**BSR/ASHRAE Addendum *u* to**
**ANSI/ASHRAE Standard 135.1-2023**

## Public Review Draft

# Proposed Addendum *u* to Standard 135.1-2023, Method of Test for Conformance to BACnet®

**First Public Review (March 2025)**
**(Draft shows Proposed Changes to Current Standard)**

This draft has been recommended for public review by the responsible project committee. To submit a comment on this proposed standard, go to the ASHRAE website at www.ashrae.org/standards-research--technology/public-review-drafts and access the online comment database. The draft is subject to modification until it is approved for publication by the Board of Directors and ANSI. Until this time, the current edition of the standard (as modified by any published addenda on the ASHRAE website) remains in effect. The current edition of any standard may be purchased from the ASHRAE Online Store at www.ashrae.org/bookstore or by calling 404-636-8400 or 1-800-727-4723 (for orders in the U.S. or Canada).

This standard is under continuous maintenance. To propose a change to the current standard, use the change submittal form available on the ASHRAE website, www.ashrae.org.

The appearance of any technical data or editorial material in this public review document does not constitute endorsement, warranty, or guaranty by ASHRAE of any product, service, process, procedure, or design, and ASHRAE expressly disclaims such.

**ASHRAE, 180 Technology Parkway NW, Peachtree Corners, GA 30092**

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

## FOREWORD

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

In the following document, language to be added to existing clauses of ANSI/ASHRAE 135.1-2023 is indicated through the use of *italics*, while deletions are indicated by ~~strikethrough~~. Where entirely new subclauses are proposed to be added, plain type is used throughout. Only this new and deleted text is open to comment at this time. All other material in this 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 XX, YY, ZZ, X1, X2, 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.

### 135.1-2023*u*-1 Updates to ELECTRONIC PICS FILE FORMAT

Rationale

Update the EPICS File Format to include the latest data links and other changes.

## 4. ELECTRONIC PICS FILE FORMAT

### 4.1 Character Encoding

…

**Table 4-1. Character Set Codes**

| code | character set |
|---|---|
| 0 | ~~ANSI X3.4~~*ISO 10646 (UTF-8)* |
| 1 | *IBM*™/Microsoft™ DBCS |
| 2 | ~~JIS C 6226~~*JIS X 0208* |
| 3 | ISO 10646 (UCS-4) |
| 4 | ISO 10646 (UCS-2) |
| 5 | ISO 8859-1 |

### 4.5 Sections of the EPICS File

### 4.5.4 Object Types Supported

This section indicates which standard object types are supported. The syntax is shown below. Each supported object type shall be listed between curly braces one object type per line, optionally followed by the words "Createable", "Deleteable", or both to indicate that dynamic creation or deletion is supported.

```
Standard Object Types Supported: ↵
{↵
   □↵
   □ Createable↵
   □ Deleteable↵
   □ Createable Deleteable↵
}↵
```

The standard objects may be any of the objects listed in the Clause 21 production, BACnetObjectTypesSupported.

The format should be the title of each corresponding object section in the standard minus the text Object Type. For example, if the device supports the *analog value* object~~ access door~~, the text ~~'Access Door'~~*'Analog Value'* should be included in this section.

For example:

~~BACnet~~ Standard ~~Application Services~~ *Object Types* Supported: ↵
{↵
Analog Value Createable Deleteable↵
Analog Input↵
Device↵
}

### 4.5.5 Data Link Layer Options

This section indicates which standard data link layer options are supported. The syntax is shown below. Each supported data link layer type shall be listed between the curly braces one per line. MS/TP and Point-To-Point data links shall also specify supported baud rate(s).

Data Link Layer Option: ↵
{↵
ISO 8802-3, 10BASE5↵
ISO 8802-3, 10BASE2↵
ISO 8802-3, 10BASET↵
ISO 8802-3, fiber↵
ARCNET, coax star↵
ARCNET, coax bus↵
ARCNET, twisted pair star↵
ARCNET, twisted pair bus↵
ARCNET, fiber star↵
ARCNET, twisted pair, EIA-485, Baud rate(s): ❑↵
MS/TP master. Baud rate(s): 9600, ❑↵
MS/TP slave. Baud rate(s): 9600, ❑↵
Point-To-Point. EIA 232, Baud rate(s): ❑↵
Point-To-Point. Modem, Baud rate(s): ❑↵
Point-To-Point. Modem, Autobaud range: ❑to ❑↵
BACnet/IP, 'DIX' Ethernet↵
BACnet/IP, Other↵
*BACnet/IPv6↵*
*BACnet/SC,Node↵*
*BACnet/SC,Hub↵*
*Zigbee↵*
Other↵
}↵

### 4.5.6 Character Sets

This section indicates which BACnet character sets are supported. The syntax is shown below. Each supported character set shall be listed one per line between the curly braces.

Character Sets Supported: ↵
{↵
*ISO 10646 (UTF-8)*~~ANSI X3.4~~↵
IBM/Microsoft DBCS↵
*JIS X 0208*~~JIS C 6226~~↵
ISO 8859-1 ↵
ISO 10646 (UCS-4) ↵
ISO 10646 (UCS2) ↵
}↵

### 4.5.7 Special Functionality

This section indicates which BACnet special functionalities are supported. The syntax is shown below. Each special functionality supported shall be listed one per line between the curly braces. The maximum APDU size and window sizes shall be specified as integers.

Special Functionality: ↵
{↵
Maximum APDU size in octets: ❑↵
Segmented Requests Supported, window size: ❑↵
Segmented Responses Supported, window size: ❑↵
Router↵
BACnet/IP BBMD↵

*BACnet/IPV6 BBMD*↵<br>
*BACnet/SC Hub*↵<br>
*BACnet/SC Direct Connect*↵<br>
}↵

**135.1-2023*u*-2 Add new and correct existing Object Support Tests**

Rationale

Errors have been identified in a number of object support tests in ANSI/ASHRAE Standard 135.1-2023.
In addition, test coverage is increased with the addition of new tests.

## 7. OBJECT SUPPORT TESTS

### 7.2 Write Support for Properties in Test Database

### 7.2.2 Write Support Test Procedure

Reason For Change: Remove specific WriteProperty and WritePropertyMultiple service requirements.

Purpose: To verify that all writable properties of all objects can be written~~ to using BACnet WriteProperty and WritePropertyMulitiple 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~~*value* of some properties *to a value different from the written*~~*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.

*Notes to Tester: Do not run this test against any properties in the Network Port objects and against the Object_Identifier of the Device object.*

Test Steps:

```
1.  REPEAT X = (all objects in the IUT's database, except *as specified in the Notes to Tester*~~Network Port objects~~) DO {
2.      REPEAT Y = (all writable properties in object X) DO {
3.          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 {
4.              WRITE (X), Y = Z,
5.              VERIFY (X), Y = Z
            }
        }
    }
```

### 7.2.4 Date Pattern Properties Test

Reason for Change:Allow IUT to return calculated day-of-week.

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 *testing*~~use~~ of ~~*an*~~ unspecified day of week is ~~always tested in conjunction with another pattern value~~ *tested separately and allows either the device to return the unspecified day of week or the correctly calculated day of week.*

Configuration Requirements: *None.*~~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)~~
*3.      WRITE P1 = (D1 updated with the value SV)*
*4.      VERIFY P1 = (D1 updated with the value SV)*
    *}*
*5.  WRITE P1 = (D1 with day of week unspecified)*
*6.  VERIFY P1 = (D1 with day of week unspecified or day of week containing the correct calculated value)*
```

### 7.2.X1 Numeric Bounds Test

Reason for Change: No test exists for this functionality.

Purpose: This test validates that the upper and lower bounds of a numeric property can be written, and values outside of the range return the correct error class and code.

Test Concept: Property (P1) in object (O1) is successfully written using the upper bound value (UB). P1 is written with a value greater than UB and an error response is verified. P1 is successfully written using the lower bound value (LB). P1 is written with a value less than LB and an error response is verified.

Configuration Requirements: If conditionally writable, Property P1, shall be made writable.

Test Steps:

