have spontaneously.

Test Steps:

1. TRANSMIT Who-Has-Request,  
 'Object Name' = V1
2. WAIT Unconfirmed Response Fail Time
3. CHECK (the IUT does not respond with an I-Have request with 'Object Name' containing V1)

[Add new **Clause 9.32.2.X5**, p. 444]

### 9.32.2.X5 Who-Has for Non-existent Object\_Identifier

Purpose: Verifies correct responses to Who-Has service requests when the object does not exist in the IUT.

Test Concept: The test verifies the correct non-response to Who-Has request with that 'Object Identifier' parameter for an object which does not exist.

Configuration Requirements: Choose any standard object (Object1) that does not exist within the IUT, i.e., any unsupported Object Type or any supported Object Type for which the instance does not exist. The IUT shall be placed in a state where it is not producing I-Have spontaneously.

#### Test Steps:

1. TRANSMIT ReadProperty-Request,  
    'Object Identifier' = Object1,  
    'Property Identifier' = Object\_Identifier
2. RECEIVE BACnet-Error-PDU,  
    'Error Class' = OBJECT,  
    'Error Code' = UNKNOWN\_OBJECT
3. TRANSMIT Who-Has-Request,  
    'Object Identifier' = Object1
4. **WAIT Unconfirmed Response Fail Time**
5. CHECK (the IUT does not respond with an I-Have request with 'Object Identifier' containing Object1)

### 9.33 Who-Is Service Execution Tests

[Change **Clause 9.33.1**, p. 444]

#### 9.33.1 Execution of Who-Is Service Requests Originating from the Local Network

The purpose of this test group is to verify the correct execution of the Who-Is request service procedure for messages originating from the local network.

~~Dependencies: None.~~

~~BACnet Reference Clause: 16.10.~~

[Change **Clause 9.33.1.3**, p. 445]

[Reviewer Note: The allowed device instance range is from 0 - 4194303 and is specified in section 16.9.1.1.1 and 16.10.1.1.1. The corresponding test incorrectly set the low limit greater than 0 when it should have been greater than or equal to 0.]

#### 9.33.1.3 Local Broadcast, Specific Device Inquiry with IUT Outside of the Device Range

Purpose: To verify that the IUT ignores Who-Is requests when it is excluded from the specified device range.

#### Test Steps:

1. TRANSMIT  
    DESTINATION = LOCAL BROADCAST,  
    Who-Is-Request,  
    'Device Instance Range Low Limit' = ~~(any value)~~ *any value*  $\geq 0$  such that the Device object instance number does not fall in  
        the range between Device Instance Low Limit and Device Instance High Limit),  
    'Device Instance Range High Limit' = ~~(any value)~~ *any value*  $\geq$  Device Instance Low Limit such that the Device  
    object  
        instance number does not fall in the range between Device Instance Low Limit  
        and Device Instance High Limit)
2. **WAIT Internal Processing Fail Time**
3. CHECK (verify that the IUT does not respond)

#### 9.33.2 Execution of Who-Is Service Requests Originating from a Remote Network

[Change **Clause 9.33.2.3**, p. 447]

[Reviewer Note: Updated the purpose and changed the expected destination on the received packet.]

### 9.33.2.3 General Inquiry, Directed to a Remote Device

Purpose: To verify ~~that the IUT responds with an I-Am service that is of the form global broadcast, remote broadcast or unicast~~ *the ability of the IUT to recognize the origin of a Who-Is service request, directed to the IUT, and respond such that the device originating the request receives the response.*

Test Steps:

1. TRANSMIT
  - DESTINATION = IUT,
  - SNET = (any remote network number),
  - SADR = (any MAC address valid for the specified network),
  - Who-Is-Request
2. WAIT **Unconfirmed Response Fail Time**
3. RECEIVE
  - DESTINATION = GLOBAL BROADCAST | ~~LOCAL BROADCAST~~ *REMOTE BROADCAST (to the network specified by SNET in step 1) | TD*
  - I-Am-Request,
  - 'I Am Device Identifier' = (the IUT's Device object),
  - 'Max APDU Length Accepted' = (the value specified in the EPICS),
  - 'Segmentation Supported' = (the value specified in the EPICS),
  - 'Vendor Identifier' = (the identifier registered for this vendor)

[Add new **Clause 9.X40**, p. 457]

[Reviewer Note: Added new test to support AuditLogQuery testing.]

## 9.X40 AuditLogQuery Service Execution Tests

### 9.X40.1 AuditLogQuery Service Positive Tests

#### 9.X40.1.1 - AuditLogQuery By Target Test

Purpose: Verify that an Audit Log correctly returns notifications filtered by a specific target.

Test Concept: An Audit Log, O1, containing a sequence of notifications related to a set of audit targets is queried about a specific target, T1. The query is repeated for each standard form of target-based query, and the result is compared against the expected set of returned notifications.

Configuration Requirements: The Audit Log, O1, contains a set of audit notifications which contains 0 or more notifications about audit target T1.

Test Steps:

1. REPEAT Q = (each query in (
  - By Target Device Identifier,
  - By Target Device Identifier & Target Device Address,
  - By Target Device Identifier & Target Object Identifier,
  - By Target Device Identifier & Target Property Identifier,
  - By Target Device Identifier & Target Array Index,
  - By Target Device Identifier & Target Priority,

```

    By Target Device Identifier & Operations,
    By Target Device Identifier & Result Filter (failed),
    By Target Device Identifier & Result Filter (success),
    By Target Device Identifier & Target Object Identifier & Target Property Identifier
  ) {
TRANSMIT AuditLogQuery-Request,
  'Audit Log' = O1,
  'Query Parameters' = Q,
  'Requested Count' = Total_Record_Count
RECEIVE AuditLogQuery-ACK,
  'Audit Log' = O1,
  Records' = (RSEQ: a set of records),
WHILE (the length of RSEQ is not the number of expected records) {
  TRANSMIT AuditLogQuery-Request,
    'Audit Log' = O1,
    'Query Parameters' = Q,
    'Start At Sequence Number' = (the 'Sequence Number' from the last entry in RSEQ)
    'Requested Count' = Total_Record_Count
  RECEIVE AuditLogQuery-ACK,
    'Audit Log' = O1,
    Records' = (NXTSEQ: a set of records),
  IF (the length of NXTSEQ is 0) THEN
    ERROR "expected more records from the Audit Log"
  RSEQ = (RSEQ record with NXTSEQ records appended)
}
CHECK(that the records in RSEQ is the set expected and are returned in sequence number order)
}

```

### 9.X40.1.2 - AuditLogQuery By Source Test

Purpose: Verify that an Audit Log correctly returns notifications filtered by a specific source.

Test Concept: An Audit Log, O1, containing a sequence of notifications related to a set of audit sources is queried about a specific source, S1. The query is repeated for each standard form of source-based query, and the result is compared against the expected set of returned notifications.

Configuration Requirements: The Audit Log, O1, contains a set of audit notifications which contains 0 or more notifications about audit source S1.

Test Steps:

```

1. REPEAT Q = (each query in (
  By Source Device Identifier,
  By Source Device Identifier & Source Device Address,
  By Source Device Identifier & Source Object Identifier,
  By Source Device Identifier & Operations,
  By Source Device Identifier & Result Filter (failed),
  By Source Device Identifier & Result Filter (success),
) {
TRANSMIT AuditLogQuery-Request,
  'Audit Log' = O1,
  'Query Parameters' = Q,
  'Requested Count' = Total_Record_Count
RECEIVE AuditLogQuery-ACK,
  'Audit Log' = O1,
  Records' = (RSEQ: a set of records),
WHILE (the length of RSEQ is not the number of expected records) {
  TRANSMIT AuditLogQuery-Request,
    'Audit Log' = O1,
    'Query Parameters' = Q,

```



```

    'Start At Sequence Number' = (the 'Sequence Number' from the last entry in RSEQ)
    'Requested Count' = Total_Record_Count
RECEIVE AuditLogQuery-ACK,
    'Audit Log' = O1,
    'Records' = (NXTSEQ: a set of records),
IF (the length of NXTSEQ is 0) THEN
    ERROR "expected more records from the Audit Log"
RSEQ = (RSEQ record with NXTSEQ records appended)
}
CHECK(that the records in RSEQ is the set expected and are returned in sequence number order)
}

```

## 9.X40.2 - AuditLogQuery Negative Tests

### 9.X40.2.1 - Attempting to Query a Non-existent Audit Log

Purpose: Verify that the correct error is returned when the queried log does not exist.

Test Concept: Send an AuditLogQuery request to the IUT for an AuditLog object which does not exist. Verify that the IUT returns an Error Class of OBJECT and an error code of UNKNOWN\_OBJECT.

Test Steps:

1. TRANSMIT AuditLogQuery-Request,
  - 'Audit Log' = (an audit log not in the IUT),
  - 'Query Parameters' = (any valid value),
  - 'Requested Count' = (any valid value)
2. RECEIVE BACnet-Error PDU,
  - 'Error Class' = OBJECT,
  - 'Error Code' = UNKNOWN\_OBJECT

[Add new **Clause 9.X41**, p. 457]

[Reviewer Note: Added new test to support WriteGroup testing.]

## 9.X41 WriteGroup Tests

### 9.X41.1.X1 Channel and Group Number Test

Purpose: To verify that the Channel object executes a WriteGroup service request only when containing a specified channel number and Group Number by a request, and the Channel object ignores a request otherwise. If a Group Number is 0, the Channel object ignores a service even when its Control\_Groups property is set to 0.

Test Concept: The Channel Object, O1, will be assigned a specific value to its channel number and Group Number. When a device containing O1 receives a WriteGroup service, O1 executes the request and propagates a specified value to its destination only if A) the O1's channel number is the same as specified number by the request and B) the O1's Control\_Groups contains the specified Group Number, except for a case when a Group Number was 0.

Configuration Requirements: Configure entry X of the Channel object's List\_Of\_Object\_Property\_References to refer to a commandable property of an object on either a local or remote device. For a commandable property, all prioritized commands have to be relinquished, and any minimum on/off time has to be accounted for prior to the test. An initial value of a commandable property must be the same as RD, which is its Relinquish\_Default value. The value to be propagated must be a valid value that does not require coercion.

## Test Steps:

- Obtain the data which will be used for the Channel Write Fail Time later in the steps
1. READ LEN = List\_Of\_Object\_Property\_References, ARRAY INDEX = 0
- Obtain the Channel Object's target object reference
2. READ PR = List\_Of\_Object\_Property\_References, ARRAY INDEX = X
- Set arbitrary numbers for Channel\_Number and Control\_Group to O1
3. TRANSMIT WritePropertyMultiple-Request,
  - 'Object Identifier' = O1,
  - 'Property Identifier' = Channel\_Number
  - 'Property Value' = (CN: Any valid value)
  - 'Property Identifier' = Control\_Groups
  - 'Property Value' = (CG: Any length of an array containing at least 1 non-zero element)
4. RECEIVE BACnet-SimpleACK-PDU
- Send a WriteGroup with a mismatching channel number and Group Number
5. TRANSMIT WriteGroup-Request,
  - 'Group Number' = (A valid value larger than 0 and not contained by CG)
  - 'Write Priority' = (any valid value),
  - 'Change List' = (Any valid value different than CN, no overriding priority, Y: a valid value different than RD)
6. WAIT **Channel Write Fail Time** \* LEN
- Make sure that O1 did not propagate a value to its target references
7. VERIFY PR = RD
- Send a WriteGroup with a matching channel number and a mismatching group number
8. TRANSMIT WriteGroup-Request,
  - 'Group Number' = (A valid value larger than 0 and not contained by CG)
  - 'Write Priority' = (any valid value),
  - 'Change List' = (CN, no overriding priority, Y: a valid value different than RD)
9. WAIT **Channel Write Fail Time** \* LEN
- Make sure that O1 did not propagate value to its target references
10. VERIFY PR = RD
- Send a WriteGroup with a mismatching channel number and a matching group number
11. TRANSMIT WriteGroup-Request,
  - 'Group Number' = (Any non 0 values contained by CG)
  - 'Write Priority' = (any valid value),
  - 'Change List' = (Any valid value different than CN, no overriding priority, Y: a valid value different than RD)
12. WAIT **Channel Write Fail Time** \* LEN
- Make sure that O1 did not propagate value to its target references
13. VERIFY PR = RD
- Send a WriteGroup service with a matching channel number and group number
14. TRANSMIT WriteGroup-Request,
  - 'Group Number' = (Any non 0 values contained by CG)
  - 'Write Priority' = (any valid value),
  - 'Change List' = (CN, no overriding priority, Y: a valid value different than RD)
- Make sure that O1 did propagate value to its target references
15. VERIFY PR = Y
- Change Control\_Groups to 0
16. TRANSMIT WritePropertyMultiple-Request,
  - 'Object Identifier' = O1,
  - 'Property Identifier' = Control\_Groups
  - 'Property Value' = 0
17. RECEIVE BACnet-SimpleACK-PDU
- Send a WriteGroup with 0 Group number
18. TRANSMIT WriteGroup-Request,
  - 'Group Number' = 0
  - 'Write Priority' = (any valid value),
  - 'Change List' = (CN, no overriding priority, Z: a valid value different than Y)
19. WAIT **Channel Write Fail Time** \* LEN
- Make sure that O1 did not propagate value to its target references
20. VERIFY PR = Y

### 9.X41.1.X2 Write Priority and Overriding Priority Test

Purpose: To verify that the overridingPriority, if provided, specifies the priority for writing the value. Otherwise, the 'Write Priority' parameter specifies the priority for writing.

Test Concept: The Channel Object, O1, receives the WriteGroup with P1 as its 'Write Priority', and it is verified that P1 is used for writing the value. O1 then receives another WriteGroup with P1 as its 'Write Priority' and P2 as its overridingPriority, and it is verified that P2 is used for writing the value.

Configuration Requirements: Configure one of the entry for the Channel object's List\_Of\_Object\_Property\_References to refer to a commendable property of an object O2 with Priority\_Array on either a local or remote device. All prioritized commands have to be relinquished, and any minimum on/off time has to be accounted for prior to the test.

Test Steps:

-- Obtain the data which will be used for the Channel Write Fail Time later in the steps

1. READ LEN = List\_Of\_Object\_Property\_References, ARRAY INDEX = 0

-- Write to O1 using P1 as its Write Priority

2. TRANSMIT WriteGroup-Request,

'Group Number' = (one of the Control\_Group values configured in O1),

'Write Priority' = (P1: Any valid value)

'Change List' = (O1's channel number, no overriding priority, Y: any valid value)

3. WAIT Channel Write Fail Time \* LEN

-- Make sure that P1 is used for writing the value

4. VERIFY (O2), Priority\_Array = Y, ARRAY INDEX = P1

-- Write to O1 using P1 as its Write Priority, P2 as its overridingPriority

5. TRANSMIT WriteGroup-Request,

'Group Number' = (one of the Control\_Group values configured in O1),

'Write Priority' = P1

'Change List' = (O1's channel number, P2: Any valid value different than P1, Z: any valid value different than Y)

6. WAIT Channel Write Fail Time \* LEN

-- Make sure that P2 is used for writing the value

7. VERIFY (O2), Priority\_Array = Z, ARRAY INDEX = P2

-- Make sure that no update on Priority\_Array[P1]

8. VERIFY (O2), Priority\_Array = Y, ARRAY INDEX = P1

### 9.X41.1.X3 Relinquish Control Test

Purpose: To verify that if a BACnetGroupChannelValue specifies a NULL value, it serves the same function as if NULL had been used with WriteProperty.

Test Concept: The Channel Object, O1, receives the WriteGroup service to propagate a value to its destination object property reference, PR. PR is verified to have the value updated accordingly. The O1 then receives another WriteGroup service with the BACnetGroupChannelValue specifying a NULL value. PR is verified to have its Relinquish\_Default value.

Configuration Requirements: Configure entry X of the Channel object's List\_Of\_Object\_Property\_References to refer to a commendable property of an object O2 with a Relinquish\_Default set to RD on either a local or remote device. All prioritized commands have to be relinquished, and any minimum on/off time has to be accounted for prior to the test.