```
1.  IF (UB supported) THEN {
2.      WRITE (O1), P1 = UB,
3.      VERIFY (O1), P1 = UB
4.      TRANSMIT WriteProperty-Request,
            'Object Identifier' =       O1,
            'Property Identifier' =     P1,
            'Property Value' =          (UB + 1)
5.      RECEIVE BACnet-Error-PDU,
            Error Class =           PROPERTY,
            Error Code =            VALUE_OUT_OF_RANGE
6.      VERIFY (O1), P1 = UB
    }
7.  IF (LB supported) THEN {
8.      WRITE (O1), P1 = LB,
9.      VERIFY (O1), P1 = LB
10.     TRANSMIT WriteProperty-Request,
            'Object Identifier' =       O1,
            'Property Identifier' =     P1,
            'Property Value' =          (LB - 1)
11.     RECEIVE BACnet-Error-PDU,
            Error Class =           PROPERTY,
            Error Code =            VALUE_OUT_OF_RANGE
12.     VERIFY (O1), P1 = LB
    }
```

### 7.3 Object Functionality Tests

#### 7.3.1 Property Tests

##### 7.3.1.1 Out_Of_Service, Status_Flags, and Reliability Tests

###### 7.3.1.1.1 Out_Of_Service, Status_Flags, and Reliability Test

Reason for Change: Modified test to be more automation friendly.

Purpose: To verify that Present_Value is writable when Out_Of_Service is TRUE and that the interrelationship between the Out_Of_Service, Status_Flags, and Reliability properties.

Test Concept: 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 their 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.  READ SF1 = Status_Flags
2.  READ PV1 = Present_Value
3   IF Reliability is present THEN
4.      READ R1 = Reliability
5.  IF (Out_Of_Service is writable) THEN
6.      WRITE Out_Of_Service = TRUE
    ELSE
```

```
7.      MAKE (Out_Of_Service TRUE)
8.  VERIFY Out_Of_Service = TRUE
9.  VERIFY Status_Flags = (?, ?, ?, TRUE)
10. REPEAT X = (all values meeting the functional range requirements of 7.2.1) DO {
11.     WRITE Present_Value, PTY1 = X
12.     VERIFY Present_Value = X
        }
13. WRITE Present_Value, PTY1 = PV1
14. IF (Reliability is present and writable) THEN
15.     REPEAT X = (all values of the Reliability enumeration appropriate to the object type except
                    NO_FAULT_DETECTED) DO {
16.         WRITE Reliability = X
17.         VERIFY Reliability = X
18.         VERIFY Status_Flags = (?, TRUE, ?, TRUE)
19.         WRITE Reliability = NO_FAULT_DETECTED
20.         VERIFY Reliability = NO_FAULT_DETECTED
21.         VERIFY Status_Flags = (?, FALSE, ?, TRUE)
            }
22. IF (Out_Of_Service is writable) THEN
23.     WRITE Out_Of_Service = FALSE
    ELSE
24.     MAKE (Out_Of_Service FALSE)
25. VERIFY Out_Of_Service = FALSE
26. VERIFY Status_Flags = SF1
27. IF Reliability is present THEN
28.         VERIFY Reliability = R1
```

### 7.3.1.1.2 Out_Of_Service for Commandable Value Objects Test

Reason for change: To improve the language of steps 1 and 5 to make it clear that in step 1 the change in Present_Value succeeds and that in step 5, the attempted change in Present_Value would not be expected to succeed.

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 who's 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.~~

*Test Concept: An object's Present_Value, at a priority, PTY1, is being controlled by a process, P1, internal to the IUT. P1 triggers a change to Present_Value when Out_Of_Service is FALSE and Present_Value is verified to change. Out_Of_Service is then set to TRUE and Present_Value read. P1 executes such that a change to Present_Value would occur and Present_Value is verified to be unchanged.*

Configuration Requirements: The values of the entries in the Priority_Array above PTY1 shall be NULL. *The test starts with Out_Of_Service = FALSE.*

*Notes to Tester: The specifics of the MAKE steps are defined by the vendor. They may require tester interaction or simply wait while P1 executes.*

Test Steps:

1. MAKE *P1* (~~Present_Value = PV1, any valid value, using~~ software local to the IUT *change the Present_Value at PTY1*)

```
*2.    CHECK (Present_Value changed)*
*3*.   IF (Out_Of_Service is writable) THEN
4.         WRITE Out_Of_Service = TRUE
       ELSE
5.         MAKE (Out_Of_Service TRUE)
6.     ~~VERIFY~~ *READ PV1* = Present_Value
~~4.    WRITE Present_Value, PTY1 = PV2, any valid value other than PV1~~
~~*5. VERIFY Present_Value = PV2*~~
7.     MAKE *P1* (~~Present_Value = PV3, any valid value other than PV2, using~~ software local to the IUT *attempt to change the Present_Value at PTY1 to a value other than PV1*)
8.     VERIFY Present_Value = ~~PV2~~*PV1*
```

### 7.3.1.1.X4 Out_Of_Service, Access_Event for Access Point Objects Test

Reason for Change: There is no test for this functionality.

Purpose: This test verifies the interrelationship between the Out_Of_Service and Access_Event properties. 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.

Test Concept: Write to and verify the interrelationship between the Out_Of_Service, and Access_Event properties

Configuration Requirements: The selected object is configured such that Property Access_Event is not OUT_OF_SERVICE before execution of this test.

Test Steps:

```
1.  IF (Out_Of_Service is writable) THEN
2.      WRITE Out_Of_Service = TRUE
    ELSE
3.      MAKE (Out_Of_Service = TRUE)
4.  VERIFY Out_Of_Service = TRUE
5.  VERIFY Access_Event = OUT_OF_SERVICE
6.  IF (Out_Of_Service is writable) THEN
7.      WRITE Out_Of_Service = FALSE
    ELSE
8.      MAKE (Out_Of_Service = FALSE)
9.  VERIFY Out_Of_Service = FALSE
10. VERIFY Access_Event = OUT_OF_SERVICE_RELINQUISHED
```

### 7.3.1.1.X5 Out_Of_Service, Status_Flags, Reliability and Command Property Test

Reason for Change: There is no test for this functionality.  New test per 135-2016bl-3

Purpose: This test verifies the interrelationship between the Out_Of_Service, Status_Flags, Reliability and Command properties.

Test Concept: Write to and verify the interrelationship between the Out_Of_Service, Status_Flags, Reliability and Command properties.

Configuration Requirements: The selected object is configured such that its Out_Of_Service shall be set to FALSE. If it exists, the Command property shall be set to IDLE and Reliability to NO_FAULT_DETECTED.

Notes to Tester: When applying this test to a Network Port object in a non-routing device, once the Network Port object is out of service, the only remaining part of the test that can be executed is the verification that no BACnet communications occur through that network port. The rest of the test shall be skipped.

Test Steps:

```
1.  IF (Out_Of_Service is writable) THEN
2.      WRITE Out_Of_Service = TRUE
    ELSE
3.      MAKE (Out_Of_Service = TRUE)
4.  VERIFY Out_Of_Service = TRUE
5.  VERIFY Status_Flags = (?, FALSE, ?, TRUE)
6.  IF (Reliability is present and writable) THEN
7.      REPEAT X = (all values of the Reliability enumeration appropriate to the object type except
                    NO_FAULT_DETECTED) DO {
8.          WRITE Reliability = X
9.          VERIFY Reliability = X
10.         VERIFY Status_Flags =(?, TRUE, ?, TRUE)
11.         WRITE Reliability = NO_FAULT_DETECTED
12.         VERIFY Reliability = NO_FAULT_DETECTED
13.         VERIFY Status_Flags = (?, FALSE, ?, TRUE)
    }
14. IF (Command is present) THEN
15.     REPEAT Y = (all values of the BACnetNetworkPortCommand enumeration except
                    RESTART_PORT, DISCONNECT, and DISCARD_CHANGES) DO {
16.         TRANSMIT WriteProperty-Request
                'Object Identifier' =    (the object being tested),
                'Property Identifier'=   Command,
                'Property Value' =       Y
17.         RECEIVE BACnet-Error-PDU,
                'Error Class' =      PROPERTY,
                'Error Code' =       VALUE_OUT_OF_RANGE

    }
18. CHECK (functionality that should stop or be disabled is. All communication of the protocol modeled
    by the object, through that port is disabled)
19. IF (Out_Of_Service is writable) THEN
20.     WRITE Out_Of_Service = FALSE
    ELSE
21.     MAKE (Out_Of_Service = FALSE)
22. VERIFY Out_Of_Service = FALSE
23. VERIFY Status_Flags = ( ?, ?, ?, FALSE)
```

### 7.3.1.2 Relinquish Default Test

Reason for Change: Add validation of Current_Command_Priority property. Simplify the test purpose to include all commandable objects.

Purpose: To verify that the Present_Value property takes on the value of Relinquish_Default when all prioritized commands have been relinquished. This test applies to ~~Analog Output, Analog Value, Binary Output, Binary Value, Multi-state Output, and Multi-state Value objects that are~~ objects that are commandable.

Test Concept: A pre-requisite to this test is that an object has been provided for which all prioritized commands have been relinquished and any minimum on/off time has been accounted for. The

Present_Value is compared to the value of Relinquish_Default to ensure that they are the same. If possible, the value of Relinquish_Default is changed to verify that Present_Value tracks the changes.

Configuration Requirements: The object to be tested shall be configured such that all slots in the Priority_Array have a value of NULL and no internal algorithms are issuing prioritized commands to this object.

Test Steps:

```
1.  VERIFY Priority_Array = (NULL, NULL, NULL, NULL, NULL, NULL, NULL, NULL, NULL,
        NULL, NULL, NULL, NULL, NULL, NULL, NULL)
2.  IF Protocol_Revision >= 17 THEN {
        VERIFY Current_Command_Priority = NULL
    }
3.  READ X = Present_Value
~~2.  TRANSMIT ReadProperty-Request,~~
~~        'Object Identifier' =     (the object being tested),~~
~~        'Property Identifier' =   Present_Value~~
~~3.  RECEIVE ReadProperty-ACK,~~
~~        'Object Identifier' =     (the object being tested),~~
~~        'Property Identifier' =   Present_Value~~
~~        'Property Value' =        (any valid value, X)~~
4.  VERIFY Relinquish_Default = X
5.  IF (Relinquish_Default is writable) THEN {
6.      WRITE Relinquish_Default = (any valid value, Y, other than X~~the one returned in step 3~~)
7.      VERIFY Present_Value = Y
    }
```

**7.3.1.3 Command Prioritization Test**

Reason for Change: Add validation of Current_Command_Priority property.

Purpose: To verify that the command prioritization algorithm is properly implemented. This test applies to all commandable objects.

Test Concept: The TD selects three different values $V_{low}$, $V_{med}$, and $V_{high}$ chosen from the valid values specified in 4.4.2. For *objects that are limited to two values,* $V_{low}$ and $V_{high}$ shall be the same, and $V_{med}$ shall be different. The TD also selects three priorities $P_{low}$, $P_{med}$, and $P_{high}$, all between 1 and 5, such that numerically $P_{low} > P_{med} > P_{high}$. The selected values are written one at a time to Present_Value at the corresponding priority. The Present_Value*, Current_Command_Priority* and Priority_Array are checked to verify correct operation. Priorities numerically smaller than 6 (higher priority) are used to eliminate minimum on/off time considerations.

Configuration Requirements: The object to be tested shall be configured such that all slots in the Priority_Array with a priority *numerically smaller* ~~higher~~ than 6 have a value of NULL.

Test Steps:

```
-- For clarity, the Test Steps numbering has changed but is not being shown.
1.  VERIFY Priority_Array = (NULL, NULL, NULL, NULL, NULL, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?)
--
2.  WRITE Present_Value = $V_{low}$, PRIORITY = $P_{low}$
3.  VERIFY Present_Value = $V_{low}$
4.  VERIFY Priority_Array = $V_{low}$, ARRAY INDEX = $P_{low}$
5.  REPEAT Z = (each index 1 through 5 not equal to $P_{low}$) DO {
6.      VERIFY Priority_Array = NULL, ARRAY INDEX = Z
    }
```

*7. IF Protocol_Revision >= 17 THEN {*
*8. VERIFY Current_Command_Priority = $P_{low}$*
*}*
--
9. WRITE Present_Value = $V_{high}$, PRIORITY = $P_{high}$
10. VERIFY Present_Value = $V_{high}$
11. VERIFY Priority_Array = $V_{high}$, ARRAY INDEX = $P_{high}$
12. REPEAT Z = (each index 1 through 5 not equal to $P_{low}$ or $P_{high}$) DO {
13. VERIFY Priority_Array = NULL, ARRAY INDEX = Z
}
*14. IF Protocol_Revision >= 17 THEN {*
*15. VERIFY Current_Command_Priority = $P_{high}$*
*}*
--
16. WRITE Present_Value = $V_{med}$, PRIORITY = $P_{med}$
17. VERIFY Present_Value = $V_{high}$
18. VERIFY Priority_Array = $V_{med}$, ARRAY INDEX = $P_{med}$
19. REPEAT Z = (each index 1 through 5 not equal to $P_{low}$, $P_{med}$ or $P_{high}$) DO {
20. VERIFY Priority_Array = NULL, ARRAY INDEX = Z
}
*21. IF Protocol_Revision >= 17 THEN {*
*22. VERIFY Current_Command_Priority = $P_{high}$*
*}*
--
23. WRITE Present_Value = NULL, PRIORITY = $P_{high}$
24. VERIFY Present_Value = $V_{med}$
25. REPEAT Z = (each index 1 through 5 not equal to $P_{low}$ or $P_{med}$) DO {
26. VERIFY Priority_Array = NULL, ARRAY INDEX = Z
}
*27. IF Protocol_Revision >= 17 THEN {*
*28. VERIFY Current_Command_Priority = $P_{med}$*
*}*
--
29. WRITE Present_Value = NULL, PRIORITY = $P_{med}$
30. VERIFY Present_Value = $V_{low}$
31. REPEAT Z = (each index 1 through 5 not equal to $P_{low}$ ) DO {
32. VERIFY Priority_Array = NULL, ARRAY INDEX = Z
}
*33. IF Protocol_Revision >= 17 THEN {*
*34. VERIFY Current_Command_Priority = $P_{low}$*
*}*
--
35. WRITE Present_Value = NULL, PRIORITY = $P_{low}$
*36. VERIFY Priority_Array = (NULL, NULL, NULL, NULL, NULL, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?)*
~~28. REPEAT Z = (each index 1 through 5) DO {~~
~~VERIFY Priority_Array = NULL, ARRAY INDEX = Z~~
~~}~~
*37. IF Protocol_Revision >= 17 THEN {*
*38. VERIFY Current_Command_Priority is numerically larger than 5 or NULL*
*}*

### 7.3.1.4 Minimum_Off_Time

Reason for Change: Add validation of Current_Command_Priority property.

Purpose: To verify that the minimum off time algorithm is properly implemented. ~~If minimum off time is not supported this test shall be omitted. This test applies to Binary Output and Binary Value objects.~~

Test Concept: The initial Present_Value of the object tested is set to ACTIVE and it is controlled at a priority numerically *larger*~~greater~~ (lower priority) than 6. The object has been in this state long enough for *the*~~any~~ minimum on time to have expired. The Present_Value is written to with a value of INACTIVE at priority 7. The value of slot 6 of the Priority_Array is monitored to verify that it contains the value INACTIVE for the duration of the minimum off time.

Configuration Requirements: The object to be tested shall be configured such that all slots in the Priority_Array numerically *smaller*~~less~~ than 7 have a value of NULL, the Present_Value is ACTIVE, and no internal algorithms are issuing commands to this object at a priority numerically *smaller*~~lower~~ (higher priority) that the priority that is currently controlling Present_Value.

Test Steps:

1. WRITE Present_Value = INACTIVE, PRIORITY = 7
2. VERIFY Present_Value = INACTIVE
3. VERIFY Priority_Array = INACTIVE, ARRAY INDEX = 6
4. *VERIFY Priority_Array = INACTIVE, ARRAY INDEX = 7*
5. *IF Protocol_Revision >= 17 THEN*
6.     *VERIFY Current_Command_Priority = 6*
7. WAIT (approximately 90% of Minimum_Off_Time~~ from step 1~~)
8. VERIFY Priority_Array = INACTIVE, ARRAY INDEX = 6
9. WAIT (**Minimum ON/OFF Fail Time** + Minimum_Off_Time~~ from step 1~~)
10. VERIFY Priority_Array = NULL, ARRAY INDEX = 6
11. *VERIFY Priority_Array = INACTIVE, ARRAY INDEX = 7*
12. *IF Protocol_Revision >= 17 THEN*
13.     *VERIFY Current_Command_Priority = 7*

### 7.3.1.5 Minimum_On_Time

Reason for Change: Add validation of Current_Command_Priority property.

Purpose: To verify that the minimum on time algorithm is properly implemented. ~~If minimum on time is not supported this test shall be omitted. This test applies to Binary Output and 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 *larger*~~greater~~ (lower priority) than 6. The object has been in this state long enough for *the*~~any~~ minimum *off*~~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 for the duration of the minimum on time.

Configuration Requirements: The object to be tested shall be configured such that all slots in the Priority_Array numerically *smaller*~~less~~ than 7 have a value of NULL, the Present_Value is INACTIVE, and no internal algorithms are issuing commands to this object at a priority numerically *smaller*~~lower~~ (higher priority) that the priority that is currently controlling Present_Value.

Test Steps:

1. WRITE Present_Value = ACTIVE, PRIORITY = 7
2. VERIFY Present_Value = ACTIVE
3. VERIFY Priority_Array = ACTIVE, ARRAY INDEX = 6
4. *VERIFY Priority_Array = ACTIVE, ARRAY INDEX = 7*
5. *IF Protocol_Revision >= 17 THEN*

*6*     *VERIFY Current_Command_Priority = 6*

7. WAIT (approximately 90% of Minimum_On_Time~~ from step 1~~)
8. VERIFY Priority_Array = ACTIVE, ARRAY INDEX = 6
9. WAIT (**Minimum ON/OFF Fail Time** + Minimum_On_Time~~ from step 1~~)

```
10. VERIFY Priority_Array = NULL, ARRAY INDEX = 6
11. VERIFY Priority_Array = ACTIVE, ARRAY INDEX = 7
12. IF Protocol_Revision >= 17 THEN
13.     VERIFY Current_Command_Priority = 7
```

### 7.3.1.9 Elapsed Active Time Test

Reason for the change: Correct calculation error in step 14.

Purpose: To verify that the properties of objects that collectively track active time function properly.