Test Steps:

-- Obtain the data which will be used for the Channel Write Fail Time later in the steps

1. READ LEN = List\_Of\_Object\_Property\_References, ARRAY INDEX = 0

-- Obtain the Channel Object's target object reference

2. READ PR = List\_Of\_Object\_Property\_References, ARRAY INDEX = X

-- Let the Channel Object propagate a value to its target

3. TRANSMIT WriteGroup-Request,

'Group Number' = (One of the Control\_Group values configured in O1),

- 'Write Priority' = (Any valid value)
- 'Change List' = (O1's channel number, no overriding priority, X: any valid value different than RD)
- 4. **WAIT Channel Write Fail Time \* LEN**
- 5. **VERIFY PR = X**
- Let the Channel Object relinquish control of the target
- 6. **TRANSMIT WriteGroup-Request,**
  - 'Group Number' = (One of the Control\_Group values configured in O1),
  - 'Write Priority' = (Any valid value)
  - 'Change List' = (O1's channel number, no overriding priority, NULL)
- 7. **WAIT Channel Write Fail Time \* LEN**
- 8. **VERIFY PR = RD**

#### 9.X41.1.X4 Inhibit Delay Test with WriteGroup

Purpose: In the case of WriteGroup, verify that Execution\_Delay always occurs unless the WriteGroup service parameter 'Inhibit Delay' is TRUE, and the Channel object property Allow\_Group\_Delay\_Inhibit is present and has the value TRUE.

Test Concept: Setup List\_Of\_Object\_Property\_References of the Channel Object, O1, to contain 2 valid entries, PR1 and PR2, and provide each with an execution delay (ED1 and ED2) which are larger than 0. Allow\_Group\_Delay\_Inhibit is set to TRUE. When a WriteGroup service is sent to O1 without 'Inhibit Delay' parameter, it is verified that Execution\_Delay occurs. When another WriteGroup service is sent to O1 with 'Inhibit Delay' set to False, it is verified that Execution\_Delay still occurs. Finally, when another WriteGroup service is sent with 'Inhibit Delay' set to TRUE, it is verified that Execution\_Delay does not occur.

Configuration Requirements: PR1 and PR2 shall be references to writable properties and shall be the same datatype. ED1 and ED2 shall be values which are large enough that the delay between writes is sufficient for the test. V1 and V2 shall be of the expected datatype for PR1 and PR2 so that no coercion occurs and shall be different values.

Test Steps:

- Setup the Channel object
- 1. **WRITE List\_Of\_Object\_Property\_References = (PR1, PR2)**
- 2. **WRITE Execution\_Delay = (ED1, ED2)**
- 3. **WRITE Allow\_Group\_Delay\_Inhibit = TRUE**
- 4. **READ V1 = PR1**
- 5. **READ V2 = PR2**
- Send a WriteGroup without 'Inhibit Delay' parameter
- 6. **TRANSMIT WriteGroup-Request,**
  - 'Group Number' = (one of the Control\_Group values configured in O1),
  - 'Write Priority' = (Any valid value)
  - 'Change List' = (O1's channel number, no overriding priority, X: any valid value different than V1 or V2)
- 7. **WAIT Channel Write Fail Time**
- Make sure that Execution\_Delay occurs
- 8. **VERIFY PR1 = V1**
- 9. **VERIFY PR2 = V2**
- 10. **WAIT (ED1)**
- 11. **VERIFY PR1 = X**
- 12. **VERIFY PR2 = V2**
- 13. **WAIT (ED2 – ED1)**
- 14. **VERIFY PR2 = X**
- Send a WriteGroup with 'Inhibit Delay' set to FALSE
- 15. **TRANSMIT WriteGroup-Request,**
  - 'Group Number' = (one of the Control\_Group values configured in O1),
  - 'Write Priority' = (Any valid value)
  - 'Change List' = (O1's channel number, no overriding priority, Y: any valid value different from X)
  - 'Inhibit Delay' = FALSE
- 16. **WAIT Channel Write Fail Time**

```
-- Make sure that Execution_Delay occurs
17. VERIFY PR1= X
18. VERIFY PR2 = X
19. WAIT (ED1)
20. VERIFY PR1 = Y
21. VERIFY PR2 = X
22. WAIT (ED2 – ED1)
23. VERIFY PR2 = Y
-- Send a WriteGroup with 'Inhibit Delay' set to TRUE
24. TRANSMIT WriteGroup-Request,
    'Group Number' = (one of the Control_Group values configured in O1),
    'Write Priority' = (Any valid value)
    'Change List' = (O1's channel number, no overriding priority, Z: any valid value different from Y)
    'Inhibit Delay' = TRUE
-- Make sure that Execution_Delay does NOT occur
25. WAIT Channel Write Fail Time
26. VERIFY PR1= Z
27. VERIFY PR2 = Z
```

[Add new **Clause 9.X42**, p. 457]

[Reviewer Note: Added new test to support DS-COVM-B testing.]

## 9.X42 SubscribeCOVPropertyMultiple Service Execution Tests

### 9.X42.1 Positive SubscribeCOVPropertyMultiple Service Execution Tests

#### 9.X42.1.1 Supports Non-Timestamped Notifications

Purpose: To verify that the IUT can execute a COVM Notification without providing a timestamp

Test Concept: A subscription for COVM notifications, with the Timestamped parameter set to FALSE. Verify that the IUT sends the appropriate COVM notification in response.

Test Steps:

```
1. TRANSMIT SubscribeCOVPropertyMultiple-Request
    'Subscriber Process Identifier' = (ID1: any valid process identifier),
    'Issue Confirmed Notifications' = TRUE | FALSE,
    'Lifetime' = L,
    'Max Notification Delay' = (any valid delay between 1 and 3600),
    'List of COV Subscription Specifications' = (any valid list with 'Timestamped' set to FALSE in all entries)
2. RECEIVE BACnet-SimpleACK-PDU
3. IF (the subscription was for confirmed notifications) THEN
    BEFORE Notification Fail Time
        RECEIVE ConfirmedCOVNotificationMultiple-Request,
            'Subscriber Process Identifier' = ID1,
            'Initiating Device Identifier' = TD,
            'Time Remaining' = (a value ~ L),
            -- 'Timestamp' = (absent)
            'List of COV Notifications' = (values appropriate to the properties subscribed to)
        TRANSMIT BACnet-SimpleACK-PDU
    ELSE
        WHILE (notifications have not been received for all subscribed to items)
            BEFORE Notification Fail Time
                RECEIVE UnconfirmedCOVNotificationMultiple-Request,
                    'Subscriber Process Identifier' = ID1,
                    'Initiating Device Identifier' = TD,
```

'Time Remaining' = (a value  $\sim$  L),  
 -- 'Timestamp' = (absent)  
 'List of COV Notifications' = (values appropriate to some or all of the properties subscribed to)

### 9.X42.1.2 Supports Timestamped Notifications

Purpose: To verify that the IUT can execute a COVM Notification providing a timestamp

Test Concept: A subscription for COVM notifications with the Timestamped parameter set to TRUE for at least 1 entry in the list of subscriptions, and FALSE for at least 1 entry in the list of subscriptions, is sent to the IUT for properties for which the IUT supports COVM. Verify that the IUT sends the appropriate COVM notification in response.

Notes to Tester: If the IUT only supports COVM for one property in one object, then the subscription shall be for the single property with Timestamped set to TRUE.

Test Steps:

1. TRANSMIT SubscribeCOVPropertyMultiple-Request  
     'Subscriber Process Identifier' = ID1; (any valid process identifier),  
     'Issue Confirmed Notifications' = TRUE | FALSE,  
     'Lifetime' = (L: a valid lifetime),  
     'Max Notification Delay' = (any valid delay between 1 and 3600),  
     'List of COV Subscription Specifications' = (any valid list of properties which exist in the IUT for which the IUT supports COVM with Timestamped set to TRUE for at least one, and FALSE for at least one)
2. RECEIVE BACnet-SimpleACK-PDU
3. IF (the subscription was for confirmed notifications) THEN  
     BEFORE **Notification Fail Time**  
     RECEIVE ConfirmedCOVNotificationMultiple-Request,  
     'Subscriber Process Identifier' = ID1,  
     'Initiating Device Identifier' = IUT,  
     'Time Remaining' = (a value  $\sim$  L),  
     'Timestamp' = (an appropriate timestamp)  
     'List of COV Notifications' = (values appropriate to the properties subscribed to along with 'Time of Change' values only for those for which timestamps were requested)  
     TRANSMIT BACnet-SimpleACK-PDU  
   ELSE  
     WHILE (notifications have not been received for all subscribed to items)  
     BEFORE **Notification Fail Time**  
     RECEIVE UnconfirmedCOVNotificationMultiple-Request,  
     'Subscriber Process Identifier' = ID1,  
     'Initiating Device Identifier' = IUT,  
     'Time Remaining' = (a value  $\sim$  L),  
     'Timestamp' = (an appropriate timestamp)  
     'List of COV Notifications' = (values appropriate to some or all of the properties subscribed to along with 'Time of Change' values only for those for which timestamps were requested)

### 9.X42.1.3 Confirmed Change of Value Notification From Property Value

Purpose: To verify that the IUT initiates a ConfirmedCOVMultipleNotification service request when a subscribed to property changes.

Test Concept: A COVM subscription is made which contains a subscription to property P1 in object O1. The value of P1 is changed, and it is verified that the IUT sends a COVM notification.

Test Steps:

1. TRANSMIT SubscribeCOVPropertyMultiple-Request
  - 'Subscriber Process Identifier' = (ID1: any valid process identifier),
  - 'Issue Confirmed Notifications' = TRUE,
  - 'Lifetime' = (L: any valid lifetime),
  - 'Max Notification Delay' = (any valid value),
  - 'List of COV Subscription Specifications' = (PROPS: a valid list of properties for which the IUT supports COVM including P1 in O1)
2. RECEIVE BACnet-SimpleACK-PDU
3. BEFORE **Notification Fail Time**
  - RECEIVE ConfirmedCOVNotificationMultiple-Request,
    - 'Subscriber Process Identifier' = ID1,
    - 'Initiating Device Identifier' = IUT,
    - 'Time Remaining' = (a value  $\sim$  L),
    - 'Timestamp' = (an appropriate timestamp, if subscribed to timestamped notifications, otherwise absent),
    - 'List of COV Notifications' = (values appropriate to the subscribed to properties)
4. TRANSMIT BACnet-SimpleACK-PDU
5. MAKE (a change to P1 that should cause a COVM notification)
6. BEFORE **Notification Fail Time**
  - RECEIVE ConfirmedCOVNotificationMultiple-Request,
    - 'Subscriber Process Identifier' = ID1,
    - 'Initiating Device Identifier' = IUT,
    - 'Time Remaining' = (a value greater than 0 and less than L),
    - 'Timestamp' = (an appropriate timestamp, if subscribed to timestamped notifications, otherwise absent),
    - 'List of COV Notifications' = (a list consisting of a valid value for P1 and values for any co-reported properties as described in clause 13.1)
7. TRANSMIT BACnet-SimpleACK-PDU

#### 9.X42.1.4 Unconfirmed Change of Value Notification From Property Value

Purpose: To verify that the IUT initiates an UnconfirmedCOVMultipleNotification service request when a subscribed to property changes.

Test Concept: A COVM subscription is made which contains a subscription to property P1 in object O1. The value of P1 is changed, and it is verified that the IUT send a COVM notification.

Test Steps:

1. TRANSMIT SubscribeCOVPropertyMultiple-Request
  - 'Subscriber Process Identifier' = (ID1: any valid process identifier),
  - 'Issue Confirmed Notifications' = FALSE,
  - 'Lifetime' = L,
  - 'Max Notification Delay' = (MND: any valid value)
  - 'List of COV Subscription Specifications' = (a valid list of properties for which the IUT supports COVM including P1 in O1)
2. RECEIVE BACnet-SimpleACK-PDU
3. WHILE (notifications have not been received for all subscribed to items)
  - BEFORE **Notification Fail Time**
    - RECEIVE UnconfirmedCOVNotificationMultiple-Request,
      - 'Subscriber Process Identifier' = ID1,
      - 'Initiating Device Identifier' = IUT,
      - 'Time Remaining' = (a value  $\sim$  L),
      - 'Timestamp' = (an appropriate timestamp, if subscribed to timestamped notifications, otherwise absent)
      - 'List of COV Notifications' = (values appropriate to some or all of the properties subscribed to)
4. MAKE (a change to the P1 that should cause a COVM notification)

## 5. BEFORE Notification Fail Time

```

RECEIVE UnconfirmedCOVNotificationMultiple-Request,
  'Subscriber Process Identifier' = ID1,
  'Initiating Device Identifier' = IUT,
  'Time Remaining' = (a value greater than 0 and less than the requested lifetime),
  'Timestamp' = (an appropriate timestamp, if subscribed to timestamped
                  notifications, otherwise absent),
  'List of COV Notifications' = (a list consisting of a valid value for P1 and values for any co-reported
properties as described in clause 13.1)

```

## 9.X42.1.5 Supports Subscriptions to Multiple Properties Using Multiple Requests

Purpose: To verify the server adds new subscriptions to existing COVM contexts when requested.

Test Concept: A subscription for COVM notifications is established for property P1 of object O1. A second subscription is sent using the same COVM context for property P2 in object O2. Verify that the IUT's Active\_COV\_Multiple\_Subscriptions property is correctly updated after each subscription.

Configuration Requirements: There are no active COVM subscriptions for properties in the IUT. If the IUT cannot be configured to have 2 properties which support COVM subscriptions, then this test shall be skipped.

Notes to Tester: Objects O1 and O2 can be the same object, and properties P1 and P2 can be the same property, but (O1, P1) must be different than (O2, P2).

Test Steps:

1. CHECK (the IUT's Active\_COV\_Multiple\_Subscriptions property is empty)
2. TRANSMIT SubscribeCOVPropertyMultiple-Request
 

```

'Subscriber Process Identifier' = (ID1: any valid process identifier),
'Issue Confirmed Notifications' = TRUE | FALSE,
'Lifetime' = L,
'Max Notification Delay' = (any valid notification delay),
'List of COV Subscription Specifications' = { ('Monitored Object' = O1,
   'List of COV References' = { ('Monitored Property' = P1,
   'COV Increment' = (any valid
  increment, or
  empty if P1 is not
  numeric),
   'Timestamped' = TRUE | FALSE))
   }

```
3. RECEIVE BACnet-SimpleAck-PDU
4. IF (confirmed notifications were requested) THEN
 

BEFORE Notification Fail Time

```

RECEIVE ConfirmedCOVNotificationMultiple-Request,
  'Subscriber Process Identifier' = ID1,
  'Initiating Device Identifier' = IUT,
  'Time Remaining' = (any value ~ L),
  'Timestamp' = (an appropriate timestamp)
  'List of COV Notifications' = (a list of values of length 1 indicating P1's value)
TRANSMIT BACnet-SimpleACK-PDU

```

ELSE

BEFORE Notification Fail Time

```

RECEIVE UnconfirmedCOVNotificationMultiple-Request,
  'Subscriber Process Identifier' = ID1,
  'Initiating Device Identifier' = IUT,
  'Time Remaining' = (any value ~ L),
  'Timestamp' = (an appropriate timestamp, or absent if timestamps not

```



```

                                requested)
        'List of COV Notifications' = (a list of values of length 1 indicating P1's value)
5.  VERIFY Active_COV_Multiple_Subscriptions = (a list with one entry for COVM context ID1 with 1 entry for the
subscription to P1)
6.  TRANSMIT SubscribeCOVPropertyMultiple-Request
    'Subscriber Process Identifier' = ID1,
    'Issue Confirmed Notifications' = TRUE | FALSE,
    'Lifetime' = L,
    'Max Notification Delay' = (any valid notification delay),
    'List of COV Subscription Specifications' = { ('Monitored Object' = O2,
  'List of COV References' = { (
  'Monitored Property' = P2,
  'COV Increment' = (any valid
  increment, or
  empty if P2 is not
  numeric),
  'Timestamped' = TRUE | FALSE))
  }
    )
}

7.  RECEIVE BACnet-SimpleAck-PDU
8.  IF confirmed notifications were requested THEN
    BEFORE Notification Fail Time
        RECEIVE ConfirmedCOVNotificationMultiple-Request,
        'Subscriber Process Identifier' = ID1,
        'Initiating Device Identifier' = IUT,
        'Time Remaining' = (any value ~= L),
        'Timestamp' = (an appropriate timestamp, or absent if timestamps not
                        requested)
        'List of COV Notifications' = (a list of values of length 1 indicating P2's new value)
    TRANSMIT BACnet-SimpleACK-PDU
ELSE
    WHILE (notifications have not been received for all subscribed to items)
        BEFORE Notification Fail Time
            RECEIVE UnconfirmedCOVNotificationMultiple-Request,
            'Subscriber Process Identifier' = ID1,
            'Initiating Device Identifier' = IUT,
            'Time Remaining' = (any value ~= L),
            'Timestamp' = (an appropriate timestamp, or absent if timestamps not
                           requested)
            'List of COV Notifications' = (a list of values of length 1 indicating P2's new value)
9.  VERIFY Active_COV_Multiple_Subscriptions = (a list with one entry for COVM context ID1 with 2 entries for P1 and
P2)

```

### 9.X42.1.6 Ensuring 5 Concurrent COV-Multiple Contexts With 5 COV-References Per Context

Purpose: To verify that the IUT can support 5 COV-multiple contexts with 5 COV-references each.