Test Concept: The Present_Value or Feedback_Value of the 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 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
2.      WRITE Present_Value = INACTIVE
3.      VERIFY Present_Value = INACTIVE
    ELSE
4.      MAKE (Present_Value = INACTIVE)
5.  IF (Feedback_Value is used for Elapsed_Active_Time tracking) THEN
6.      WAIT(long enough for Feedback_Value to reflect the Present_Value)
7.      VERIFY Feedback_Value = INACTIVE
8.  READ Elapsed_Active_Time = initialElapsedTime

-- verify that Elapsed_Active_Time does not change when the object is INACTIVE
9.  WAIT (more than Internal_Processing Fail Time + at least 1 second)
10. VERIFY Elapsed_Active_Time = initialElapsedTime

-- verify that Elapsed_Active_Time correctly reflects the time the object is ACTIVE5
11. IF (Present_Value is writable) THEN
12.     WRITE Present_Value = ACTIVE
13.     VERIFY Present_Value = ACTIVE
    ELSE
14.     MAKE (Present_Value = ACTIVE)
15. IF (Feedback_Value is used for Elapsed_Active_Time tracking) THEN
16.     WAIT (long enough for Feedback_Value to reflect the Present_Value)
17.     VERIFY Feedback_Value = ACTIVE
18. READ initialTime = (the IUT's Device object) Local_Time
19. WAIT (more than Internal Processing Fail Time + 30 seconds)
20. IF (Present_Value is writable) THEN
21.     WRITE Present_Value = INACTIVE
22.     VERIFY Present_Value = INACTIVE
    ELSE
```

```
23.       MAKE (Present_Value = INACTIVE)
24.  IF (Feedback_Value is used for Elapsed_Active_Time tracking) THEN
25.       WAIT (long enough for Feedback_Value to reflect the Present_Value)
26.       VERIFY Feedback_Value = INACTIVE
27.  READ currentTime = (the IUT's Device object) Local_Time
28.  READ totalElapsedTime = Elapsed_Active_Time
~~14.  CHECK (totalElapsedTime ~= (currentTime - initialTime) - initialElapsedTime)~~
*29.  CHECK (totalElapsedTime ~= (currentTime - initialTime) + initialElapsedTime)*

-- verify ability to reset Elapsed_Active_Time, if it is writable
30.  IF (Elapsed_Active_Time is writable) THEN
31.       WRITE Elapsed_Active_Time = 0
32.       READ currentDate = (the IUT's Device object) Local_Date
33.       READ currentTime = (the IUT's Device object) Local_Time
34.       VERIFY Time_Of_Active_Time_Reset ~= { currentDate, currentTime }
```

### 7.3.1.11 Acked_Transitions Tests

#### 7.3.1.11.1 Acked_Transitions Test

Reason for Change: Corrected errata issues that are in 135.1-2023. Improved the text for Notes To Tester. Removed to_fault part of the test and moved to a new 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.

*Notes to Tester: For life safety objects that latch pMonitoredValue, the LifeSafetyOperation Service will be required to reset pMonitoredValue.*

Test Steps:

```
1.   VERIFY pCurrentState = NORMAL
2.   VERIFY Acked_Transitions = (TRUE, TRUE, TRUE)
3.   IF (Protocol_Revision is present AND Protocol_Revision >= 13) THEN
4.        VERIFY Status_Flags  = (FALSE, FALSE, ?, ?)
5.   IF (pMonitoredValue is writable) THEN
6.        WRITE pMonitoredValue = (a value that is OFFNORMAL)
     ELSE
7.        MAKE (pMonitoredValue have a value that is OFFNORMAL)
```

```
8.  WAIT (pTimeDelay)
9.  BEFORE Notification Fail Time
10.     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)
11. TRANSMIT BACnet-SimpleACK-PDU
12. VERIFY pCurrentState = OFFNORMAL
13. VERIFY Acked_Transitions = (FALSE, TRUE, TRUE)
14. IF (Protocol_revision is present AND Protocol_Revision >= 13 THEN
15.     VERIFY Status_Flags   = (TRUE, FALSE, ?, ?)
16. IF (pMonitoredValue is writable) THEN
17.     WRITE pMonitoredValue = (a value that is NORMAL)
    ELSE
18.     MAKE (pMonitoredValue have a value that is NORMAL)
19. WAIT (pTimeDelayNormal)
20. BEFORE Notification Fail Time
21.     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' =                     OFFNORMAL,
            'To State' =                       NORMAL,
            'Event Values' =                   (values appropriate to the event type)
22. TRANSMIT BACnet-SimpleACK-PDU
23. VERIFY pCurrentState = NORMAL
24. VERIFY Acked_Transitions = (FALSE, TRUE, FALSE)
25. IF (Protocol_Revision is present AND Protocol_Revision >= 13) THEN
26.     VERIFY Status_Flags   = (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' =                       (Tfault *the 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' =                       (Tfault *the 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)~~
    ~~}~~
27. 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)
28.  RECEIVE BACnet-SimpleACK-PDU
29.  IF (Protocol_Revision is present and Protocol_Revision ≥ 1) THEN
30.       BEFORE Notification Fail Time
31.            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 being tested),
                    'Priority' =                      (Pnormal),
                    'Event Type' =                    (any valid event type),
                    'Message Text' =                  (optional, any valid message text),
                    'Notify Type' =                   ACK_NOTIFICATION,
                    'To State' =                      NORMAL
     ELSE
32.       BEFORE Notification Fail Time
33.            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
34.  TRANSMIT BACnet-SimpleACK-PDU
35.. VERIFY Acked_Transitions = (FALSE, TRUE, TRUE)
36.  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)
37.  RECEIVE BACnet-SimpleACK-PDU
38.  IF (Protocol_Revision is present and Protocol_Revision ≥ 1) THEN
39.       BEFORE Notification Fail Time
40.            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
```

```
41.     BEFORE **Notification Fail Time**
42.         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
43. TRANSMIT BACnet-SimpleACK-PDU
44. VERIFY Acked_Transitions = (TRUE, TRUE, TRUE)
```

### 7.3.1.11.2 Acked_Transitions Test for Latching Objects

[Remove Test 7.3.1.11.2]

### 7.3.1.11.X1 Acked_Transitions Test for Normal-To-Normal Transitions

Reason for Change: No test exists for this functionality.

Purpose: To verify that the Acked_Transitions property tracks whether an acknowledgment has been received for a previously issued normal 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 normal event transitions are to trigger an event notification. The normal 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 when an acknowledgment is received.

Configuration Requirements: The Event_Enable and Acked_Transitions properties shall be configured with a value of (?, ?, 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).

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 notification.

Notes to Tester: For life safety objects that latch pMonitoredValue, the LifeSafetyOperation Service will be required to reset pMonitoredValue.

Test Steps:

```
1. VERIFY pCurrentState = NORMAL
2. VERIFY Acked_Transitions = (?, ?, TRUE)
3. IF (Protocol_Revision is present AND Protocol_Revision >= 13) THEN
     VERIFY Status_Flags = (FALSE, FALSE, ?, ?)
4. IF (pMonitoredValue is writable) THEN
5.         WRITE pMonitoredValue = (a value that will result in a TO_NORMAL transition)
     ELSE
6.         MAKE (pMonitoredValue a value that will result in a TO_NORMAL transition)
7. WAIT (pTimeDelayNormal)
8. BEFORE **Notification Fail Time**
9.   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' = NORMAL,
            'To State' = NORMAL,
            'Event Values' = (values appropriate to the event type)
10. TRANSMIT BACnet-SimpleACK-PDU
11. VERIFY pCurrentState = NORMAL
12. VERIFY Acked_Transitions = (?, ?, FALSE)
13. IF (Protocol_Revision is present AND Protocol_Revision >= 13) THEN
14.         VERIFY Status_Flags = (FALSE, FALSE, ?,?)
15. 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)
16. RECEIVE BACnet-SimpleACK-PDU
17. IF (Protocol_Revision is present AND Protocol_Revision >= 1) THEN
18.         BEFORE Notification Fail Time
19.             RECEIVE ConfirmedEventNotification-Request,
                    'Process Identifier' = (PI2),
                    'Initiating Device Identifier' = IUT,
                    'Event Object Identifier' = (the event-generating object configured for this test),
                    'Time Stamp' = (the 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),
                    'Message Text' = (optional, any valid message text),
                    'Notify Type' = ACK_NOTIFICATION,
                    'To State' = NORMAL
    ELSE
20.         BEFORE Notification Fail Time
21.             RECEIVE ConfirmedEventNotification-Request,
                    'Process Identifier' = (PI2),
                    'Initiating Device Identifier' = IUT,
                    'Event Object Identifier' = (the event-generating object configured for this test),
                    'Time Stamp' = (the 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),
                    'Message Text' = (optional, any valid message text),
                    'Notify Type' = ACK_NOTIFICATION
22. TRANSMIT BACnet-SimpleACK-PDU
23. VERIFY Acked_Transitions = (?, ?, TRUE)
```

### 7.3.1.11.X2 Acked_Transitions Test for Normal to Fault Transitions

Reason for Change: New test.

Purpose: To verify that the Acked_Transitions property tracks whether or not an acknowledgment has been received for a previously issued fault 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 fault event transitions are to trigger an event notification. The Acked_Transitions property shall have the value (?, TRUE, ?). The fault 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 when an acknowledgment is received.

Configuration Requirements: The Event_Enable and Acked_Transitions properties shall be configured with a value of (?, 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, ?).

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, ?)
3.  IF (Protocol_Revision is present AND Protocol_Revision >= 13) THEN
4.      VERIFY Status_Flags = (FALSE, FALSE, ?, ?)
5.  MAKE (a condition exist that will cause the object to generate a fault condition)
6.  BEFORE Notification Fail Time
7.      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)
8.  TRANSMIT BACnet-SimpleACK-PDU
9.  VERIFY pCurrentState = FAULT
10. VERIFY Acked_Transitions = (?, FALSE, ?)
11. IF (Protocol_revision is present AND Protocol_Revision >= 13 THEN
12.     VERIFY Status_Flags = (FALSE, TRUE, ?, ?)
13. 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)
14. RECEIVE BACnet-SimpleACK-PDU
15. IF (Protocol_Revision is present AND Protocol_Revision ≥ 1) THEN
16.     BEFORE **Notification Fail Time**
17.         RECEIVE ConfirmedEventNotification-Request,
                'Process Identifier' =              (PI3),
                'Initiating Device Identifier' =    IUT,
                'Event Object Identifier' =         (the event-generating object configured for this test),
                'Time Stamp' =                      (Tfault the 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
18.     BEFORE **Notification Fail Time**
19.         RECEIVE ConfirmedEventNotification-Request,
                'Process Identifier' =              (PI3),
                'Initiating Device Identifier' =    IUT,
                'Event Object Identifier' =         (the event-generating object configured for this test),
                'Time Stamp' =                      (Tfault the 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),
                'Message Text' =                    (optional, any valid message text),
                'Notify Type' =                     ACK_NOTIFICATION
20. TRANSMIT BACnet-SimpleACK-PDU
21. VERIFY Acked_Transitions = (?, TRUE, ?)
```

### 7.3.1.16 Array Resizing Test

Reason for Change: Modified the test steps to match the Test Concept of < and > vs <= and >=.

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

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

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

Test Steps:

```
1. WRITE (the array property being tested) = (array of non-zero size N1)
2. VERIFY (array is as written in step 1)
```

3. WRITE (the array property being tested) = (array of non-zero size N2, ~~*where N2 ≤ N1*~~ *where N2 < N1*)
4. VERIFY (array is as written in step 3)
5. WRITE (the array property being tested) = (array of non-zero size N3, ~~*where N3 ≥ N1*~~ *where N3 > N1*)
6. VERIFY (array is as written in step 5)
7. WRITE (the array property being tested) = (a non-zero unsigned value N4, ~~*where N4 ≤ N1*~~ *where N4 < N1*), ARRAY INDEX = 0
8. VERIFY (array contains first N4 elements of the array written in step 5)
9. WRITE (the array property being tested) = (N5, ~~*where N5 ≥ N4*~~ *where N5 > N4*), ARRAY INDEX = 0
10. VERIFY (array contains first N4 elements of the array written in step 5, plus N5 – N4 additional elements, initialized to
    particular values if specified for the array property being tested)
11. TRANSMIT WriteProperty-Request,
    'Object Identifier' = (the object being tested),
    'Property Identifier' = (the array property being tested),
    'Property Array Index' = (N6, ~~*where N6 ≥ N5*~~ *where N6 > N5*),
    'Property Value' = (one array element)
12. RECEIVE BACnet-Error-PDU
    Error Class = PROPERTY,
    Error Code = INVALID_ARRAY_INDEX
13. VERIFY (array is unchanged from step 10)
14. IF (the array can be resized to have zero elements) THEN
    WRITE (the array property being tested) = (empty array)
    VERIFY (array is empty)

**7.3.1.21 Reliability_Evaluation_Inhibit Tests**

**7.3.1.21.1 Reliability_Evaluation_Inhibit Test**

Reason for Change: Removed conditional event reporting and the conditionality that allows the test to be skipped.

Purpose: To verify that Reliability_Evaluation_Inhibit controls whether or not fault conditions are detected *and events are generated*.

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

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

*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 fault condition exist for O1~~*a condition exist that would cause O1 to generate a TO_FAULT transition*)
4. ~~IF the IUT supports event reporting THEN~~BEFORE **Notification Fail Time**

5. 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)
6. VERIFY Reliability <> NO_FAULT_DETECTED
7. *VERIFY pCurrentState = FAULT*
8. IF Reliability_Evaluation_Inhibit is writable THEN
9.   WRITE Reliability_Evaluation_Inhibit = TRUE
  ELSE
10.   MAKE(Reliability_Evaluation_Inhibit TRUE)
11. ~~IF the IUT supports event reporting THEN~~
12. BEFORE **Internal Processing Fail Time** + **Notification Fail Time**
13.   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)
14. VERIFY Reliability = NO_FAULT_DETECTED
15. VERIFY pCurrentState = NORMAL
16. MAKE(remove the fault condition)
17. WAIT(pTimeDelayNormal)
18. WAIT **Notification Fail Time**
19. CHECK (that the IUT did not send any event notifications for O1)
20. VERIFY Reliability = NO_FAULT_DETECTED
21. MAKE(~~a fault condition exist for O1~~*a condition exist that would cause O1 to generate a TO_NORMAL transition*)
22. WAIT **Notification Fail Time**
23. VERIFY Reliability = NO_FAULT_DETECTED
24. VERIFY pCurrentState = NORMAL
25. CHECK (that the IUT did not send any event notifications for O1)

### 7.3.1.21.X1 Reliability_Evaluation_Inhibit Object Test

Reason for Change: No test for this functionality.

Purpose: To verify that Reliability_Evaluation_Inhibit controls whether or not an object performs the reliability-evaluation process.

Test Concept: Select an object, O1, which supports the Reliability_Evaluation_Inhibit property. With Reliability_Evaluation_Inhibit FALSE, make a fault condition exist. Verify that the Reliability property changes. Set Reliability_Evaluation_Inhibit to TRUE. Verify that the Reliability changes to NO_FAULT_DETECTED. Make a fault condition exist and verify that Reliability property remains NO_FAULT_DETECTED.

Configuration Requirements: The Event_State of O1 is NORMAL, and Reliability_Evaluation_Inhibit equals FALSE.

Test Steps:

```
1.  VERIFY Event_State = NORMAL
2.  VERIFY Reliability = NO_FAULT_DETECTED
3.  MAKE(a fault condition exist for O1)
4.  VERIFY Reliability <> NO_FAULT_DETECTED
5.  VERIFY Event_State = FAULT
6.  IF Reliability_Evaluation_Inhibit is writable THEN
7.      WRITE Reliability_Evaluation_Inhibit = TRUE
    ELSE
8.      MAKE(Reliability_Evaluation_Inhibit TRUE)
9.  VERIFY Reliability = NO_FAULT_DETECTED
10. VERIFY Event_State = NORMAL
11. MAKE(remove the fault condition)
12. VERIFY Reliability = NO_FAULT_DETECTED
13. MAKE(a fault condition exist for O1)
14. VERIFY Reliability = NO_FAULT_DETECTED
15. VERIFY Event_State = NORMAL
```

### 7.3.1.22 Event_Detection_Enable Tests

#### 7.3.1.22.1 Event_Detection_Enable Inhibits Event Generation

Reason for Change: Remove expectation that Event_Detection_Enable is able to be equal to both TRUE and FALSE as per CR-0417.

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

Test Concept: Select an event generating object, O1, that ~~is configured for~~ *supports* event reporting. ~~If possible, make the object generate an event, to an offnormal, so that if the object can have a non-normal state, it enters that state early in the test. This will help detect incorrect implementations that initiate a TO_NORMAL transition when the algorithm is disabled. Set the Event_Detection_Enable property to FALSE.~~ Verify the Event_State is NORMAL and ~~the~~ Acked_Transitions, Event_Time_Stamps, and Event_Message_Texts are equal to their respective initial conditions, as mandated in the standard. ~~Repeat the process that made the object generate an event and observe that no notification messages are transmitted.~~ *Make a condition exist that would cause a transition if event reporting were enabled and observe that no notification messages are transmitted.*

Configuration Requirements: O1 is configured *with Event_Detection_Enable set to FALSE. If Event_Detection_Enable cannot be set to FALSE, this test shall be skipped. DELAY shall represent the time delay appropriate to the transition being tested (i.e. Time_Delay for TO_OFFNORMAL, 0 for TO_FAULT and FAULT to NORMAL transitions, and either Time_Delay or Time_Delay_Normal for TO_NORMAL).*~~to detect and report unconfirmed events and requires acknowledgments for all transitions. Event_Detection_Enable is equal to TRUE. D1 is either the pTimeDelay, or pTimeDelayNormal parameter, or 0 (for transitions to and from FAULT state) depending on the event transition.~~ For this test, NO_TS equals a BACnetDateTime with all unspecified values, a BACnet Time with all unspecified values, or a sequence number of 0.

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

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

### 7.3.1.22.2 Event_Detection_Enable Inhibits FAULT

Reason for Change: Fix to address the fact that Reliability_Evaluation_Inhibit is not a required property.

Purpose: To verify that Event_Detection_Enable disables fault reporting.

Test Concept: ~~When the event-state-detection process is disabled via the Event_Detection_Enable, both the event algorithm and the Reliability value are ignored, and Event_State remains NORMAL. Select an event generating object, O1, that is configured for event reporting, and which can be made to go into FAULT. Set the Event_Detection_Enable property to FALSE. Create a condition which will cause O1 to transition to FAULT, if Event_Detection_Enable is TRUE.~~ *An event generating object, O1, is put into a condition that would cause it to go into a FAULT state if Event_Detection_Enable were TRUE.* Verify the Event_State is NORMAL and the Acked_Transitions, Event_Time_Stamps, and Event_Message_Texts are equal to their respective initial conditions, as mandated in the standard, and no notification messages are transmitted.

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

Test Steps:

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

### 7.3.1.X54 Color_Override Property Tests

### 7.3.1.X54.1 Color Override Test

Reason for Change: No test exists for this functionality.

Purpose: This test ensures that a color override will follow the existing transition or no transition, but to the new commanded color or color temperature.

Test Concept: A companion Color or Color Temperature object (COLOR1) is referenced by the Color_Reference property of the lighting output object (O1). A different Color or Color Temperature object (COLOR2) is referenced by the Override_Color_Reference property of O1. No transitions or fades are in progress. Color override is enabled and the output of O1 is verified to match the color referenced by COLOR2. Color override is disabled and the output of O1 is verified to match the color referenced by COLOR1. A Color_Command is written to COLOR1 which would cause a transition to begin. During the transition period, color override is enabled and the output is verified to go to the color referenced by COLOR2 immediately. The override is again disabled and TD verifies that the color returns to the color that would have been in effect had the override not occurred. One more override is enabled and a Color_Command is written to COLOR2 which would cause another transition to begin. During this transition color override is disabled and the TD verifies that the output matches COLOR1's Present_Value.