Test Concept: Subscriptions for COVM notifications are made using process identifiers PID1 through PID5. The required post subscription notifications are verified. Once all subscriptions are made, the Active\_COV\_Multiple\_Subscriptions is verified to contain all subscriptions.

Configuration Requirements: The IUT has no active COVM subscriptions.

Test Steps:

```

1.  REPEAT (X=PID1 to PID5) {
    TRANSMIT SubscribeCOVPropertyMultiple-Request
    'Subscriber Process Identifier' = X,
    'Issue Confirmed Notifications' = TRUE | FALSE,
    'Lifetime' = (L, any value large enough to complete the test),

```

'Max Notification Delay' = (any valid value),  
 'List of COV Subscription Specifications' = (any valid list of properties for which the IUT supports COVM)

RECEIVE BACnet-SimpleACK-PDU

2. IF (if confirmed notifications were requested) THEN

BEFORE **Notification Fail Time**

RECEIVE ConfirmedCOVNotificationMultiple-Request,

'Subscriber Process Identifier' = X,

'Initiating Device Identifier' = IUT,

'Time Remaining' = (a value  $\sim$  L),

'Timestamp' = (any appropriate timestamp, if subscribed to timestamped notifications),

'List of COV Notifications' = (values appropriate to the subscribed to properties)

TRANSMIT BACnet-SimpleACK-PDU

ELSE

WHILE (notifications have not been received for all subscribed to items)

BEFORE **Notification Fail Time**

RECEIVE UnconfirmedCOVNotificationMultiple-Request,

'Subscriber Process Identifier' = X,

'Initiating Device Identifier' = IUT,

'Time Remaining' = (any valid value),

'Timestamp' = (any appropriate timestamp, if subscribed to timestamped notifications)

'List of COV Notifications' = (values appropriate to some or all of the properties subscribed to)

}

3. VERIFY Active\_COV\_Multiple\_Subscriptions = (a list of 5 COVM contexts as subscribed to)

### 9.X42.1.7 Supports Client-Supplied COV Increment

Purpose: To verify that the IUT abides by client supplied COV increments from SubscribeCOVPropertyMultiple requests.

Test Concept: A subscription for COVM notifications is made to a numeric property P1 which supports COVM in object O1. The COV Increment, N, is specified in the subscription request. Verify that the COV Increment N is stored in the COVM context for this subscription. The value of P1 is changed by less than the COV Increment, and the TD waits to ensure the IUT does not generate a notification. The value of P1 is changed such that the total change is more than N, and it is verified that the IUT sends a notification within the delay time.

Configuration Requirements: If the property being subscribed to has a related COV\_Increment property in the object, then the value of N should be significantly different than the value of the COV\_Increment property. If the object does not have a COV\_Increment property, then N shall be significantly different than the device's internal COV Increment.

Test Steps:

1. TRANSMIT SubscribeCOVPropertyMultiple-Request

'Subscriber Process Identifier' = (ID1: any valid process identifier),

'Issue Confirmed Notifications' = TRUE | FALSE,

'Lifetime' = L,

'Max Notification Delay' = (MND: any valid value),

'List of COV Subscription Specifications' = {('Monitored Object' = O1,  
 'List of COV References' = { (  
 'Monitored Property' = P1,  
 'COV Increment' = N,  
 'Timestamped' = TRUE | FALSE))  
 } }

2. RECEIVE BACnet-SimpleACK-PDU

3. VERIFY Active\_COV\_Multiple\_Subscriptions = (a list containing a COVM context for ID1 containing 1 entry for P1 with a COV\_Increment of N)

4. MAKE (P1's value change by less than COV Increment)

```

5. WAIT Notification Fail Time + MND
5. CHECK (verify that the IUT did not transmit a notification message for the monitored property)
6. MAKE (P1's value change such that the total change to P1 is slightly more than N)
7. IF (the subscription was for confirmed notifications) THEN
    BEFORE Notification Fail Time
        RECEIVE ConfirmedCOVNotificationMultiple-Request,
            'Subscriber Process Identifier' = ID1,
            'Initiating Device Identifier' = IUT,
            'Time Remaining' = (any valid value greater than 0 and less than L),
            'Timestamp' = (an appropriate timestamp, if subscribed to timestamped
                           notifications, otherwise absent)
            'List of COV Notifications' = (a list of values of length 1 indicating P1's new value)
        TRANSMIT BACnet-SimpleACK-PDU
    ELSE
        WHILE (notifications have not been received for all subscribed to items)
            BEFORE Notification Fail Time
                RECEIVE UnconfirmedCOVNotificationMultiple-Request,
                    'Subscriber Process Identifier' = (ID1),
                    'Initiating Device Identifier' = IUT,
                    'Time Remaining' = (any valid value greater than 0 and less than L),
                    'Timestamp' = (an appropriate timestamp, if subscribed to
                                 timestamped notifications, otherwise absent)
                    'List of COV Notifications' = (a list of values of length 1 indicating P1's new value)

```

### 9.X42.1.8 Updating Existing Subscriptions

Purpose: To verify that the IUT supports resubscriptions to extend the lifetime of COVM contexts.

Test Concept: A COVM subscription is made for 1 or more properties in the IUT. The IUT shall be made to transmit a notification to the TD and the Time Remaining value is validated. Before the subscription expires, the TD resubscribes with a different, and longer, lifetime and the new lifetime is verified in the resultant COVM notification.

Test Steps:

```

1. TRANSMIT SubscribeCOVPropertyMultiple-Request
    'Subscriber Process Identifier' = ID1,
    'Issue Confirmed Notifications' = TRUE | FALSE,
    'Lifetime' = L,
    'Max Notification Delay' = (MND: any valid value)
    'List of COV Subscription Specifications' = (PROPS: a valid list of subscriptions)
2. RECEIVE BACnet-SimpleACK-PDU
3. IF (the subscription was for confirmed notifications) THEN
    BEFORE Notification Fail Time
        RECEIVE ConfirmedCOVNotificationMultiple-Request,
            'Subscriber Process Identifier' = IUD1,
            'Initiating Device Identifier' = IUT,
            'Time Remaining' = (TR: TR ~ L),
            'Timestamp' = (an appropriate timestamp, if subscribed to timestamped
                           Notifications, otherwise absent)
            'List of COV Notifications' = (values appropriate to each entry in PROPS)
        TRANSMIT BACnet-SimpleACK-PDU
    ELSE
        WHILE (notifications have not been received for all subscribed to items)
            BEFORE Notification Fail Time
                RECEIVE UnconfirmedCOVNotificationMultiple-Request,
                    'Subscriber Process Identifier' = ID1,
                    'Initiating Device Identifier' = IUT,
                    'Time Remaining' = (TR: TR ~ L),
                    'Timestamp' = (an appropriate timestamp, if subscribed to timestamped
                                 notifications, otherwise absent),

```

```

        'List of COV Notifications' = (values appropriate to some or all entries in PROPS)
4.  MAKE (a change to a monitored property, P1, that should cause a COVM notification)
5.  WAIT N seconds, where  $L > N >$  the resolution of the IUT's COVM lifetime timer
6.  IF (the subscription was for confirmed notifications) THEN
    BEFORE Notification Fail Time
    RECEIVE ConfirmedCOVNotificationMultiple-Request,
        'Subscriber Process Identifier' = ID1,
        'Initiating Device Identifier' = IUT,
        'Time Remaining' = (TR:  $0 < TR < (L - N)$ ),
        'Timestamp' = (an appropriate timestamp, if subscribed to timestamped
                        notifications, otherwise absent)
        'List of COV Notifications' = (a list of values of length 1 indicating P1's new value)
    TRANSMIT BACnet-SimpleACK-PDU
ELSE
    WHILE (notifications have not been received for all subscribed to items)
    BEFORE Notification Fail Time
    RECEIVE UnconfirmedCOVNotificationMultiple-Request,
        'Subscriber Process Identifier' = ID1,
        'Initiating Device Identifier' = IUT,
        'Time Remaining' = (TR:  $0 < TR < (L - N)$ ),
        'Timestamp' = (an appropriate timestamp, if subscribed to
                        timestamped notifications, otherwise absent),
        'List of COV Notifications' = (values appropriate to some or all entries in PROPS)
7.  TRANSMIT SubscribeCOVPropertyMultiple-Request
    'Subscriber Process Identifier' = ID1,
    'Issue Confirmed Notifications' = (the same value used previously),
    'Lifetime' = ( $L2$ : where  $L < L2 \leq 28800$ ),
    'Max Notification Delay' = MND,
    'List of COV Subscription Specifications' = PROPS
8.  RECEIVE BACnet-SimpleACK-PDU
9.  IF (the subscription was for confirmed notifications) THEN
    BEFORE Notification Fail Time
    RECEIVE ConfirmedCOVNotificationMultiple-Request,
        'Subscriber Process Identifier' = ID1,
        'Initiating Device Identifier' = IUT,
        'Time Remaining' = (TR2:  $TR \approx L2$ ),
        'Timestamp' = (an appropriate timestamp, if subscribed to timestamped
                        notifications, otherwise absent)
        'List of COV Notifications' = (values appropriate to each entry in PROPS)
    TRANSMIT BACnet-SimpleACK-PDU
ELSE
    WHILE (notifications have not been received for all subscribed to items)
    BEFORE Notification Fail Time
    RECEIVE UnconfirmedCOVNotificationMultiple-Request,
        'Subscriber Process Identifier' = (the same identifier used in step 2),
        'Initiating Device Identifier' = IUT,
        'Time Remaining' = (TR2:  $TR2 \approx L2$ ),
        'Timestamp' = (an appropriate timestamp, if subscribed to timestamped
                        notifications, otherwise absent),
        'List of COV Notifications' = (values appropriate to some or all entries in PROPS)

```

### 9.X42.1.9 Canceling Subsets of COVM Subscriptions

Purpose: To verify that the IUT correctly cancels COVM subscriptions for some, not all, of the properties subscribed to in a COVM context.

Test Concept: A subscription for COVM notifications is established for multiple properties within the IUT. Before the subscriptions expire, one of the subscriptions is cancelled. Verify that the IUT's Active\_COV\_Multiple\_Subscriptions property only contains an entry for the remaining subscriptions.

Configuration Requirements: There are no active COVM subscription for properties in the IUT. If the IUT cannot be configured to have 2 properties which support COVM subscriptions, then this test shall be skipped.

Test Steps:

1. VERIFY Active\_COV\_Multiple\_Subscriptions = ()
2. TRANSMIT SubscribeCOVPropertyMultiple-Request
  - 'Subscriber Process Identifier' = (any valid process identifier),
  - 'Issue Confirmed Notifications' = FALSE,
  - 'Lifetime' = L,
  - 'Max Notification Delay' = (any valid notification delay),
  - 'List of COV Subscription Specifications' = (a list of 2 or more properties for which the IUT supports COVM)
3. RECEIVE BACnet-SimpleACK-PDU
4. WHILE (notifications have not been received for all subscribed to items)
  - BEFORE **Notification Fail Time**
  - RECEIVE UnconfirmedCOVNotificationMultiple-Request,
    - 'Subscriber Process Identifier' = ID1,
    - 'Initiating Device Identifier' = IUT,
    - 'Time Remaining' = (any value  $\sim$  L),
    - 'Timestamp' = (an appropriate timestamp, or absent if not requested)
    - 'List of COV Notifications' = (values appropriate to some or all of the properties subscribed to)
5. VERIFY Active\_COV\_Multiple\_Subscriptions = (a list with 1 COVM context containing all properties subscribed to)
6. TRANSMIT SubscribeCOVPropertyMultiple-Request
  - 'Subscriber Process Identifier' = ID1,
  - 'Issue Confirmed Notifications' = FALSE
  - 'Lifetime' = (absent)
  - 'Max Notification Delay' = (absent)
  - 'List of COV Subscription Specifications' = (CANCELLED: a subset of the properties subscribed to)
7. RECEIVE BACnet-SimpleACK-PDU
8. VERIFY Active\_COV\_Multiple\_Subscriptions = (a list with 1 COVM context containing all remaining properties subscribed to, excluding those in CANCELLED)

### 9.X42.1.10 Canceling Expired or Non-Existing Subscriptions

Purpose: To verify the IUT does not return an error when the client cancels a COVM subscription that doesn't match any of the COV contexts in the IUT's list of active subscriptions.

Test Concept: Send a SubscribeCOVPropertyMultiple request to cancel a subscription for property P1 in object O1, which is not in the list of subscriptions in the IUT's Active\_COV\_Multiple\_Subscriptions property. Verify that the IUT sends a BACnet-SimpleACK-PDU in response.

Configuration Requirements: The IUT is configured with 1 or more COVM subscriptions. One of the subscriptions is using a process identifier ID1 and includes a subscription to property P1 in object O1. Property P2 in object O2 shall not be included in the subscriptions for ID1 (but may in subscriptions using a different process identifier). Where possible, P2 in O2 should be a property for which the IUT supports COVM subscriptions.

Test Steps:

1. READ COVM\_LIST = Active\_COV\_Multiple\_Subscriptions
2. CHECK (COVM\_LIST contains an COVM context with a process identifier of ID1 and includes a subscription to property P1 in object O1)
3. TRANSMIT SubscribeCOVPropertyMultiple-Request
  - 'Subscriber Process Identifier' = (any valid process identifier which is not ID1),
  - 'Issue Confirmed Notifications' = (the value matching the entry for ID1),
  - 'Lifetime' = (absent),

- 'Max Notification Delay' = (absent),
- 'List of COV Subscription Specifications' = (a list with 1 entry matching the subscription details for P1 in O1)
- 4. RECEIVE BACnet-SimpleACK-PDU
- 5. VERIFY Active\_COV\_Multiple\_Subscriptions = COVM\_LIST
- 6. TRANSMIT SubscribeCOVPropertyMultiple-Request
  - 'Subscriber Process Identifier' = ID1,
  - 'Issue Confirmed Notifications' = (the value matching the entry for ID1),
  - 'Lifetime' = (absent),
  - 'Max Notification Delay' = (absent),
  - 'List of COV Subscription Specifications' = (a list with 1 entry referencing P2 in O2)
- 7. RECEIVE BACnet-SimpleACK-PDU
- 8. VERIFY Active\_COV\_Multiple\_Subscriptions = COVM\_LIST

### 9.X42.1.11 Subscription Expiration Test

Purpose: To verify that the IUT removes subscriptions from the list of active subscriptions once the subscription lifetime has elapsed.

Test Concept: A COVM subscription is made for 1 or more properties in the IUT. One of the subscribed to properties is made to change, and it is verified that the IUT transmits a notification to the TD containing an accurate Time Remaining value. The tester then waits for the subscription to expire, and it is verified that Active\_COV\_Multiple\_Subscriptions is updated. The property is changed again, and it is verified that the IUT does not send a notification.

Configuration Requirements: No existing subscription exists for ID1 for the TD. A value for L is chosen which is long enough to complete the initial test steps, but which is short enough to wait for it to expire.