Configuration Requirements: COLOR1 is referenced by O1's Color_Reference property. COLOR2 is referenced by O1's Override_Color_Reference property. Color_Override is False. No fades or transitions are taking place at the start of the test. COLOR1 and COLOR 2 exist in the IUT, are of the same object type, and have different Present_Values at the start of the test.

Notes to tester: Select Present_Values for COLOR1 and COLOR2 which are measurably different from each other, and color commands which result in measurable changes, to facilitate a successful test run. The CHECK steps will need to be defined by the vendor if a physical output device is not provided.

Test Steps:

```
1.  READ C1 = (COLOR1), Present_Value
2.  READ C2 = (COLOR2), Present_Value
3.  WRITE (O1), Color_Override = TRUE
4.  CHECK (that the color output of O1 is represented by C2)
5.  WRITE (O1), Color_Override = FALSE
6.  CHECK (that the color output of O1 is represented by C1)
7.  WRITE (COLOR1), Color_Command = (any valid operation for the color object with a specified fade time T1 which will still be in effect after step 10)
8.  WRITE (O1), Color_Override = TRUE
9.  CHECK (that the color output of O1 is represented by C2 and is not doing any fading)
10. WRITE (O1), Color_Override = FALSE
11. CHECK (that the color output of O1 is now fading in the direction of the last color command written to COLOR1)
12. WAIT (T1 seconds)   -- Wait for the transition to finish
13. CHECK (the color output from O1 is represented by the last color command written to COLOR1)
14. WRITE (O1), Color_Override = TRUE
15. WRITE (COLOR2), Color_Command = (any valid operation for the color object with a specified fade time T2 which will still be in effect after step xyz)
16. CHECK (that the color output of O1 is fading to a color that represents the last color command written to COLOR2)
17. BEFORE (T2 has elapsed)
18.     WRITE (O1), Color_Override = FALSE
19. CHECK (that the color output of O1 is represented by the last color command written to COLOR1 and is not doing any fading)
```

### 7.3.1.X1 Verify No Objects Contain a Zero Length Object_Name

Reason for Change: No test exists.

Purpose: To verify that Object_Name is not blank for any object in the IUT.

Test Concept: Read the Object_List from the IUT and for each object in the IUT, read the Object_Name and verify that its length is not zero.

Configuration Requirements: None.

Notes to Tester: If the whole BACnetARRAY cannot be read because it exceeds the Maximum Transmissible APDU, then the tester shall read it element-by-element to obtain the complete value.

Test Steps:

```
1. READ OL = Object_List
2. REPEAT O1 = (each object in the content of OL) DO {
3.     READ NM = O1, Object_Name
4.     CHECK (NM length > 0)
   }
```

### 7.3.1.X2 Verify a Zero Length Object_Name cannot be Written

Reason for Change: No test exists.

Purpose: To verify that for every object in the IUT, an empty string value cannot be written to the Object_Name property.

Test Concept: For each object in the IUT, read the Object_Name. Write an empty string to the Object_Name property and verify the name does not change and an appropriate error is returned.

Configuration Requirements: None.

Notes to Tester: If the whole BACnetARRAY cannot be read because it exceeds the Maximum Transmissible APDU, then the tester shall read it element-by-element to obtain the complete value.

Test Steps:

```
1.  READ OL = Device, Object_List
2.  REPEAT O1 = (each object in content of OL) DO {
3.        READ NM = Object, Object_Name
4.        TRANSMIT WriteProperty-Request,
              'Object Identifier' = O1,
              'Property Identifier' = Object_Name,
              'Property Value' = (empty string)
5.        RECEIVE BACnet-Error-PDU,
              'Error Class' = PROPERTY,
              'Error Code' = WRITE_ACCESS_DENIED | VALUE_OUT_OF_RANGE
6.        READ NM2 = O1, Object_Name
7.        VERIFY NM2 == NM
    }
```

### 7.3.1.X73 Number_Of_States Property Tests

### 7.3.1.X73.1 Writable Number_Of_States Test

Reason for Change: New Test

Purpose: This test verifies that when the value of the Number_Of_States property is written, the size of the State_Text array is changed accordingly.

Test Concept: N1 and N2 are valid values of the Number_Of_States property, N1 and N2 do not equal the value of the Number_Of_States, and N1 does not equal N2. The value of the Number_Of_States property is written to N1 and the size of the State_Text and the value of the Number_Of_States property is verified. The procedure is repeated with N2 and again with N1.

Configuration Requirements: The IUT shall be configured with a Multi-state object O1, with a writable Number_Of_States-property.

Test Steps:

```
1.  WRITE O1, Number_Of_States = N1
2.  VERIFY Number_Of_States = N1
3.  VERIFY State_Text = N1, ARRAY INDEX = 0
4.  WRITE O1, Number_Of_States = N2
5.  VERIFY Number_Of_States = N2
6.  VERIFY State_Text = N2, ARRAY INDEX = 0
7.  WRITE O1, Number_Of_States = N1
```

8. VERIFY Number_Of_States = N1
9. VERIFY State_Text = N1, ARRAY INDEX = 0

### 7.3.1.X110 State_Text Property Tests

### 7.3.1.X110.1 Resizable State_Text Test

Reason for Change: New Test.

Purpose: This test verifies that when the State_Text array is changed, the value of the Number_Of_States property is changed accordingly to the same size.

Test Concept: N1 and N2 are valid sizes for the State_Text property, N1 and N2 do not equal the present size of the State_Text, and N1 does not equal N2. The size of the State_Text property is written to N1 and the value of the Number_Of_States and the size of the State_Text is verified. The procedure is repeated with N2. The size of the State_Text is changed to N1 by writing the entire array and Number_Of_States and the size of the State_Text is verified. The procedure is repeated with N2.

Configuration Requirements: The IUT shall be configured with a Multi-state object O1, with a resizable State_Text array.

Test Steps:

1. WRITE O1, State_Text = N1, ARRAY INDEX = 0
2. VERIFY Number_Of_States = N1
3. VERIFY State_Text = N1, ARRAY INDEX = 0
4. WRITE O1, State_Text = N2, ARRAY INDEX = 0
5. VERIFY Number_Of_States = N2
6. VERIFY State_Text = N2, ARRAY INDEX = 0
7. TRANSMIT WriteProperty-Request,
   'Object Identifier' = (O1),
   'Property Identifier' = State_Text,
   'Property Value' = (State_Text array of length N1)
8. RECEIVE Simple-ACK-PDU
9. VERIFY Number_Of_States = N1
10. VERIFY State_Text = N1, ARRAY INDEX = 0
11. TRANSMIT WriteProperty-Request,
    'Object Identifier' = (O1),
    'Property Identifier' = State_Text,
    'Property Value' = (State_Text array of length N2)
12. RECEIVE Simple-ACK-PDU
13. VERIFY Number_Of_States = N2
14. VERIFY State_Text = N2, ARRAY INDEX = 0

### 7.3.2 Object Specific Tests

### 7.3.2.10 Device Object Tests

### 7.3.2.10.1 Active_COV_Subscriptions SubscribeCOV Test

Reason for Change: Double negation was confusing in the Test Concept.

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: $INC_1$, $INC_2$, and $INC_3$ 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, $O_1$, $O_2$, and $O_3$, for which the device supports SubscribeCOV. $O_1$, $O_2$, and $O_3$ are not required to refer to different objects. The tester shall also choose three nonzero unique process identifiers, $P_1$, $P_2$, and $P_3$, and three non-zero lifetimes $L_1$, $L_2$, and $L_3$. Lifetime $L_1$ shall be long enough to allow the initial part of the test to run through to step 14. Lifetimes $L_2$ and $L_3$ 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' =   $P_1$,
            'Monitored Object Identifier' =     $O_1$,
            'Issue Confirmed Notifications' =   TRUE,
            'Lifetime' = $L_1$
2.  RECEIVE BACnet-SimpleACK-PDU
3.  BEFORE **Notification Fail Time**
4.      RECEIVE ConfirmedCOVNotification-Request,
            'Subscriber Process Identifier' = $P_1$,
            'Initiating Device Identifier' = IUT,
            'Monitored Object Identifier' = $O_1$,
            'Time Remaining' = (a value approximately equal to $L_1$),
            'List of Values' =  (values appropriate to the object type of the monitored object)
5.  TRANSMIT BACnet-SimpleACK-PDU
6.  IF P1 is numeric THEN
7.      VERIFY Active_COV_Subscriptions = {{ {TD, $P_1$}, {$O_1$, Present_Value }, TRUE, (a value less than $L_1$),
                    ($INC_1$ : not present or a valid Increment)}}
    ELSE
8.      VERIFY Active_COV_Subscriptions = {{ {TD, $P_1$, { $O_1$, Present_Value }, TRUE, (a value less than $L_1$),
    ($INC_1$: not present)}}
9.  TRANSMIT SubscribeCOV-Request,
        'Subscriber Process Identifier' =       $P_2$,
        'Monitored Object Identifier' =         $O_2$,
        'Issue Confirmed Notifications' =       FALSE,
          'Lifetime' = $L_2$
10. RECEIVE BACnet-SimpleACK-PDU
11. BEFORE **Notification Fail Time**
12.     RECEIVE UnconfirmedCOVNotification-Request,
            'Subscriber Process Identifier' =   $P_2$,
            'Initiating Device Identifier' =    IUT,
            'Monitored Object Identifier' =     $O_2$,
            'Time Remaining' =      (a value approximately equal to $L_2$),
            'List of Values' =    (values appropriate to the object type of the monitored object)
13. VERIFY Active_COV_Subscriptions = {{ {TD, $P_1$}, {$O_1$, Present_Value}, TRUE, (a value less than $L_1$), *$INC_1$*},
                                      {{TD, $P_2$}, {$O_2$, Present_Value}, FALSE, (a value less than $L_2$),
                                        ($INC_2$: not present if the property is not numeric; present
                                         if a valid Increment was provided in the subscription;
                                        optionally present otherwise}}
14. TRANSMIT SubscribeCOV-Request,
        'Subscriber Process Identifier' =   $P_3$,
        'Monitored Object Identifier' =     $O_3$,
        'Issue Confirmed Notifications' =   FALSE,
        'Lifetime' =    $L_3$
15. RECEIVE BACnet-SimpleACK-PDU
16. BEFORE **Notification Fail Time**
17.     RECEIVE UnconfirmedCOVNotification-Request,
            'Subscriber Process Identifier' =   $P_3$,
            'Initiating Device Identifier' =    IUT,
            'Monitored Object Identifier' =     $O_3$,
            'Time Remaining' =     (a value approximately equal to $L_3$),
            'List of Values' =    (values appropriate to the object type of the monitored object)
18. IF $P_3$ is numeric THEN
19.     VERIFY Active_COV_Subscriptions = {{{TD, $P_1$}, {$O_1$, Present_Value}, TRUE, (a value less than $L_1$), $INC_1$},
                                          {{TD, P2}, {O2, Present_Value}, FALSE, (a value less than L2), $INC_2$},
                                          {{TD, P3}, {O3, Present_Value}, FALSE, (a value less than L3),
                                              $INC_3$: not present or ~~(~~a valid Increment }}
    ELSE
20.         VERIFY Active_COV_Subscriptions = {{{TD, $P_1$}, {$O_1$, Present_Value}, TRUE, (a value less than $L_1$), $INC_1$ },
                                              {{TD, $P_2$,}, {$O_2$, Present_Value}, FALSE, (a value less than $L_2$), $INC_2$ },
                                              {{TD, $P_3$}, {$O_3$, Present_Value}, FALSE, (a value less than $L_3$),
                                                  ($INC_3$: not present)}}
21. WAIT $L_1$ + the IUT's timer granularity
22. VERIFY Active_COV_Subscriptions = {{TD, $P_2$ }, {$O_2$ , Present_Value}, FALSE, (a value less than $L_2$ ),
                                      $INC_2$ (a valid Increment if the property is REAL)},
                                  {{TD, $P_3$ }, {$O_3$ , Present_Value}, FALSE, (a value less than $L_3$ ),
                                      $INC_3$(a valid Increment if the property is REAL)}}
23. TRANSMIT SubscribeCOV-Request,
        'Subscriber Process Identifier' =  $P_3$ ,
        'Monitored Object Identifier' =  $O_3$
24. RECEIVE BACnet-SimpleACK-PDU
25. VERIFY Active_COV_Subscriptions =  {{{TD, $P_2$ }, {$O_2$ , Present_Value}, FALSE, (a value less than $L_2$ ),
                                      $INC_2$(a valid Increment if the property is REAL) }}
26. TRANSMIT SubscribeCOV-Request,
        'Subscriber Process Identifier' =  $P_2$ ,
        'Monitored Object Identifier' =  $O_2$
27. RECEIVE BACnet-SimpleACK-PDU
28. VERIFY Active_COV_Subscriptions = { }
```

### 7.3.2.13 Global Group Object Tests

### 7.3.2.13.1 Resizing Group_Member_Names by Writing Group_Members Property Test

Reason for the change: Step 11 Object instance was not present in the test, reference: 135 2020- 12.50.5.2

Purpose: This test case verifies that when the size of the Group_Members array is changed by writing to it, the size of the Group_Member_Names and Present_Value arrays change accordingly and any new entries contain the specified initialized values. If the Group_Members array cannot be written, then this test shall not be performed.

Configuration Requirements: The IUT shall be configured with a Global Group object with a writable Group_Members property.

Test Concept: The Group_Members array is set to a certain size. It is then increased by writing the array size, decreased by writing the array, increased by writing the array and decreased by writing the array size. At each step the size of the Group_Member_Names and Present_Value arrays are verified and the initialized values of the new elements, if any, are checked. *Object1 shall be any Object-Type present in the IUT's Standard Object Types Supported, except object type Global-Group.*

Test Steps:

1. TRANSMIT WriteProperty-Request,
   'Object Identifier' = (the Global Group object being tested),
   'Property Identifier' = Group_Members,
   'Property Array Index' = 0,
   'Property Value' = 2
2. RECEIVE Simple-ACK-PDU
3. VERIFY Group_Members = 2, ARRAY INDEX = 0
4. VERIFY Group_Member_Names = 2, ARRAY INDEX = 0
5. VERIFY Present_Value = 2, ARRAY INDEX = 0
6. TRANSMIT WriteProperty-Request,
   'Object Identifier' = (the Global Group object being tested),
   'Property Identifier' = Group_Members,
   'Property Array Index' = 0,
   'Property Value' = (some value greater than 2)
7. RECEIVE Simple-ACK-PDU
8. VERIFY Group_Members = (the value written in step 6), ARRAY INDEX = 0
9. VERIFY Group_Member_Names = (the value written in step 6), ARRAY INDEX = 0
10. VERIFY Present_Value = (the value written in step 6), ARRAY INDEX = 0
11. VERIFY Group_Members = (a BACnetDeviceObjectPropertyReference containing
    (Device, Instance number 4194303) | *(Object1,4194303)*),
    ARRAY INDEX = (some value from 3 through the value written in step 6)
12. VERIFY Group_Member_Names = (an empty string),
    ARRAY INDEX = (some value from 3 through the value written in step 6)
13. VERIFY Present_Value = 'Access_Result' = PropertyAccessError (PROPERTY, VALUE_NOT_INITIALIZED),
    ARRAY INDEX = (some value from 3 through the value written in step 6)
14. TRANSMIT WriteProperty-Request,
    'Object Identifier' = (the Global Group object being tested),
    'Property Identifier' = Group_Members,
    'Property Value' = (a one-element array containing any valid BACnetDeviceObjectPropertyReference)
15. RECEIVE Simple-ACK-PDU
16. VERIFY Group_Members = 1, ARRAY INDEX = 0
17. VERIFY Group_Member_Names = 1, ARRAY INDEX = 0
18. VERIFY Present_Value = 1, ARRAY INDEX = 0
19. VERIFY Group_Members = (the array written in step 14)
20. TRANSMIT WriteProperty-Request,
    'Object Identifier' = (the Global Group object being tested),
    'Property Identifier' = Group_Members,
    'Property Value' = (an array of two or more valid BACnetDeviceObjectPropertyReference values)
21. RECEIVE Simple-ACK-PDU
22. VERIFY Group_Members = (the size of the array written in step 20), ARRAY INDEX = 0
23. VERIFY Group_Member_Names = (the size of the array written in step 20), ARRAY INDEX = 0
24. VERIFY Present_Value = (the size of the array written in step 20), ARRAY INDEX = 0
25. VERIFY Group_Members = (the array written in step 20)
26. TRANSMIT WriteProperty-Request,
    'Object Identifier' = (the Global Group object being tested),
    'Property Identifier' = Group_Members,
    'Property Array Index' = 0,
    'Property Value' = (some value between 0 and the size of the array written in step 20)
27. RECEIVE Simple-ACK-PDU
28. VERIFY Group_Members = (the size of the array written in step 26), ARRAY INDEX = 0
29. VERIFY Group_Member_Names = (the size of the array written in step 26), ARRAY INDEX = 0
30. VERIFY Present_Value = (the size of the array written in step 26), ARRAY INDEX = 0

### 7.3.2.13.2 Resizing Group_Members by Writing Group_Member_Names Property Test

Reason for the change: Step 11 Object instance was not present in the test, reference: 135 2020- 12.50.5.2
~~Dependencies: WriteProperty Service Execution Tests, 9.22~~

Purpose: This test case verifies that when the size of the Group_Member_Names array is changed by writing to it, the size of the Group_Members and Present_Value arrays change accordingly and any new entries contain the specified initialized values. If the Group_Member_Names array cannot be written, then this test shall not be performed.

Configuration Requirements: The IUT shall be configured with a Global Group object with a writable Group_Member_Names property.

Test Concept: The Group_Member_Names array is set to a certain size. It is then increased by writing the array size, decreased by writing the array, increased by writing the array and decreased by writing the array size. At each step the size of the Group_Members and Present_Value arrays are verified and the initialized values of the new elements, if any, are checked. *Object1 shall be any Object-Type present in the IUT's Standard Object Types Supported, except object type Global-Group.*

Test Steps:

1. TRANSMIT WriteProperty-Request,
   'Object Identifier' = (the Global Group object being tested),
   'Property Identifier' = Group_Member_Names,
   'Property Array Index' = 0,
   'Property Value' = 2
2. RECEIVE Simple-ACK-PDU
3. VERIFY Group_Member_Names = 2, ARRAY INDEX = 0
4. VERIFY Group_Members = 2, ARRAY INDEX = 0
5. VERIFY Present_Value = 2, ARRAY INDEX = 0
6. TRANSMIT WriteProperty-Request,
   'Object Identifier' = (the Global Group object being tested),
   'Property Identifier' = Group_Member_Names,
   'Property Array Index' = 0,
   'Property Value' = (some value greater than 2)
7. RECEIVE Simple-ACK-PDU
8. VERIFY Group_Member_Names = (the value written in step 6), ARRAY INDEX = 0
9. VERIFY Group_Members = (the value written in step 6), ARRAY INDEX = 0
10. VERIFY Present_Value = (the value written in step 6), ARRAY INDEX = 0
11. VERIFY Group_Member_Names = (an empty string),
    ARRAY INDEX = (some value from 3 through the value written in step 6)
12. VERIFY Group_Members = (Device, Instance number 4194303) | *(Object1,4194303)*,
    ARRAY INDEX = (some value from 3 through the value written in step 6)
13. VERIFY Present_Value = 'Access_Result' = PropertyAccessError (PROPERTY, VALUE_NOT_INITIALIZED),
    ARRAY INDEX = (some value from 3 through the value written in step 6)
14. TRANSMIT WriteProperty-Request,
    'Object Identifier' = (the Global Group object being tested),
    'Property Identifier' = Group_Member_Names,
    'Property Value' = (an array of one Character String)
15. RECEIVE Simple-ACK-PDU
16. VERIFY Group_Member_Names = 1, ARRAY INDEX = 0
17. VERIFY Group_Members = 1, ARRAY INDEX = 0
18. VERIFY Present_Value = 1, ARRAY INDEX = 0
19. VERIFY Group_Member_Names = (the array written in step 14)
20. TRANSMIT WriteProperty-Request,
    'Object Identifier' = (the Global Group object being tested),
    'Property Identifier' = Group_Member_Names,
    'Property Value' = (an array of two or more Character Strings)
21. RECEIVE Simple-ACK-PDU
22. VERIFY Group_Member_Names = (the size of the array written in step 20), ARRAY INDEX = 0