Test Steps:

1. TRANSMIT SubscribeCOVPropertyMultiple-Request
  - 'Subscriber Process Identifier' = ID1,
  - 'Issue Confirmed Notifications' = TRUE | FALSE,
  - 'Lifetime' = L,
  - 'Max Notification Delay' = 0,
  - 'List of COV Subscription Specifications' = (a valid list of subscriptions)
2. RECEIVE BACnet-SimpleACK-PDU
3. IF (the subscription was for confirmed notifications) THEN
  - BEFORE **Notification Fail Time**
  - RECEIVE ConfirmedCOVNotificationMultiple-Request,
    - 'Subscriber Process Identifier' = ID1,
    - 'Initiating Device Identifier' = IUT,
    - 'Time Remaining' = (a value  $\sim$  L),
    - 'Timestamp' = (an appropriate timestamp, if subscribed to timestamped Notifications, otherwise absent),
    - 'List of COV Notifications' = (values appropriate to the properties subscribed to)
  - TRANSMIT BACnet-SimpleACK-PDU
- ELSE
  - WHILE (notifications have not been received for all subscribed to items)
  - BEFORE **Notification Fail Time**
  - RECEIVE UnconfirmedCOVNotificationMultiple-Request,
    - 'Subscriber Process Identifier' = ID1,
    - 'Initiating Device Identifier' = IUT,
    - 'Time Remaining'  $\sim$  (a value approximately equal to, but not greater than, the requested subscription lifetime),
    - 'Timestamp' = (an appropriate timestamp, if subscribed to timestamped Notifications, otherwise absent),
    - 'List of COV Notifications' = (values appropriate to some or all of the properties subscribed to)
4. MAKE (a change to a monitored property, P1, that should cause a COVM notification)

5. WAIT N seconds, where  $L > N >$  the resolution of the IUT's COVM lifetime timer
6. IF (the subscription was for confirmed notifications) THEN
  - BEFORE **Notification Fail Time**
  - RECEIVE ConfirmedCOVNotificationMultiple-Request,
    - 'Subscriber Process Identifier' = ID1,
    - 'Initiating Device Identifier' = IUT,
    - 'Time Remaining' = (TR:  $0 < TR < (L - N)$ ),
    - 'Timestamp' = (an appropriate timestamp, if subscribed to timestamped Notifications, otherwise absent)
    - 'List of COV Notifications' = (a list of values of length 1 indicating P1's value)
  - TRANSMIT BACnet-SimpleACK-PDU
- ELSE
  - BEFORE **Notification Fail Time**
  - RECEIVE UnconfirmedCOVNotificationMultiple-Request,
    - 'Subscriber Process Identifier' = ID1,
    - 'Initiating Device Identifier' = IUT,
    - 'Time Remaining' = (TR:  $0 < TR < (L - N)$ ),
    - 'Timestamp' = (an appropriate timestamp, if subscribed to timestamped Notifications, otherwise absent)
    - 'List of COV Notifications' = (values appropriate to the properties subscribed to)
7. WAIT L seconds
8. MAKE (a change to a monitored property that would cause a COVM notification if there were an active subscription)
9. CHECK (verify that the IUT did not transmit a COVM notification message for the modified property)
10. VERIFY Active\_COV\_Multiple\_Subscriptions = (a list which does not contain a COVM context for ID1)

## 9.X42.2 Negative SubscribeCOVPropertyMultiple Service Execution Tests

### 9.X42.2.1 The Monitored Object Does Not Support COVM Notification

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

Test Concept: A subscription for COVM notifications is made which includes a request for property P1 in object O1 where O1 does not support COVM. All requested subscriptions before O1 are selected such that they would succeed if O1 were not in the list. It is verified that the IUT returns the correct error indicating O1 and P1 as the first failed element encountered.

Configuration Requirements: The object, O1, shall not support COVM notification for any of its properties. If the IUT cannot be configured in this manner, then this test shall be skipped.

Test Steps:

1. TRANSMIT SubscribeCOVPropertyMultiple-Request
  - 'Subscriber Process Identifier' = (any valid process identifier),
  - 'Issue Confirmed Notifications' = TRUE | FALSE,
  - 'Lifetime' = (any valid lifetime),
  - 'Max Notification Delay' = (any valid value smaller than the lifetime),
  - 'List of COV Subscription Specifications' = (a list of subscriptions for properties in O1 and optionally other objects with P1 being the first property requested from O1)
2. RECEIVE BACnet-Error-PDU,
  - 'First-Failed-Subscription' = {
    - 'Monitored Object Identifier' = O1,
    - 'Monitored Property Reference' = P1,
    - 'Error Class' = OBJECT,
    - 'Error Code' = OPTIONAL\_FUNCTIONALITY\_NOT\_SUPPORTED

### 9.X42.2.2 The Monitored Property Does Not Support COVM Notification

**Purpose:** To verify that the IUT correctly responds to a SubscribeCOVPropertyMultiple request to establish a subscription when the monitored property does not support COVM notifications.

**Test Concept:** A subscription for COVM notifications is made which includes a request for property P1 in object O1 where P1 does not support COVM. All requested subscriptions before P1 are selected such that they would succeed if P1 were not in the list. It is verified that the IUT returns the correct error indicating O1 and P1 as the first failed element encountered.

**Configuration Requirements:** The object, O1, shall support COVM notification for any of its properties. If the IUT does not support objects for which COVM is supported for only a subset of the properties, then this test shall be skipped.

**Test Steps:**

1. TRANSMIT SubscribeCOVPropertyMultiple-Request
  - 'Subscriber Process Identifier' = (any valid process identifier),
  - 'Issue Confirmed Notifications' = TRUE | FALSE,
  - 'Lifetime' = (any valid lifetime),
  - 'Max Notification Delay' = (any valid value smaller than the lifetime),
  - 'List of COV Subscription Specifications' = (a list of subscriptions for properties in O1 and optionally other objects with P1 being the first property which cannot be subscribed to)
2. RECEIVE BACnet-Error-PDU,
  - 'First-Failed-Subscription' = {
    - 'Monitored Object Identifier' = O1,
    - 'Monitored Property Reference' = P1,
    - 'Error Class' = PROPERTY,
    - 'Error Code' = NOT\_COV\_PROPERTY

### 9.X42.2.3 Monitored Object Does Not Exist

**Purpose:** To verify that the IUT correctly responds to a SubscribeCOVPropertyMultiple request to establish a subscription when the monitored object does not exist.

**Test Concept:** A subscription for COVM notifications is made which includes a request for property P1 in object O1 where O1 does not exist but would support COVM for P1 if it did. All requested subscriptions before O1 are selected such that they would succeed if O1 were not in the list. It is verified that the IUT returns the correct error indicating O1 and P1 as the first failed element encountered.

**Configuration Requirements:** The object, O1, shall be of a type for which the IUT supports COVM notifications for property P1. If the IUT cannot be configured in this manner, then this test shall be skipped.

**Test Steps:**

1. TRANSMIT SubscribeCOVPropertyMultiple-Request
  - 'Subscriber Process Identifier' = (any valid process identifier),
  - 'Issue Confirmed Notifications' = TRUE | FALSE,
  - 'Lifetime' = (any valid lifetime),
  - 'Max Notification Delay' = (any valid value smaller than the lifetime),
  - 'List of COV Subscription Specifications' = (a list of subscriptions for properties in O1 and optionally other objects with P1 being the first property requested from O1)
2. RECEIVE BACnet-Error-PDU,
  - 'First-Failed-Subscription' = {
    - 'Monitored Object Identifier' = O1,
    - 'Monitored Property Reference' = P1,
    - 'Error Class' = OBJECT,
    - 'Error Code' = UNKNOWN\_OBJECT



#### 9.X42.2.4 Monitored Property Does Not Exist

Purpose: To verify that the IUT correctly responds to a SubscribeCOVPropertyMultiple request to establish a subscription when the monitored property does not exist.

Test Concept: A subscription for COVM notifications is made which includes a request for property P1 in object O1 where P1 does not exist in O1. All requested subscriptions before P1 in O1 are selected such that they would succeed if P1 in O1 were not in the list. It is verified that the IUT returns the correct error indicating O1 and P1 as the first failed element encountered.

Configuration Requirements: The object, O1, shall exist, shall not contain P1, and be of a type for which the IUT supports COVM notifications. If the IUT cannot be configured in this manner, then this test shall be skipped.

Test Steps:

1. TRANSMIT SubscribeCOVPropertyMultiple-Request
  - 'Subscriber Process Identifier' = (any valid process identifier),
  - 'Issue Confirmed Notifications' = TRUE | FALSE,
  - 'Lifetime' = (any valid lifetime),
  - 'Max Notification Delay' = (any valid value smaller than the lifetime),
  - 'List of COV Subscription Specifications' = (a list of subscriptions for properties in O1 and optionally other objects with P1 being the first property which cannot be subscribed to)
2. RECEIVE BACnet-Error-PDU,
  - 'First-Failed-Subscription' = {
    - 'Monitored Object Identifier' = O1,
    - 'Monitored Property Reference' = P1,
    - 'Error Class' = PROPERTY,
    - 'Error Code' = UNKNOWN\_PROPERTY

#### 9.X42.2.5 Array Index Provided But Property is Not an Array

Purpose: To verify that the IUT correctly responds to a SubscribeCOVPropertyMultiple request to establish a subscription when the monitored property is not an array, but an array index is provided.

Test Concept: A subscription for COVM notifications is made which includes a request for property P1, with an array index, in object O1 where the IUT supports COVM for P1 in O1 but P1 is not an array. All requested subscriptions before P1 in O1 are selected such that they would succeed if P1 in O1 were not in the list. It is verified that the IUT returns the correct error indicating O1 and P1 as the first failed element encountered.

Configuration Requirements: The property P1 shall be one which supports COVM and is not an array. If the IUT cannot be configured in this manner, then this test shall be skipped.

Test Steps:

1. TRANSMIT SubscribeCOVPropertyMultiple-Request
  - 'Subscriber Process Identifier' = (any valid process identifier),
  - 'Issue Confirmed Notifications' = TRUE | FALSE,
  - 'Lifetime' = (any valid lifetime),
  - 'Max Notification Delay' = (any valid value smaller than the lifetime),
  - 'List of COV Subscription Specifications' = (a list of subscriptions for properties in O1 and optionally other objects with P1 being the first property which cannot be subscribed to. An array index shall be included in the entry for P1)
2. RECEIVE BACnet-Error-PDU,
  - 'First-Failed-Subscription' = {
    - 'Monitored Object Identifier' = O1,
    - 'Monitored Property Reference' = P1,

```
'Error Class' = PROPERTY,
'Error Code' = PROPERTY_IS_NOT_AN_ARRAY
}
```

### 9.X42.2.6 Array Index Provided Is Out Of Range

Purpose: To verify that the IUT correctly responds to a SubscribeCOVPropertyMultiple request to establish a subscription when the monitored property is an array, but the provided array index is outside the range of the array.

Test Concept: A subscription for COVM notifications is made which includes a request for an array property P1, with an array index, in object O1 where the IUT supports COVM for P1. All requested subscriptions before P1 in O1 are selected such that they would succeed if P1 in O1 were not in the list. It is verified that the IUT returns the correct error indicating O1 and P1 as the first failed element encountered.

Configuration Requirements: If the IUT does not support COVM on any array properties, then this test shall be skipped.

Test Steps:

1. TRANSMIT SubscribeCOVPropertyMultiple-Request
 

|                                             |                                                                                                                                                                                                                                           |
|---------------------------------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| 'Subscriber Process Identifier' =           | (any valid process identifier),                                                                                                                                                                                                           |
| 'Issue Confirmed Notifications' =           | TRUE   FALSE,                                                                                                                                                                                                                             |
| 'Lifetime' =                                | (any valid lifetime),                                                                                                                                                                                                                     |
| 'Max Notification Delay' =                  | (any valid value smaller than the lifetime),                                                                                                                                                                                              |
| 'List of COV Subscription Specifications' = | (a list of subscriptions for properties in O1 and optionally other objects with P1 being the first property which cannot be subscribed to. The array index included in the entry for P1 shall be larger than the number of entries in P1) |
2. RECEIVE BACnet-Error-PDU,
 

|                               |                                      |
|-------------------------------|--------------------------------------|
| 'First-Failed-Subscription' = | {                                    |
|                               | 'Monitored Object Identifier' = O1,  |
|                               | 'Monitored Property Reference' = P1, |
|                               | 'Error Class' = PROPERTY,            |
|                               | 'Error Code' = INVALID_ARRAY_INDEX   |
|                               | }                                    |

### 9.X42.2.7 No Space to Add List Element

Purpose: To verify that the IUT correctly responds to a SubscribeCOVPropertyMultiple request to establish a subscription when there is no space for a subscription.

Test Concept: Repeatedly subscribe to the same object each time with a different Process Identifier until the device runs out of resources and returns the appropriate error.

Configuration Requirements: If the device cannot be configured such that the maximum number of subscriptions the IUT can accept is less than 10000, then this test shall be skipped.

Test Steps:

REPEAT PID = (1 through the maximum number of subscriptions the IUT can accept plus 1 or until the IUT returns an Error-PDU) {

1. TRANSMIT SubscribeCOVPropertyMultiple-Request
 

|                                             |                                                         |
|---------------------------------------------|---------------------------------------------------------|
| 'Subscriber Process Identifier' =           | PID,                                                    |
| 'Issue Confirmed Notifications' =           | TRUE   FALSE,                                           |
| 'Lifetime' =                                | (any valid lifetime large enough to complete the test), |
| 'Max Notification Delay' =                  | (any valid value smaller than the lifetime),            |
| 'List of COV Subscription Specifications' = | (any valid list of subscriptions)                       |
2. RECEIVE BACNET-SimpleACK-PDU

```
| (BACnet-Error-PDU,
  'Error Class' = RESOURCES,
  'Error Code' = NO_SPACE_TO_ADD_LIST_ELEMENT)
}
```

#### 9.X42.2.8 The Lifetime Parameter is Out Of Range

Purpose: To verify that the IUT correctly responds to a SubscribeCOVPropertyMultiple request to establish a subscription when the Lifetime parameter is out of range.

Configuration Requirements: If the device supports lifetimes across the full range of valid lifetimes, then this test shall be skipped.

Test Steps:

1. TRANSMIT SubscribeCOVPropertyMultiple-Request
  - 'Subscriber Process Identifier' = (any valid process identifier),
  - 'Issue Confirmed Notifications' = TRUE | FALSE,
  - 'Lifetime' = (a value larger than that supported by the IUT)
  - 'Max Notification Delay' = (any valid value smaller than the lifetime),
  - 'List of COV Subscription Specifications' = (any valid list of subscriptions)
2. RECEIVE BACnet-Error-PDU,
  - 'Error Class' = SERVICES,
  - 'Error Code' = VALUE\_OUT\_OF\_RANGE

#### 9.X42.2.9 The Max Notification Delay Parameter is Out Of Range

Purpose: To verify that the IUT correctly responds to a SubscribeCOVPropertyMultiple request to establish a subscription when the Max Notification Delay parameter is out of range.

Configuration Requirements: If the device supports Max Notification Delays across the full range of valid values, then this test shall be skipped.

Test Steps:

1. TRANSMIT SubscribeCOVPropertyMultiple-Request
  - 'Subscriber Process Identifier' = (any valid process identifier),
  - 'Issue Confirmed Notifications' = TRUE | FALSE,
  - 'Lifetime' = (any valid value large enough to complete the test),
  - 'Max Notification Delay' = (a value larger than supported by the IUT),
  - 'List of COV Subscription Specifications' = (any valid list of subscriptions)
2. RECEIVE BACnet-Error-PDU,
  - 'Error Class' = SERVICES,
  - 'Error Code' = VALUE\_OUT\_OF\_RANGE

#### 9.X42.2.10 The Max Notification Delay is Greater Than the Lifetime

Purpose: To verify that the IUT correctly responds to a SubscribeCOVPropertyMultiple request to establish a subscription when the Max Notification Delay parameter is greater than the Lifetime parameter.

Test Steps:

1. TRANSMIT SubscribeCOVPropertyMultiple-Request
  - 'Subscriber Process Identifier' = (any valid process identifier),
  - 'Issue Confirmed Notifications' = TRUE | FALSE,
  - 'Lifetime' = (a value supported by the IUT but within the normal range of

Max Notification Delay)  
'Max Notification Delay' = (a value greater than the lifetime),  
'List of COV Subscription Specifications' = (any valid list of subscriptions)  
2. RECEIVE BACnet-Error-PDU,  
'Error Class' = SERVICES,  
'Error Code' = VALUE\_OUT\_OF\_RANGE

## 135.1-2019s-10 Improve existing DEFINITIONS

### Rationale

New definitions are added to ANSI/ASHRAE Standard 135.1-2019.

In addition, the reference to the BACnet standard is updated.

[Change **Clause 3**, p. 1]

### 3. DEFINITIONS

All definitions from ANSI/ASHRAE Standard ~~135-2016~~/135-2020 also apply to this addendum.

[Add new **Clause 3.5**, p. 1]

#### 3.5 Common language used in tests

'any valid value': Any valid value refers to any value of the correct data type and within the vendor's range specified for the property this is applied to.

'any appropriate password': Any password that meets the Configuration Requirements specified in the test or test section. Passwords when required by the vendor are required to be no more than 20 characters.

'reset': Some tests require to reset the IUT. Reset includes power cycle via switch, power cycle via loss of power, and reinitializeDevice WARMSTART. As defined by the BACnet standard, "WARMSTART shall mean to reboot the device and start over, retaining all data and programs that would normally be retained during a brief power outage."

## 135.1-2019s-11 Improve the EPICS FILE FORMAT and the EPICS CONSISTENCY TESTS

### Rationale

New fail timers are added to ANSI/ASHRAE Standard 135.1-2019.

In addition, the EPICS Consistency Tests are improved.

## 4. ELECTRONIC PICS FILE FORMAT

### 4.5 Sections of the EPICS File

[Change **Clause 4.5.9**, p. 8]

#### 4.5.9 Timers

This section defines timer values that are used to determine when a test has failed because an appropriate response has not been observed by the TD. A Real value in seconds must be provided for each timer. See 6.3.

Fail Times: ↵

{ ↵

Notification Fail Time: □ ↵

Internal Processing Fail Time: □ ↵

Minimum ON/OFF Time: □ ↵

Schedule Evaluation Fail Time: □ ↵

External Command Fail Time: □ ↵

Program Object State Change Fail Time: □ ↵

Acknowledgement Fail Time: □ ↵

Slave Proxy Confirm Interval: □ ↵

Unconfirmed Response Fail Time: □ ↵

*Channel Write Fail Time: □ ↵*

*Auto Negotiation Fail Time: □ ↵*

*Activate Changes Fail Time: □ ↵*

*Foreign Device Registration Fail Time: □ ↵*

} ↵

[Change **Clause 5**, p. 10]

[Reviewer Note: Improved the language in this set of tests to clarify the exact requirement of the test.]

## 5. EPICS CONSISTENCY TESTS

These tests are static tests of the EPICS and do not involve interrogating the IUT. There are no Test Configuration or Test Step sections with TCSL in these tests because the tests are static tests of the EPICS and not tests of the IUT itself.

Each implementation shall be tested to ensure consistency among interrelated data elements. These tests shall include:

- (a) All object types required by the specified BIBBs shall be indicated as supported in the Standard Object Types Supported section of the EPICS.
- (b) A minimum of one instance of each object type required by the specified BIBBs shall be included in the test database.

- (c) The Protocol\_Object\_Types\_Supported property of the Device object in the test database shall indicate support for each object type required by the supported BIBBs.
- (d) All application services required by the supported BIBBs shall be indicated as supported in the BACnet Standard Application Services Supported section of the EPICS with Initiate and Execute indicated as required by the supported BIBBs.
- (e) The Protocol\_Services\_Supported property of the Device object in the test database shall indicate support for each application service for which the supported BIBBs requires support for execution of the service.
- (f) The object types listed in the Standard Object Types Supported section of the EPICS shall have a one-to-one correspondence with object types listed in the Protocol\_Object\_Types\_Supported property of the Device object contained in the test database.
- (g) For each object type listed in the Standard Object Types Supported\* section of the EPICS there shall be at least one object of that type in the test database. It is permissible for there to be no instance of the File object type if File objects are dynamically creatable and come into existence only temporarily during Backup and restore.  
*\*An object type is supported if it can be made to exist in the IUT's database.*
- (h) There shall be a one-to-one correspondence between the objects listed in the Object\_List property of the Device object and the objects included in the test database. The Object\_List property and the test database shall both include all proprietary objects. Properties of proprietary objects that are not required by BACnet Clause 23.4.3 need not be included in the test database.
- (i) For each object included in the test database, all required properties for that object as defined in Clause 12 of BACnet shall be present. Standard properties which are not defined for the implemented Protocol\_Revision shall not be present. In addition, if any of the properties supported for an object require the conditional presence of other properties, their presence shall be verified.
- (j) For each property that is required to be writable, or conditionally writable, that property shall be marked as writable, or conditionally writable, in the EPICS.
- (k) The length of the Protocol\_Services\_Supported bitstring shall have the number of bits defined for BACnetProtocolServicesSupported for the IUT's declared protocol revision.
- (l) The length of the Protocol\_Object\_Types\_Supported bitstring shall have the number of bits defined for BACnetObjectTypesSupported for the IUT's declared protocol revision
- (m) For each object included in the test database, any properties that are deprecated or removed shall not appear after the Protocol\_Revision in which the property was deprecated or removed.
- (n) If the Protocol\_Revision property is present in the Device object and its value is greater than or equal to 14, *the Property\_List property of each object included in the test database shall have one entry for each property present, including non-standard properties with the exception of Object\_Type, Object\_Identifier, Object\_Name, and Property\_List*~~for each object included in the test database, the Property\_List property shall have one entry for each property present, including non-standard properties, but excluding Object\_Type, Object\_Identifier, Object\_Name, and Property\_List.~~
- (o) If the Segmentation\_Supported property in the Device object is SEGMENTED\_BOTH or SEGMENTED\_RECEIVE, then the value of the Max\_Segments\_Accepted property of the Device object shall be greater than 1.
- (p) *For each property that is required to be read-only, that property shall not be marked as writable, or conditionality writable, in the EPICS.*

## 135.1-2019s-12 Add new and improve existing CONVENTIONS FOR SPECIFYING BACnet CONFORMANCE TESTS

### Rationale

New conventions for specifying BACnet conformance tests in ANSI/ASHRAE Standard 135.1-2019 are added. In addition, the language for Internal Processing Fail Time is improved.

## 6. CONVENTIONS FOR SPECIFYING BACnet CONFORMANCE TESTS

### 6.2 TCSL Statements

[Add new **Clause 6.2.14**, p. 18]

#### 6.2.14 Assignment Statement

The assignment statement is used to set the value of a TCSL variable

<assignment statement> := <variable> '=' '(' <value description> ')'

The <value description> is a simple English phrase describing the content that is to be placed into the variable. It may reference other variables already in use by the test. For example:

READ X = O1, Present\_Value  
READ Y = O2, Present\_Value  
MAX = (the larger of X and Y)

### 6.3 Time Dependencies

[Change **Clause 6.3.2**, p. 18]

#### 6.3.2 Internal Processing Fail Time

The Internal Processing Fail Time is the elapsed time, in seconds, between the receipt of a write to a BACnet property or some other event that changes the value of the property and when a test is considered to have failed because the property value has not been updated.

*The Internal Processing Fail Time contained in the EPICS is the maximum value for all situations. In order to reduce test time, it is acceptable that a shorter Internal Processing Fail Time value be used on a per test basis where the shorter time will not negatively impact the test result.*

[Add new **Clause 6.3.X1**, p. 19]

#### 6.3.X1 Channel Write Fail Time

The Channel Write Fail Time is the elapsed time, in seconds, between a change to the Present\_Value of a Channel object and when a test is considered to have failed because the first write operation associated with the newly written value state has not been performed. If the Channel object has multiple target properties to write to, the time to write all of them would be less than or equal to the number of target properties times this value.

[Add new **Clause 6.3.X2**, p. 19]



### 6.3.X2 Auto-negotiation Fail Time

The **Auto-negotiation Fail Time** is the elapsed time, in seconds, between when auto-negotiation is requested and when a test is considered to have failed because the IUT has not completed auto-negotiation of link speed.

[Add new **Clause 6.3.X3**, p. 19]

### 6.3.X3 Activate Changes Fail Time

The **Activate Changes Fail Time** is the elapsed time, in seconds, that the IUT requires in order to complete activation of changes in a Network Port object, including any required reset of the device and when a test is considered to have failed because the IUT has not completed activation of the Network Port changes.

[Add new **Clause 6.3.X4**, p. 19]

### 6.3.X4 Foreign Device Registration Fail Time

The **Foreign Device Registration Fail Time** is the elapsed time, in seconds, between when the device sends a Register-Foreign-Device request and when the device considers the request to have failed due to having not received a BVLC-Result for the registration request.

[Add new **Clause 6.3.X5**, p. 19]

### 6.3.X5 Audit Notification Forwarder Fail Time

The **Audit Notification Forwarder Fail Time** is the elapsed time, in seconds, between when a forwarding audit log receives an audit notification and when a test is considered to have failed because the expected audit notification message has not been transmitted.

[Add new **Clause 6.X6**, p. 19]

## 6.X6 Test Execution Considerations

There are some implicit test considerations which apply to all tests.

### 6.X6.1 Value Comparisons

Value comparisons, such as those in conditionals and statements (6.1.1 and 6.2) should always take into account the resolution constraints of 4.4.2, such that if any value is written with a higher or finer granularity than the IUT supports, any and all subsequent comparisons to that written value shall not result in the test step failing once the constraints have been applied to the value from the testing device.

Any value from the IUT is compared to a value from the testing device, the comparison shall not fail as long as the values are identical once the resolution constraints are applied to the external value. Devices may either truncate or round values written/received at a finer resolution than they claim in 4.4.2. This should be consistent across all values of that datatype.

Floating point values may be stored in the IUT using a different precision than what the value is encoded with from the testing device. Comparisons to these types of values should always consider that the IUT can round or truncate them, and that the value may also lose precision when stored, although the IUT must always present the stored value using full precision on a subsequent read.

Example: An IUT has a property with type REAL which has a resolution of 0.5. If a TD writes 0.75, it is acceptable for the IUT to present, on a subsequent read, either 1.0 or 0.5 for the property value so long as it is consistent.

### **6.X6.2 Functional Expectations**

Because devices may store information in truncated or rounded format due to internal limitations, they may not behave exactly as all test cases prescribe. In these cases, it is acceptable for the IUT to present or act on information based on the granularity supported by the IUT. For example, an IUT may be configured with a schedule which begins at 12:01:01, but since the device only supports granularity of 1 minute, the scheduled behavior may begin at 12:01:00 or 12:02:00.

### **6.X6.3 Complex Datatypes**

Since all complex datatypes can be broken down into one or more primitive datatypes, value comparisons and functional expectations from the above sections shall apply to each of the components that comprise a complex datatype. When there are relationships present within the fields of a complex datatype, the IUT shall continue to meet the functional expectations between all fields of complex datatypes when applying any rounding or truncation.

## 135.1-2019s-13 Add new and improve existing SPECIAL FUNCTIONALITY TESTS

### Rationale

Errors have been identified in a number of special functionality tests in ANSI/ASHRAE Standard 135.1-2019. In addition, the test coverage is increased by the addition of new tests.

## 13. SPECIAL FUNCTIONALITY TESTS

### 13.1 Segmentation

#### 13.1.12 IUT Abort When Segmentation Not Possible

[Change **Clause 13.1.12.1**, p. 633]

[Reviewer Note: Added 'Server' flag, in consequence of BTL-CRR-0177.]

##### 13.1.12.1 IUT Does Not Support Segmented Response

Purpose: To verify that the IUT returns the correct abort message when it does not support segmented responses and a response would be larger than 1 segment.

~~BACnet Reference Clause: 5.4.5.3.~~

Test Concept: The TD uses ReadPropertyMultiple to ask for more data than can fit in a single segment. The TD also specifies that the smallest (50 octet) segment size be used for the response. The data that are requested is the Object\_Identifier property of the Device object of the IUT. The number of copies of the data that is requested is one more than the maximum number which would fit in a 50-octet segment.

Configuration Requirements: The IUT supports execution of the ReadPropertyMultiple service but does not support transmission of segmented responses.

Test Steps:

1. TRANSMIT ReadPropertyMultiple-Request,  
    'max-APDU-length-accepted' = B'0000',  
    'segmented-response-accepted' = TRUE,  
    'Object Identifier' = (Device, X),  
    'Property Identifier' = Object\_Identifier,  
    'Property Identifier' = Object\_Identifier,  
    'Property Identifier' = Object\_Identifier,  
    'Property Identifier' = Object\_Identifier,  
    'Property Identifier' = Object\_Identifier
2. RECEIVE BACnet-Abort-PDU,  
    'Server' = TRUE,  
    'Abort Reason' = SEGMENTATION\_NOT\_SUPPORTED

[Add new **Clause 13.1.12.X1**, p. 635]

##### 13.1.12.X1 Reading with maximum-segments-accepted bit pattern B'000'

Purpose: To verify that the IUT implements at least support for two segments, when the 'max-segments-accepted' parameter that it sends is B'000'.

Configuration Requirements: If the IUT cannot be configured to issue any BACnet-Confirmed-Request-PDU with 'segmented-response-accepted' = TRUE and the 'max-segments-accepted' parameter equal to B'000', then this test shall be skipped.

1. RECEIVE BACnet-Confirmed-Request-PDU,  
    'segmented-response-accepted' = TRUE  
    'max-segments-accepted' = B'000'
2. TRANSMIT BACnet-ComplexACK-PDU,  
    'segmented-message' = TRUE,  
    'more-follows' = TRUE,  
    'sequence-number' = 0,  
    'proposed-window-size' = (any valid value)
3. RECEIVE BACnet-SegmentACK-PDU,  
    'server' = FALSE,  
    'negativeACK' = FALSE
4. TRANSMIT BACnet-ComplexACK-PDU,  
    'segmented-message' = TRUE,  
    'more-follows' = FALSE,  
    'sequence-number' = 1
5. RECEIVE BACnet-SegmentACK-PDU,  
    'server' = FALSE,  
    'negativeACK' = FALSE

### 13.5 Slave Proxy Tests

[Change **Clause 13.5.3**, p. 643]

[Reviewer Note: The third Who-Is should be a remote broadcast as the global broadcast case is already tested by the first Who-Is. Steps 3 and 4 rely on functionality not required by the standard and should be deleted (see IR 135.1-2019-1).]

#### 13.5.3 Proxy Test

~~BACnet Reference Clauses: 12.11.40, 12.11.41, 12.11.42, 12.11.43, and 16.10.2.~~

Purpose: This test verifies that an IUT correctly proxies for slaves that are listed in its Slave\_Address\_Binding property.

Test Concept: Configure the IUT so that it will proxy for a slave device and wait for the IUT to find and confirm the slave. Issue Who-Is requests in all forms to ensure that the IUT correctly proxies the I-Am responses for the slave device. The test should be repeated with the TD connected to the MS/TP segment, and with the TD connected to a different BACnet network.

Configuration Requirements: The MS/TP network, *NI*, shall contain a slave device at address A1 with a device identifier of D1. The IUT shall be configured to perform slave proxying. The test starts after the IUT has successfully found and confirmed the slave device. This test shall be repeated once with the TD connected to the MS/TP network and once with the TD connected to a different BACnet network.

Test Steps:

1. TRANSMIT DESTINATION = GLOBAL BROADCAST, Who-Is-Request
2. BEFORE **Unconfirmed Response Fail Time**  
    RECEIVE  
        DESTINATION = GLOBAL BROADCAST | LOCAL BROADCAST | TD  
        SOURCE = A1  
        I-Am-Request,  
        'I Am Device Identifier' = ~~(the slave's Device object's Object\_Identifier),~~ D1,  
        'Max APDU Length Accepted' = (the slave's value for this property),  
        'Segmentation Supported' = FALSE,  
        'Vendor Identifier' = (the slave's value for this property)
- ~~3. TRANSMIT DESTINATION = A1, Who-Is~~

#### 4. ~~BEFORE Unconfirmed Response Fail Time~~

~~RECEIVE~~

~~DESTINATION = GLOBAL BROADCAST | LOCAL BROADCAST | TD~~

~~SOURCE = A1~~

~~I Am Request,~~

~~'I Am Device Identifier' = (the slave's Device object's Object Identifier),~~

~~'Max APDU Length Accepted' = (the slave's value for this property),~~

~~'Segmentation Supported' = FALSE,~~

~~'Vendor Identifier' = (the slave's value for this property)~~

#### 3. *IF the TD is not connected to the MS/TP network*

~~TRANSMIT DESTINATION = GLOBAL BROADCAST, Who Is~~

~~TRANSMIT~~

~~DA = IUT, -- targeting the MS/TP network~~

~~DNET = NI,~~

~~DLEN = 0,~~

~~Who-Is-Request~~

#### 64. BEFORE Unconfirmed Response Fail Time

RECEIVE

DESTINATION = GLOBAL BROADCAST | ~~LOCAL~~ REMOTE BROADCAST | TD

SOURCE = A1

I-Am-Request,

'I Am Device Identifier' = ~~(the slave's Device object's Object Identifier),~~ DI,

'Max APDU Length Accepted' = (the slave's value for this property),

'Segmentation Supported' = FALSE,

'Vendor Identifier' = (the slave's value for this property)

## 13.8 Backup and Restore Procedure Tests

### 13.8.1 Backup and Restore Execution Tests

[Change **Clause 13.8.1.1**, p. 645]

[Reviewer Note: Corrected the Backup\_And\_Restore\_State in step 22. Changed test to account for optional properties.]

#### 13.8.1.1 Execution of Full Backup and Restore Procedure

Purpose: This test case verifies that the IUT can execute a full Backup and Restore procedure.

Test Concept: This test takes the IUT through a successful Backup and then a successful Restore procedure. The Database\_Revision and Last\_Restore\_Time properties are noted before the procedure begins for later comparison. The IUT is then commanded to enter the Backup state; all the files are read, and the IUT is commanded to end the backup. If the Database\_Revision property can be changed by means other than the restore procedure, it is modified and checked to ensure that it incremented correctly; then the IUT is commanded to enter the Restore state. If the file objects do not exist on the IUT, the TD will create them in the IUT. The files are then truncated to size 0, the file contents are written to the IUT, and the IUT is commanded to end the restore. The Database\_Revision and Last\_Restore\_Time properties are checked to ensure that they incremented or advanced correctly.

For IUTs that use Stream Access when performing the AtomicReadFile and AtomicWriteFile services, a Maximum Requested Octet Count (MROC) and a Maximum Write Data Length (MWDL) shall be calculated before starting the test. These values shall be used during the test. MROC shall be 16 less than the minimum of the TD's Max\_APDU\_Length\_Accepted and the IUT's maximum transmittable APDU length. MWDL shall be 21 less than the minimum of the TD's maximum transmittable APDU length and the IUT's Max\_APDU\_Length\_Accepted.

Test Steps:

1. READ DR1 = Database\_Revision

```

2. READ LRT1 = Last_Restore_Time
3. READ OL1 = Object_List
4. REPEAT X = (1 through length of OL1) DO {
    READ NAMES[X] = (OL1[X]), Object_Name
}
5. IF (Protocol_Revision is present and Protocol_Revision ≥ 10) THEN
    IF (Backup_Preparation_Time is present) THEN
        READ BPT = Backup_Preparation_Time
    ELSE
        READ BPT = APDU_Timeout
    IF (Restore_Preparation_Time is present) THEN
        READ RPT = Restore_Preparation_Time
    ELSE
        READ RPT = APDU_Timeout
    IF (Restore_Completion_Time is present) THEN
        READ RCT = Restore_Completion_Time
    ELSE
        READ RCT = APDU_Timeout
    IF (Backup_And_Restore_State is present or Protocol_Revision ≥ 13) THEN
        VERIFY Backup_And_Restore_State = IDLE
6. TRANSMIT ReinitializeDevice-Request,
   'Reinitialized State of Device' = STARTBACKUP,
   'Password' = (any valid password)
7. RECEIVE BACnet-Simple-ACK-PDU BACnet-SimpleACK-PDU
8. IF (Protocol_Revision is present and Protocol_Revision ≥ 10) THEN
    WAIT BPT
    IF (Backup_And_Restore_State is present or Protocol_Revision ≥ 13) THEN
        READ BRSTATE = Backup_And_Restore_State
        READ CF = Configuration_Files
        WHILE (BRSTATE = PREPARING_FOR_BACKUP) DO {
            WAIT 1 second
            READ BRSTATE = Backup_And_Restore_State
            IF (CF is an empty list) THEN
                READ CF = Configuration_Files
            IF (CF is a non-empty list) THEN
                READ X = (the file referenced by Configuration_Files[1]).Name
        }
        CHECK (BRSTATE = PERFORMING_A_BACKUP)
9. READ CF = Configuration_Files
10. CHECK (CF is a non-empty array of BACnetObjectIdentifiers referring to File objects)
11. REPEAT X = (each entry in CF) DO {
    READ Y = X, File_Access_Method
    IF (Y = RECORD_ACCESS) THEN
        WHILE (the last read resulted in an Ack with 'End Of File' == FALSE) DO {
            TRANSMIT AtomicReadFile-Request,
                'Object Identifier' = X,
                'File Start Record' = (the next unread record),
                'Requested Record Count' = 1
            RECEIVE AtomicReadFile-ACK,
                'End Of File' = TRUE | FALSE,
                'File Start Record' = Z,
                'Requested Record Count' = 1
                'Returned Data' = (File contents)
            | Error-PDU -- only acceptable for the first record and only when there are no records in the file
                'Error Class' = SERVICES,
                'Error Code' = INVALID_FILE_START_POSITION
        }
    ELSE
        WHILE (the last read did not indicate 'End Of File') DO {
            TRANSMIT AtomicReadFile-Request,

```

```

        'Object Identifier' = X,
        'File Start Position' = (the next unread octet),
        'Requested Octet Count' = MROC
    RECEIVE AtomicReadFile-ACK,
        'End Of File' = TRUE | FALSE,
        'File Start Position' = (the next unread octet)
        'File Data' = (File contents of length MROC if 'End Of File' is FALSE
            or of length MROC or less if 'End Of File' is TRUE)
    | Error-PDU -- only acceptable for the first record and only when there are no records in the file
        'Error Class' = SERVICES,
        'Error Code' = INVALID_FILE_START_POSITION
    }
}
12. TRANSMIT ReinitializeDevice-Request,
    'Reinitialize State Of Device' = ENDBACKUP,
    'Password' = (any valid password)
13. RECEIVE BACnet-Simple-ACK-PDU BACnet-SimpleACK-PDU
14. VERIFY System_Status != BACKUP_IN_PROGRESS
15. IF (Backup_And_Restore_State is present or (Protocol_Revision is present and Protocol_Revision ≥ 10/13)) THEN
    VERIFY Backup_And_Restore_State = IDLE
16. IF (Database_Revision is changeable) THEN
    MAKE (the configuration in the IUT different, such that the Database_Revision property increments)
    VERIFY Database_Revision <> DR1
    READ DR2 = Database_Revision
    CHECK (DR1 <> DR2)
17. TRANSMIT ReinitializeDevice-Request,
    'Reinitialize State Of Device' = STARTRESTORE,
    'Password' = (any valid password)
18. RECEIVE BACnet-Simple-ACK-PDU BACnet-SimpleACK-PDU
19. IF (Protocol_Revision is present and Protocol_Revision ≥ 10) THEN
    WAIT RPT
    IF (Backup_And_Restore_State is present or Protocol_Revision ≥ 13) THEN
        READ BRSTATE = Backup_And_Restore_State
        WHILE (BRSTATE = PREPARING_FOR_RESTORE) DO {
            WAIT 1 second
            READ BRSTATE = Backup_And_Restore_State
        }
        CHECK (BRSTATE = PERFORMING_A_RESTORE)
20. READ OL2 = Object_List
21. REPEAT X = (entry in CF) DO {
    IF (X is not in OL2) THEN
        TRANSMIT CreateObject-Request
        'Object Identifier' = X
        RECEIVE CreateObject-ACK
        'Object Identifier' = X
    READ FS = X, File_Size
    IF (File_Size is not equal to the size of the backed up file) THEN
        WRITE X, File_Size = 0
    IF (Y = RECORD_ACCESS) THEN
        TRANSMIT AtomicWriteFile-Request
        'File Identifier' = X
        'File Start Record' = 0
        'Record Data' = (file content for first record obtained in step 11)
        RECEIVE AtomicWriteFile-ACK
        'File Start Record' = 0
        REPEAT REC = (each record in the backup of this file) {
            TRANSMIT AtomicWriteFile-Request
            'File Identifier' = X
            'File Start Record' = -1
            'Record Count' = 1

```

```

        'Record Data' = REC
    RECEIVE AtomicWriteFile-ACK
        'File Start Record' = (the record number)
    }
ELSE
    REPEAT Z = (0 through the file size, in increments of MWDL) DO {
        TRANSMIT AtomicWriteFile-Request
            'File Identifier' = X
            'File Start Position' = Z
            'Record Data' = (file contents obtained from the backup, the number of octets
                being the lesser of (file size - Z) and MWDL)
        RECEIVE AtomicWriteFile-ACK
            'File Start Position' = Z
    }
}
22. IF (Backup_And_Restore_State is present or (Protocol_Revision is present and Protocol_Revision ≥ 10/13)) THEN
    VERIFY Backup_And_Restore_State = RESTORE_IN_PROGRESS PERFORMING_A_RESTORE
23. TRANSMIT ReinitializeDevice-Request,
    'Reinitialize State Of Device' = ENDRESTORE,
    'Password' = (any valid password)
24. RECEIVE BACnet-Simple-ACK-PDU BACnet-SimpleACK-PDU
25. IF (Protocol_Revision is present and Protocol_Revision ≥ 10) THEN
    WAIT RCT
    IF (Backup_And_Restore_State is present or Protocol_Revision ≥ 13) THEN
        VERIFY Backup_And_Restore_State = IDLE
26. READ DR3 = Database_Revision
27. CHECK (DR3 <> DR1)
28. IF (Database_Revision was changed in step 16) THEN
    CHECK (DR3 <> DR2)
29. VERIFY Last_Restore_Time > LRT1
30. READ OL3 = Object_List
31. CHECK (that OL1 and OL3 contain the same set of objects)
32. REPEAT X = (1 through length of OL1) DO {
    VERIFY (OL1[X]), Object_Name = NAMES[X]
}

```

[Change **Clause 13.8.1.2**, p. 648]

[Reviewer Note: Changed test to account for optional properties.]

### 13.8.1.2 Attempting a Backup Procedure While Already Performing a Backup Procedure

Purpose: To verify that the IUT correctly rejects a command to start a Backup Procedure when already performing a Backup Procedure.

Test Concept: The IUT is commanded to start a Backup Procedure from one client and then is commanded to start a Backup Procedure from a different client.

Test Steps:

1. IF (Protocol\_Revision is present and Protocol\_Revision ≥ 10) THEN
 

IF (*Backup\_Preparation\_Time is present*) THEN
 READ BPT = Backup\_Preparation\_Time
 ELSE
 READ BPT = *APDU\_Timeout*
2. TRANSMIT ReinitializeDevice-Request,
 'Reinitialized State of Device' = STARTBACKUP,
 'Property Identifier' = (any valid password)



3. RECEIVE ~~Simple-ACK-PDU~~BACnet-SimpleACK-PDU
4. IF (Protocol\_Revision is present and Protocol\_Revision  $\geq$  10) THEN  
WAIT BPT
5. TRANSMIT  
SNET = (N, any remote network number),  
SADR = (M, any MAC address valid for the specified network),  
ReinitializeDevice-Request,  
'Reinitialized State of Device' = STARTBACKUP,  
'Property Identifier' = (any valid password),
6. RECEIVE  
DNET = N,  
DADR = M,  
BACnet-Error PDU,  
'Error Class' = DEVICE,  
'Error Code' = CONFIGURATION\_IN\_PROGRESS
7. TRANSMIT ReinitializeDevice-Request,  
'Reinitialized State of Device' = ENDBACKUP,  
'Property Identifier' = (any valid password)
8. RECEIVE ~~Simple-ACK-PDU~~BACnet-SimpleACK-PDU

[Change **Clause 13.8.1.3**, p. 649]

[Reviewer Note: Changed test to account for optional properties.]

### 13.8.1.3 Attempting a Backup Procedure While Already Performing a Restore Procedure

Purpose: To verify that the IUT correctly rejects a command to start a Backup Procedure when already performing a Restore Procedure.

*Notes to Tester: After an incomplete restore attempt, the IUT may revert to a default configuration or another state that is different from the IUT state when this test was started.*

Test Steps:

1. IF (Protocol\_Revision is present and Protocol\_Revision  $\geq$  10) THEN  
IF (Restore\_Preparation\_Time is present) THEN  
READ RPT = Restore\_Preparation\_Time  
ELSE  
READ RPT = APDU\_Timeout  
IF (Restore\_Completion\_Time is present) THEN  
READ RCT = Restore\_Completion\_Time  
ELSE  
READ RCT = APDU\_Timeout
2. TRANSMIT ReinitializeDevice-Request,  
'Reinitialized State of Device' = STARTRESTORE,  
'Property Identifier' = (any valid password)
3. RECEIVE ~~Simple-ACK-PDU~~BACnet-SimpleACK-PDU
4. IF (Protocol\_Revision is present and Protocol\_Revision  $\geq$  10) THEN  
WAIT RPT
5. TRANSMIT ReinitializeDevice-Request,  
'Reinitialized State of Device' = STARTBACKUP,  
'Property Identifier' = (any valid password),
6. RECEIVE BACnet-Error PDU,  
'Error Class' = DEVICE,  
'Error Code' = CONFIGURATION\_IN\_PROGRESS
7. TRANSMIT ReinitializeDevice-Request,  
'Reinitialized State of Device' = ABORTRESTORE,  
'Property Identifier' = (any valid password)

8. RECEIVE ~~Simple-ACK-PDU~~BACnet-SimpleACK-PDU
9. IF (Protocol\_Revision is present and Protocol\_Revision  $\geq$  10) THEN  
WAIT RCT

~~Notes to Tester: After an incomplete restore attempt, the IUT may revert to a default configuration or another state that is different from the IUT state when this test was started.~~

[Change **Clause 13.8.1.4**, p. 649]

[Reviewer Note: Changed test to account for optional properties.]

#### 13.8.1.4 Attempting a Restore Procedure While Already Performing a Backup Procedure

Purpose: To verify that the IUT correctly rejects a command to start a Restore Procedure when already performing a Backup Procedure.

Test Concept: The IUT is commanded to start a Restore Procedure from one client and then is commanded to start a Restore Procedure from a different client.

Test Steps:

1. IF (Protocol\_Revision is present and Protocol\_Revision  $\geq$  10) THEN  
    IF (*Backup\_Preparation\_Time is present*) THEN  
        READ BPT = Backup\_Preparation\_Time  
    ELSE  
        READ BPT = APDU\_Timeout
2. TRANSMIT ReinitializeDevice-Request,  
    'Reinitialized State of Device' = STARTBACKUP,  
    'Property Identifier' = (any valid password)
3. RECEIVE ~~Simple-ACK-PDU~~BACnet-SimpleACK-PDU
4. IF (Protocol\_Revision is present and Protocol\_Revision  $\geq$  10) THEN  
    WAIT BPT
5. TRANSMIT  
    SNET = (N, any remote network number),  
    SADR = (M, any MAC address valid for the specified network),  
    ReinitializeDevice-Request,  
    'Reinitialized State of Device' = STARTRESTORE,  
    'Property Identifier' = (any valid password),
6. RECEIVE  
    DNET = N,  
    DADR = M,  
    BACnet-Error PDU,  
    'Error Class' = DEVICE,  
    'Error Code' = CONFIGURATION\_IN\_PROGRESS
7. TRANSMIT ReinitializeDevice-Request,  
    'Reinitialized State of Device' = ENDBACKUP,  
    'Property Identifier' = (any valid password)
8. RECEIVE ~~Simple-ACK-PDU~~BACnet-SimpleACK-PDU

[Change **Clause 13.8.1.5**, p. 650]

[Reviewer Note: Changed test to account for optional properties.]

### 13.8.1.5 Attempting a Restore Procedure While Already Performing a Restore Procedure

Purpose: To verify that the IUT correctly rejects a command to start a Restore Procedure when already performing a Restore Procedure.

*Notes to Tester: After an incomplete restore attempt, the IUT may revert to a default configuration or another state that is different from the IUT state when this test was started.*

Test Steps:

1. IF (Protocol\_Revision is present and Protocol\_Revision  $\geq$  10) THEN  
     IF (Restore\_Preparation\_Time is present) THEN  
         READ RPT = Restore\_Preparation\_Time  
     ELSE  
         READ RPT = APDU\_Timeout  
     IF (Restore\_Completion\_Time is present) THEN  
         READ RCT = Restore\_Completion\_Time  
     ELSE  
         READ RCT = APDU\_Timeout
2. TRANSMIT ReinitializeDevice-Request,  
     'Reinitialized State of Device' = STARTRESTORE,  
     'Property Identifier' = (any valid password)
3. RECEIVE ~~Simple-ACK-PDU~~BACnet-SimpleACK-PDU
4. IF (Protocol\_Revision is present and Protocol\_Revision  $\geq$  10) THEN  
     WAIT RPT
5. TRANSMIT ReinitializeDevice-Request,  
     'Reinitialized State of Device' = STARTRESTORE,  
     'Property Identifier' = (any valid password),
6. RECEIVE BACnet-Error PDU,  
     'Error Class' = DEVICE,  
     'Error Code' = CONFIGURATION\_IN\_PROGRESS
7. TRANSMIT ReinitializeDevice-Request,  
     'Reinitialized State of Device' = ABORTRESTORE,  
     'Property Identifier' = (any valid password)
8. RECEIVE ~~Simple-ACK-PDU~~BACnet-SimpleACK-PDU
9. IF (Protocol\_Revision is present and Protocol\_Revision  $\geq$  10) THEN  
     WAIT RCT

~~Notes to Tester: After an incomplete restore attempt, the IUT may revert to a default configuration or another state that is different from the IUT state when this test was started.~~

[Change **Clause 13.8.1.6**, p. 650]

[Reviewer Note: Modified to consider the protocol revision for WAITs. Changed test to account for optional properties.]

### 13.8.1.6 Ending Backup and Restore Procedures via Timeout

Purpose: This test case verifies that the IUT will end Backup and Restore procedures after not receiving any messages related to the backup or restore for longer than Backup\_Failure\_Timeout and that the Backup\_Failure\_Timeout property is writeable.

*Notes to Tester: After an incomplete restore attempt, the IUT may revert to a default configuration or another state that is different from the IUT state when this test was started.*

Test Steps:

1. WRITE Backup\_Failure\_Timeout = (A value T1 greater than Backup\_Preparation\_Timeout)
2. VERIFY Backup\_Failure\_Timeout = T1
3. IF (Protocol\_Revision is present and Protocol\_Revision  $\geq$  10) THEN  
     IF (Backup\_Preparation\_Time is present) THEN

```

        READ BPT = Backup_Preparation_Time
    ELSE
        READ BPT = APDU_Timeout
4.  TRANSMIT ReinitializeDevice-Request,
    'Reinitialized State of Device' = STARTBACKUP,
    'Property Identifier' = (any valid password)
5.  RECEIVE Simple-ACK-PDUBACnet-SimpleACK-PDU
6.  IF (Protocol_Revision is present and Protocol_Revision ≥ 10) THEN
    WAIT BPT
    IF (Backup_And_Restore_State is present or Protocol_Revision ≥ 13) THEN
        READ BRSTATE = Backup_And_Restore_State
        WHILE (BRSTATE = PREPARING_FOR_BACKUP) DO {
            WAIT 1 second
            READ BRSTATE = Backup_And_Restore_State
        }
        CHECK (BRSTATE = PERFORMING_A_BACKUP)
7.  WAIT (T1 + 10 seconds)
8.  IF (Backup_And_Restore_State is present or (Protocol_Revision is present and Protocol_Revision ≥ 1013)) THEN
    VERIFY Backup_And_Restore_State = IDLE
9.  VERIFY System_Status != BACKUP_IN_PROGRESS
10. IF (Protocol_Revision is present and Protocol_Revision ≥ 10) THEN
    IF (Restore_Preparation_Time is present) THEN
        READ RPT = Restore_Preparation_Time
    ELSE
        READ RPT = APDU_Timeout
    IF (Restore_Completion_Time is present) THEN
        READ RCT = Restore_Completion_Time
    ELSE
        READ RCT = APDU_Timeout
11. TRANSMIT ReinitializeDevice-Request,
    'Reinitialized State of Device' = STARTRESTORE,
    'Password' = (any valid password)
12. RECEIVE BACnet-Simple ACK-PDU
13. IF (Protocol_Revision is present and Protocol_Revision ≥ 10) THEN
    WAIT RPT
    IF (Backup_And_Restore_State is present or Protocol_Revision ≥ 13) THEN
        READ BRSTATE = Backup_And_Restore_State
        WHILE (BRSTATE = PREPARING_FOR_RESTORE) DO {
            WAIT 1 second
            READ BRSTATE = Backup_And_Restore_State
        }
        CHECK (BRSTATE = PERFORMING_A_RESTORE)
    ELSE
        WAIT (30 seconds)
14. WAIT (T1 + 10 40-seconds)
15. IF (Protocol_Revision is present and Protocol_Revision ≥ 10) THEN
    WAIT RCT
    IF (Backup_And_Restore_State is present or Protocol_Revision ≥ 13) THEN
        VERIFY Backup_And_Restore_State = IDLE
16. VERIFY System_Status != DOWNLOAD_IN_PROGRESS

```

~~Notes to Tester: After an incomplete restore attempt, the IUT may revert to a default configuration or another state that is different from the IUT state when this test was started.~~

[Change **Clause 13.8.1.7**, p. 651]

[Reviewer Note: Changed test to account for optional properties.]

### 13.8.1.7 Ending Backup and Restore Procedures via Abort

Purpose: This test case verifies that the IUT will leave the BACKUP\_IN\_PROGRESS and DOWNLOAD\_IN\_PROGRESS states upon a command to abort.

*Notes to Tester: After an incomplete restore attempt, the IUT may revert to a default configuration or another state that is different from the IUT state when this test was started.*

Test Steps:

1. IF (Protocol\_Revision is present and Protocol\_Revision  $\geq$  10) THEN  
     IF (Backup\_Preparation\_Time is present) THEN  
         READ BPT = Backup\_Preparation\_Time  
     ELSE  
         READ BPT = APDU\_Timeout
2. TRANSMIT ReinitializeDevice-Request,  
     'Reinitialized State of Device' = STARTBACKUP,  
     'Password' = (any valid password)
3. RECEIVE BACnet-Simple ACK-PDU
4. IF (Protocol\_Revision is present and Protocol\_Revision  $\geq$  10) THEN  
     WAIT BPT
5. TRANSMIT ReinitializeDevice-Request,  
     'Reinitialized State of Device' = ENDBACKUP,  
     'Password' = (any valid password)
6. RECEIVE BACnet-Simple ACK-PDU
7. IF (Backup\_And\_Restore\_State is present or (Protocol\_Revision is present and Protocol\_Revision  $\geq$  10/13)) THEN  
     VERIFY Backup\_And\_Restore\_State = IDLE
8. VERIFY System\_Status != BACKUP\_IN\_PROGRESS
9. IF (Protocol\_Revision is present and Protocol\_Revision  $\geq$  10) THEN  
     IF (Restore\_Preparation\_Time is present) THEN  
         READ RPT = Restore\_Preparation\_Time  
     ELSE  
         READ RPT = APDU\_Timeout  
     IF (Restore\_Completion\_Time is present) THEN  
         READ RCT = Restore\_Completion\_Time  
     ELSE  
         READ RCT = APDU\_Timeout
10. TRANSMIT ReinitializeDevice-Request,  
     'Reinitialized State of Device' = STARTRESTORE,  
     'Password' = (any valid password)
11. RECEIVE BACnet-Simple ACK-PDU
12. IF (Protocol\_Revision is present and Protocol\_Revision  $\geq$  10) THEN  
     WAIT RPT
13. TRANSMIT ReinitializeDevice-Request,  
     'Reinitialized State of Device' = ABORTRESTORE,  
     'Password' = (any valid password)
14. RECEIVE BACnet-Simple ACK-PDU
15. IF (Protocol\_Revision is present and Protocol\_Revision  $\geq$  10) THEN  
     WAIT RCT  
     IF (Backup\_And\_Restore\_State is present or Protocol\_Revision  $\geq$  13) THEN  
         VERIFY Backup\_And\_Restore\_State = IDLE
16. VERIFY System\_Status != DOWNLOAD\_IN\_PROGRESS

~~Notes to Tester: After an incomplete restore attempt, the IUT may revert to a default configuration or another state that is different from the IUT state when this test was started.~~

[Change **Clause 13.8.1.8**, p. 652]

[Reviewer Note: Added supported error codes.]

### 13.8.1.8 Attempting a Backup Procedure with an Invalid Password

Purpose: To verify the correct execution of the Backup procedure when an invalid password is provided *and when a password is required but no password is provided*. If the IUT cannot be made to deny a ReinitializeDevice <STARTBACKUP> service request that does not contain a valid password, then this test shall be omitted.

Test Steps:

1. TRANSMIT ReinitializeDevice-Request,  
    'Reinitialized State of Device' = STARTBACKUP,  
    'Password' = (any invalid password)
2. IF (Protocol\_Revision is present and Protocol\_Revision >= 7) THEN  
    RECEIVE BACnet-Error-PDU,  
    'Error Class' = SECURITY,  
    'Error Code' = PASSWORD\_FAILURE  
ELSE  
    (RECEIVE BACnet-Error-PDU,  
    'Error Class' = SECURITY,  
    'Error Code' = PASSWORD\_FAILURE) |  
    (RECEIVE BACnet-Error-PDU,  
    'Error Class' = SERVICES,  
    'Error Code' = SERVICE\_REQUEST\_DENIED) |
3. TRANSMIT ReinitializeDevice-Request,  
    'Reinitialized State of Device' = STARTBACKUP
4. IF (Protocol\_Revision is present and Protocol\_Revision >= 7) THEN  
    RECEIVE BACnet-Error-PDU,  
    'Error Class' = SECURITY,  
    'Error Code' = PASSWORD\_FAILURE  
ELSE  
    (RECEIVE BACnet-Error-PDU,  
    'Error Class' = SECURITY,  
    'Error Code' = PASSWORD\_FAILURE) |  
    (RECEIVE BACnet-Error-PDU,  
    'Error Class' = SERVICES,  
    'Error Code' = SERVICE\_REQUEST\_DENIED) |

[Change **Clause 13.8.1.9**, p. 652]

[Reviewer Note: Added supported error codes.]

### 13.8.1.9 Attempting a Restore Procedure with an Invalid Password

Purpose: To verify the correct execution of the Restore procedure when an invalid password is provided *and when a password is required but no password is provided*. If the IUT cannot be made to deny a ReinitializeDevice <STARTRESTORE> service request that does not contain a valid password, then this test shall be omitted.

Test Steps:

1. TRANSMIT ReinitializeDevice-Request,  
    'Reinitialized State of Device' = STARTRESTORE,  
    'Password' = (any invalid password)
2. IF (Protocol\_Revision is present and Protocol\_Revision >= 7) THEN  
    RECEIVE BACnet-Error-PDU,  
    'Error Class' = SECURITY,  
    'Error Code' = PASSWORD\_FAILURE  
ELSE  
    (RECEIVE BACnet-Error-PDU,  
    'Error Class' = SECURITY,  
    'Error Code' = PASSWORD\_FAILURE) |  
    (RECEIVE BACnet-Error-PDU,  
    'Error Class' = SERVICES,  
    'Error Code' = SERVICE\_REQUEST\_DENIED) |

- ```

        'Error Class' = SERVICES,
        'Error Code' = SERVICE_REQUEST_DENIED) |
3. TRANSMIT ReinitializeDevice-Request,
   'Reinitialized State of Device' = STARTRESTORE
4. IF (Protocol_Revision is present and Protocol_Revision >= 7) THEN
   RECEIVE BACnet-Error-PDU,
   'Error Class' = SECURITY,
   'Error Code' = PASSWORD_FAILURE
ELSE
   (RECEIVE BACnet-Error-PDU,
   'Error Class' = SECURITY,
   'Error Code' = PASSWORD_FAILURE) |
   (RECEIVE BACnet-Error-PDU,
   'Error Class' = SERVICES,
   'Error Code' = SERVICE_REQUEST_DENIED)

```

[Change **Clause 13.8.1.10**, p. 653]

[Reviewer Note: Changed test to account for optional properties.]

### 13.8.1.10 Starting and Ending a Backup Procedure when a Password is not Required

Purpose: This test case verifies that the IUT ignores the password. If the IUT cannot be made to accept a ReinitializeDevice service request that contains any or no password, then this test shall be omitted.

Test Steps:

1. IF (Protocol\_Revision is present and Protocol\_Revision  $\geq$  10) THEN  
     IF (Backup\_Preparation\_Time is present) THEN  
         READ BPT = Backup\_Preparation\_Time  
     ELSE  
         READ BPT = APDU\_Timeout
2. TRANSMIT ReinitializeDevice-Request,  
     'Reinitialized State of Device' = STARTBACKUP,  
     'Password' = (any non-zero length password)
3. RECEIVE BACnet-Simple ACK-PDU
4. IF (Protocol\_Revision is present and Protocol\_Revision  $\geq$  10) THEN  
     WAIT BPT
5. TRANSMIT ReinitializeDevice-Request,  
     'Reinitialized State of Device' = ENDBACKUP,  
     'Password' = (any non-zero length password)
6. RECEIVE BACnet-Simple ACK-PDU
7. IF (Backup\_And\_Restore\_State is present or (Protocol\_Revision is present and Protocol\_Revision  $\geq$  ~~40~~13)) THEN  
     VERIFY Backup\_And\_Restore\_State = IDLE
8. VERIFY System\_Status != BACKUP\_IN\_PROGRESS

[Change **Clause 13.8.1.11**, p. 653]

[Reviewer Note: Corrected the 'Reinitialized State of Device' value in step 5. Changed test to account for optional properties.]

### 13.8.1.11 Starting and Ending a Restore Procedure when a Password is not Required

Purpose: This test case verifies that the IUT ignores the password. If the IUT cannot be made to accept a ReinitializeDevice service request that contains any or no password, then this test shall be omitted.

Test Steps:

1. IF (Protocol\_Revision is present and Protocol\_Revision  $\geq$  10) THEN  
     IF (Restore\_Preparation\_Time is present) THEN  
         READ RPT = Restore\_Preparation\_Time  
     ELSE  
         READ RPT = APDU\_Timeout  
     IF (Restore\_Completion\_Time is present) THEN  
         READ RCT = Restore\_Completion\_Time  
     ELSE  
         READ RCT = APDU\_Timeout
2. TRANSMIT ReinitializeDevice-Request,  
     'Reinitialized State of Device' = STARTRESTORE,  
     'Password' = (any non-zero length password)
3. RECEIVE BACnet-Simple ACK-PDU
4. IF (Protocol\_Revision is present and Protocol\_Revision  $\geq$  10) THEN  
     WAIT RPT
5. TRANSMIT ReinitializeDevice-Request,  
     'Reinitialized State of Device' = ~~END~~ABORTRESTORE,  
     'Password' = (any non-zero length password)
6. RECEIVE BACnet-Simple ACK-PDU
7. IF (Protocol\_Revision is present and Protocol\_Revision  $\geq$  10) THEN  
     WAIT RCT
8. VERIFY System\_Status != DOWNLOAD\_IN\_PROGRESS

[Change **Clause 13.8.1.12**, p. 653]

[Reviewer Note: Changed test to account for optional properties.]

### 13.8.1.12 System\_Status during a Backup Procedure

Purpose: This test case verifies that the IUT correctly sets its System\_Status during a Backup procedure. If the IUT does not change its operational behavior during a Backup Procedure, then this test shall be omitted.

Test Steps:

1. IF (Protocol\_Revision is present and Protocol\_Revision  $\geq$  10) THEN  
     IF (Backup\_Preparation\_Time is present) THEN  
         READ BPT = Backup\_Preparation\_Time  
     ELSE  
         READ BPT = APDU\_Timeout
2. TRANSMIT ReinitializeDevice-Request,  
     'Reinitialized State of Device' = STARTBACKUP,  
     'Password' = (any valid password)
3. RECEIVE BACnet-Simple ACK-PDU
4. IF (Protocol\_Revision is present and Protocol\_Revision  $\geq$  10) THEN  
     WAIT BPT
5. VERIFY System\_Status = BACKUP\_IN\_PROGRESS
6. TRANSMIT ReinitializeDevice-Request,  
     'Reinitialized State of Device' = ENDBACKUP,  
     'Password' = (any valid password)
7. RECEIVE BACnet-Simple ACK-PDU
8. WAIT a vendor specified period of time for the device to complete the backup operation
9. VERIFY System\_Status != BACKUP\_IN\_PROGRESS

[Change **Clause 13.8.1.13**, p. 654]

[Reviewer Note: Changed test to account for optional properties.]



### 13.8.1.13 System\_Status during a Restore Procedure

Purpose: This test case verifies that the IUT correctly sets its System\_Status during a Restore procedure. If the IUT does not change its operational behavior during a Restore Procedure, this test shall be omitted.

Test Steps:

1. IF (Protocol\_Revision is present and Protocol\_Revision  $\geq$  10) THEN  
     IF (Restore\_Preparation\_Time is present) THEN  
         READ RPT = Restore\_Preparation\_Time  
     ELSE  
         READ RPT = APDU\_Timeout  
     IF (Restore\_Completion\_Time is present) THEN  
         READ RCT = Restore\_Completion\_Time  
     ELSE  
         READ RCT = APDU\_Timeout
2. TRANSMIT ReinitializeDevice-Request,  
     'Reinitialized State of Device' = STARTRESTORE,  
     'Password' = (any valid password)
3. RECEIVE BACnet-Simple ACK-PDU
4. IF (Protocol\_Revision is present and Protocol\_Revision  $\geq$  10) THEN  
     WAIT RPT
5. VERIFY System\_Status = DOWNLOAD\_IN\_PROGRESS
6. TRANSMIT ReinitializeDevice-Request,  
     'Reinitialized State of Device' = ABORTRESTORE,  
     'Password' = (any valid password)
7. RECEIVE BACnet-Simple ACK-PDU
8. IF (Protocol\_Revision is present and Protocol\_Revision  $\geq$  10) THEN  
     WAIT RCT
9. VERIFY System\_Status != DOWNLOAD\_IN\_PROGRESS

## 13.8.2 Backup and Restore Initiation Tests

[Change Clause 13.8.2.1, p. 654]

[Reviewer Note: Added note about preparation time properties. Clarified test characteristics for backup file names.]

### 13.8.2.1 Initiate a Full Backup and Restore

Purpose: To verify that the IUT can perform a Backup and Restore on a BACnet server device.

Test Concept: The IUT is first made to initiate a Backup and then a Restore of the TD device. This test verifies that the IUT performs the Backup procedure correctly by comparing the resulting restored file with the original. The TD is made to respond appropriately such that the Backup and Restore procedures are completed normally. The final check can be accomplished using a file compare of the original files to the files restored or by comparing the network traffic during the backup to the network traffic during the restore. The number of files, the order of the files, and the file content should be the same. The test is to be executed multiple times with the TD configured with different sets of backup and restore characteristics.

Configuration Requirements: The IUT is configured to already contain a device binding for the TD device. The TD is configured with some of the following characteristics:

Backup Characteristics:

1. The TD is configured to contain an APDU size that is smaller than the APDU size of the IUT. If the TD and the IUT support segmentation, the TD is configured to support a smaller window size than the IUT.
2. The TD is configured to contain a configuration file of size zero.

3. The TD is configured to contain some configuration files that are STREAM\_ACCESS and some that are RECORD\_ACCESS.
4. The TD is configured to only allow access to File and Device objects during the Backup and Restore procedures. All other attempts shall result in an error from the TD.
5. The TD is configured to require the same password for all of the reinitialize device requests.
6. The TD is configured to contain *characters in the object name of some file name objects, such as \* " and \*, that would reveal weakness in the implementation process that assigns names to files where the backup is stored ~~not be accepted~~ by the OS which the IUT is running on.
7. The TD is configured with a Protocol\_Revision < 10.
8. The TD is configured with a Protocol\_Revision  $\geq 10$ . This is only used if the IUT claims Protocol\_Revision  $\geq 10$ .

*Note that if IUT claims Protocol\_Revision < 10, the presence of preparation time properties in a TD with Protocol\_Revision  $\geq 10$  may be ignored and cannot be relied upon.*

#### Restore Characteristics:

1. The TD is configured to support CreateObject service, and some of the configuration files exist while others do not.
2. The TD is configured such that some of the configuration file File objects exist, but the file size is different from that of the file to be restored.
3. The TD is configured to not support the CreateObject service.
4. The TD is configured to contain some configuration files that are STREAM\_ACCESS and some that are RECORD\_ACCESS.
5. The TD is configured to only allow access to File and Device objects during the Backup and Restore procedures. All other attempts shall result in an error from the TD.
6. The TD is configured to require the same password for all of the reinitialize device requests.
7. The TD is configured with a Protocol\_Revision < 10.
8. The TD is configured with a Protocol\_Revision  $\geq 10$ . This is only used if the IUT claims Protocol\_Revision  $\geq 10$ .

*Note that if IUT claims Protocol\_Revision < 10, the presence of preparation time properties in a TD with Protocol\_Revision  $\geq 10$  may be ignored and cannot be relied upon.*

*Notes to Tester: Other items to ensure were correct during execution of the test:*

1. Verify the order the IUT read the configuration files was the same as the order returned by the Configuration\_Files property.
2. Verify that any file with a File\_Size of zero was restored.
3. Verify that each file read is in byte order if STREAM\_ACCESS and in record order if RECORD\_ACCESS.

#### Test Steps:

1. MAKE (IUT initiate a backup on the TD device)
2. WAIT (for backup to complete)
3. MAKE (changes required in TD to meet restore characteristics for this test)
4. MAKE (IUT initiate a restore on the TD device)
5. WAIT (for restore to complete)
6. CHECK (that the file content restored is the same as the file content that was backed up)

~~Notes to Tester: Other items to ensure were correct during execution of the test:~~

- ~~1. Verify the order the IUT read the configuration files was the same as the order returned by the Configuration\_Files property.~~
- ~~2. Verify that any file with a File\_Size of zero was restored.~~
- ~~3. Verify that each file read is in byte order if STREAM\_ACCESS and in record order if RECORD\_ACCESS.~~

### 13.9 Application State Machine Tests

[Add new **Clause 13.9.X1**, p. 658]

### 13.9.X1 Ignore Confirmed Broadcast Requests

Purpose: This test case verifies that the IUT will quietly discard any Confirmed-Request-PDU, whose destination address is a multicast or broadcast address, received from the network layer.

Test Concept: The TD transmits the Confirmed-Request-PDU services whose destination address is a multicast or broadcast address. The IUT is required to silently drop the requests because it should only respond to unicast confirmed requests.

Test Steps:

1. TRANSMIT Any BACnet-Confirmed-Request-PDU,  
DESTINATION = LOCAL BROADCAST | GLOBAL BROADCAST,
2. CHECK (that the IUT does not send any packets in response to above Confirmed-Request-PDU)

## 13.10 Workstation Scheduling Tests

### 13.10.4 Modify and Exception\_Schedule

[Change Clause 13.10.4.9, p. 671]

[Reviewer Note: Modified to include new WeekOfMonth values defined in PR 18.]

#### 13.10.4.9 Modify an Exception\_Schedule by Adding a BACnetSpecialEvent with Period of Choice calendarEntry of Choice WeekNDay

Purpose: Demonstrate that the IUT can accept user input and modify the Exception\_Schedule by adding a BACnetSpecialEvent of the schedule with the calendarEntry of type WeekNDay.

Test Concept: For different WeekNDay values, add a BACnetCalendarEntry to the Date\_List property of an Exception\_Schedule, and verify the IUT writes the modified Date\_List correctly to the reference device.

Test Configuration: A reference device contains reference objects S1, C1, and C2. Choose the value set to use from the table below based on the protocol-revision for the IUT.

		WeekNDay = <Month><weekOfMonth><dayOfWeek>		
Value Set	Protocol_Revision	Month	weekOfMonth	dayOfWeek
1	PR < 4	0 to 12 and FF	1 to 6 and FF	1 to 7 and FF
2	PR ≥ 4	0 to 12 and FF and 13 and 14	1 to 6 and FF	1 to 7 and FF
3	PR ≥ 18	0 to 12 and FF and 13 and 14	1 to 9 and FF	1 to 7 and FF

Test Steps:

REPEAT WND = (WeekNDay Value Set from table)

- ```
{
  1. MAKE (the IUT add an entire BACnetSpecialEvent with period of choice calendarEntry of choice
    WeekNDay with value of WND to the Exception_Schedule of S1)
  2. CHECK (Did the IUT write the modified Exception_Schedule correctly?)
}
```

Notes to Tester: For example, try different WeekNDay representations that leave one or more of each of the following fields unspecified: month, weekOfMonth, dayOfWeek. If the IUT supports devices with Protocol\_Revision ≥ 4, try using the special values for month 13 (all odd) and 14 (all even).

### 13.10.5 Modify a Calendar Object

[Change **Clause 13.10.5.4**, p. 674]

[Reviewer Note: Modified to include new WeekOfMonth values defined in PR 18.]

#### 13.10.5.4 Modify a Calendar by Adding a BACnetCalendarEntry of Choice WeekNDay to the Date\_List

Purpose: Demonstrate that the IUT can accept user input and use it to add a BACnetCalendarEntry of choice WeekNDay to the Date\_List.

*Test Concept: For different WeekNDay values, add a BACnetCalendarEntry to the Date\_List property of a Calendar, and verify the IUT writes the modified Date\_List correctly to the reference device.*

*Test Configuration: A reference device contains reference object C1. Choose the value set to use from the table below based on the protocol-revision for the IUT.*

|                  |                          | <i>WeekNDay = &lt;Month&gt;&lt;weekOfMonth&gt;&lt;dayOfWeek&gt;</i> |                      |                      |
|------------------|--------------------------|---------------------------------------------------------------------|----------------------|----------------------|
| <i>Value Set</i> | <i>Protocol_Revision</i> | <i>Month</i>                                                        | <i>weekOfMonth</i>   | <i>dayOfWeek</i>     |
| <i>1</i>         | <i>PR &lt; 4</i>         | <i>0 to 12 and FF</i>                                               | <i>1 to 6 and FF</i> | <i>1 to 7 and FF</i> |
| <i>2</i>         | <i>PR ≥ 4</i>            | <i>0 to 12 and FF and 13 and 14</i>                                 | <i>1 to 6 and FF</i> | <i>1 to 7 and FF</i> |
| <i>3</i>         | <i>PR ≥ 18</i>           | <i>0 to 12 and FF and 13 and 14</i>                                 | <i>1 to 9 and FF</i> | <i>1 to 7 and FF</i> |

Test Steps:

*REPEAT WND = (WeekNDay Value Set from table)*

- ```
{
  1. MAKE (the IUT add a BACnetCalendarEntry of type WeekNDay with value of WND to the Date_List of C1)
  2. CHECK (Did the IUT write the modified Date_List correctly?)
}
```

~~Notes to Tester: For example, try different WeekNDay representations that leave one or more of each of the following fields unspecified: month, weekOfMonth, dayOfWeek. If the IUT supports devices with Protocol\_Revision ≥ 4, try using the special month values 13 (all odd) and 14 (all even)~~

[Add new **Clause 13.10.X8**, p. 675]

#### 13.10.X8 Modify a Self-inconsistent Timer to be Consistent

Purpose: Demonstrate that the IUT can read a Timer that is self-inconsistent with regard to datatype and modify it to be consistent. This capability is required in order to be able to fix Timer objects that may be in a self-inconsistent state when the process of changing datatype in an earlier attempt was interrupted.

Test Configuration: A reference device contains any valid Timer object T that supports modifying the datatype.

Notes to Tester: A consistent timer is one that meets all of these criteria:

1. All non-NULL values used in the State\_Change\_Values property shall be of the same datatype.
2. All properties referenced by the List\_Of\_Object\_Property\_References are writable with that datatype.

Until those conditions are met, the inconsistency can be detected by reading the Reliability property which will have the value CONFIGURATION\_ERROR.

Test Steps:

1. MAKE (the IUT modify T so that the datatype which State\_Change\_Values holds, and which List\_Of\_Object\_Property\_References values points to, is consistent)
2. CHECK (Did the IUT write a consistent timer?)

[Add new **Clause 13.10.X9**, p. 675]

### **13.10.X9 Change the Datatype that a Timer Object References**

Purpose: Verify that the IUT can alter the datatype of a Timer object.

Test Configuration: A reference device contains any valid Timer object that supports modifying the datatype.

Notes to Tester: It is acceptable that the IUT modify the Timer one property at a time, so there may be a time period when the Timer is in a self-inconsistent state during reconfiguration.

Test Steps:

1. MAKE (the State\_Change\_Values have, and/or List\_Of\_Object\_Property\_References point to a scheduled datatype that is different)
2. CHECK (Did the IUT write the modified properties correctly?)

## 135.1-2019s-14 Apply miscellaneous editorial changes

### Rationale

Several editorial changes are applied:

The reference to the BACnet standard is updated to ANSI/ASHRAE Standard 135-2020.

The Dependencies and BACnet Reference Clause(s) sections have been identified as not valuable. Both sections are removed to reduce maintenance.

The position of the Notes to Tester section at end of the test bears the risk of testers missing this important instruction. This section is repositioned within tests specifications.

[Change **Clause 6.2.8**, p.16]

### 6.2.9 WAIT Statement

The WAIT statement is used to pause the execution of the TD for some specified amount of time.

<wait statement> ::= WAIT <timer value>

Test Steps: The TD shall pause the amount of time specified by the <timer value> before proceeding to the next test step. The <timer value> shall be one of the timers specified in 6.3 of this standard, in ANSI/ASHRAE 135-~~2008~~2020, or as otherwise specified.

Example: WAIT **Internal Processing Fail Time**

[Change **Clause 6.2.11**, p.17]

### 6.2.11 BEFORE Statement

The BEFORE statement is used to test for the occurrence of an expected action before a timer expires.

<before statement> ::= BEFORE <timer> <statement>

The <timer value> shall be one of the timers specified in 6.3 of this standard, in ANSI/ASHRAE 135-~~2008~~2020, or as otherwise specified. If the action indicated by <statement> has not yet occurred when the timer expires the test fails. Otherwise this test step passes and the test continues. For example:

Example: BEFORE **Acknowledgment Fail Time** RECEIVE BACnet-SimpleACK-PDU

[Change **Clause 6.4**, p.19]

### 6.4 BACnet References

All references to BACnet clauses in this standard refer to ANSI/ASHRAE 135-~~2008~~2020, except as otherwise noted ("BACnet-2001" refers to ANSI/ASHRAE 135-2001).

[Change **Clauses 7 – 14.9.3**, p. 20 - 703]

[Reviewer Note: In Clauses 7 through 14, remove the references to clauses of the BACnet standard.]

~~BACnet 2001 Reference Clause:~~

~~BACnet 2001 Reference Clauses:~~

~~BACnet Reference Clause:~~

~~BACnet Reference Clauses:~~

[Change **Clauses 7 – 14.9.3**, p. 20 - 703]

[Reviewer Note: In Clauses 7 through 14, remove the Dependencies section.]

~~Dependencies:~~

[Change **Clauses 7 – 14.9.3**, p. 20 - 703]

[Reviewer Note: In Clauses 7 through 14, rename “Note to Tester”, “Note to tester”, and “Notes to tester” to “Notes to Tester”. Move the Notes to Tester section from after the Test Steps to immediately before the Test Steps. Example: 135.1-2019s-7, Clause 7.3.2.9.3]

## **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 · 180 Technology Parkway · Peachtree Corners, GA 30092 · [www.ashrae.org](http://www.ashrae.org)**

## **About ASHRAE**

Founded in 1894, ASHRAE is a global professional society committed to serve humanity by advancing the arts and sciences of heating, ventilation, air conditioning, refrigeration, and their allied fields.

As an industry leader in research, standards writing, publishing, certification, and continuing education, ASHRAE and its members are dedicated to promoting a healthy and sustainable built environment for all, through strategic partnerships with organizations in the HVAC&R community and across related industries.

To stay current with this and other ASHRAE Standards and Guidelines, visit [www.ashrae.org/standards](http://www.ashrae.org/standards), and connect on LinkedIn, Facebook, Twitter, and YouTube.

## **Visit the ASHRAE Bookstore**

ASHRAE offers its Standards and Guidelines in print, as immediately downloadable PDFs, and via ASHRAE Digital Collections, which provides online access with automatic updates as well as historical versions of publications. Selected Standards and Guidelines are also offered in redline versions that indicate the changes made between the active Standard or Guideline and its previous version. For more information, visit the Standards and Guidelines section of the ASHRAE Bookstore at [www.ashrae.org/bookstore](http://www.ashrae.org/bookstore).

## **IMPORTANT NOTICES ABOUT THIS STANDARD**

**To ensure that you have all of the approved addenda, errata, and interpretations for this Standard, visit [www.ashrae.org/standards](http://www.ashrae.org/standards) to download them free of charge.**

**Addenda, errata, and interpretations for ASHRAE Standards and Guidelines are no longer distributed with copies of the Standards and Guidelines. ASHRAE provides these addenda, errata, and interpretations only in electronic form to promote more sustainable use of resources.**