```
23.  VERIFY Group_Members = (the size of the array written in step 20), ARRAY INDEX = 0
24.  VERIFY Present_Value = (the size of the array written in step 20), ARRAY INDEX = 0
25.  VERIFY Group_Member_Names = (the array of Character Strings written in step 20)
26.  TRANSMIT WriteProperty-Request,
         'Object Identifier' = (the Global Group object being tested),
         'Property Identifier' = Group_Member_Names,
         'Property Array Index' = 0,
         'Property Value' = (some value between 0 and the size of the array written in step 20)
27.  RECEIVE Simple-ACK-PDU
28.  VERIFY Group_Member_Names = (the size of the array written in step 26), ARRAY INDEX = 0
29.  VERIFY Group_Members = (the size of the array written in step 26), ARRAY INDEX = 0
30.  VERIFY Present_Value = (the size of the array written in step 26), ARRAY INDEX = 0
```

### 7.3.2.13.5 Reliability COMMUNICATION_FAILURE Test

Reason for Change: Reliability property was tested even if it did not exist in the object.

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. 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.

Test Steps:

```
1.   MAKE (M1 discontinue communications)
2.   WAIT (W1)
~~3.   VERIFY Reliability = COMMUNICATION_FAILURE~~
3.   IF (Reliability is present) THEN
4.           VERIFY Reliability = COMMUNICATION_FAILURE
5..  VERIFY Member_Status_Flags = {?, FALSE, ?, ?}
```

### 7.3.2.13.X7 First Stage Faults Take Precedence Over Second Stage Faults When Presenting Reliability

Reason for Change: No longer an internal matter which Reliability is presented, COMMUNICATION_FAILURE takes precedence for all objects, 135-2016bu-7.

Purpose: This test verifies that the Reliability and FAULT flags will reflect a COMMUNICATION_FAILURE, even when the conditions for a MEMBER_FAULT are also present in the Global Group Object.

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. Then force a different member of the Group_Members property to enter a Fault condition and verify the Member_Status_Flags FAULT flag equals TRUE and Reliability equals

COMMUNICATION_FAILURE. The Fault conditions are then removed and reapplied in the inverse order, and the Member_Status_Flags FAULT flag and Reliability properties are verified again.

Configuration Requirements: The IUT shall be configured with a Global Group object, O1, with the Group_Members property containing a member M1 at index N1 that can be made to discontinue communications. O1's Group_Members property shall also contain a member M2 at index N2 that can be made to indicate a fault condition. The IUT begins the test with Reliability equal to NO_FAULT_DETECTED. 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 object to receive an update from M1. W2 is the maximum time it takes for the Global Group object to receive an update from M2. The test steps begin with M1 communicating with O1 and M2 with its FAULT flag cleared.

Test Steps:

```
1.  MAKE (M1 discontinue communications)
2.  WAIT (W1)
3.  IF (Reliability is present) THEN
4.      VERIFY Reliability = COMMUNICATION_FAILURE
5.  VERIFY Member_Status_Flags = {?, FALSE, ?, ?}
6.  MAKE (M2 go into a fault condition)
7.  WAIT (W2)
8.  VERIFY M2.Status_Flags = {?, TRUE, ?, ?}
9.  IF (Reliability is present) THEN
10.     VERIFY Reliability = COMMUNICATION_FAILURE
11. VERIFY Member_Status_Flags = {?, TRUE, ?, ?}
12. MAKE (M1 resume communications)
13. MAKE (M2 come out of the fault condition)
14. WAIT (the greater of W1 and W2)
15. VERIFY M2.Status_Flags = {?, FALSE, ?, ?}
16. If (Reliability is present) THEN
17.     VERIFY Reliability = NO_FAULT_DETECTED
18. MAKE (M2 go into a fault condition)
19. WAIT (W2)
20. VERIFY M2.Status_Flags = {?, TRUE, ?, ?}
21. If (Reliability is present) THEN
22.     VERIFY Reliability = MEMBER_FAULT
23. VERIFY Member_Status_Flags = {?, TRUE, ?, ?}
24. MAKE (M1 discontinue communications)
25. WAIT (W1)
26. If (Reliability is present) THEN
27.     VERIFY Reliability = COMMUNICATION_FAILURE
28. VERIFY Member_Status_Flags = {?, TRUE, ?, ?}
```

### 7.3.2.23 Schedule Object Tests

#### 7.3.2.23.7 List_Of_Object_Property_Reference Internal Test

Reason for Change: Test changed to match the test concept.

Purpose: To verify that the Schedule object writes to objects and properties contained within the IUT.

Test Concept: The Schedule object is configured to write to a property of another object within the same device. The IUT's clock is then set to a time between a pair of scheduled write operations, and verification of the first write operation's data value is performed. The time is advanced to the second time, the Schedule object's Present_Value is checked, and verifications of the write operations are performed. If the IUT does not support writing to object properties within the IUT, then this test shall not be performed.

Configuration Requirements: The IUT is configured with a Schedule object containing a List_Of_Object_Property_ References property that references, if possible, at least one property in another object within the IUT. The Schedule object is configured with either a Weekly_Schedule or an active Exception_Schedule, during a period where Effective_Period is active, with at least two consecutive entries with distinguishable values in the List of BACnetTimeValues, and with no Exception_Schedules at a higher priority. $D_1$ represents the date and time of the first of these two BACnetTimeValues, with corresponding value $V_1$, while $D_2$ and $V_2$ (a value distinguishable from $V_1$) represent the second BACnetTimeValue. A time $D_t$ is defined to occur between $D_1$ and $D_2$.

Test Steps:

1. (TRANSMIT TimeSynchronization-Request, 'Time' = $D_t$) |
   (TRANSMIT UTCTimeSynchronization-Request, 'Time' = ~~$D_1$~~ *$D_t$*) |
   MAKE (the local date and time = $D_t$)
2. WAIT Schedule Evaluation Fail Time
3. VERIFY Present_Value = $V_1$
4. VERIFY (value of referenced property in IUT) = $V_1$
5. (TRANSMIT TimeSynchronization-Request, 'Time' = $D_2$) |
   (TRANSMIT UTCTimeSynchronization-Request, 'Time' = $D_2$) |
   MAKE (the local date and time = $D_2$)
6. WAIT Schedule Evaluation Fail Time
7. VERIFY Present_Value = $V_2$
8. VERIFY (value of referenced property in IUT) = $V_2$

**7.3.2.23.8 List_Of_Object_Property_Reference External Test**

Reason for Change: Test changed to match the test concept.

Purpose: To verify that the Schedule object writes to objects and properties contained within the IUT.

Test Concept: The Schedule object is configured to write to a property of another object within the same device. The IUT's clock is then set to a time between a pair of scheduled write operations, and verification of the first write operation's data value is performed. The time is advanced to the second time, the Schedule object's Present_Value is checked, and verifications of the write operations are performed. If the IUT does not support writing to object properties within the IUT, then this test shall not be performed.

Configuration Requirements: The IUT is configured with a Schedule object containing a List_Of_Object_Property_ References property that references, if possible, at least one property in another object within the IUT. The Schedule object is configured with either a Weekly_Schedule or an active Exception_Schedule, during a period where Effective_Period is active, with at least two consecutive entries with distinguishable values in the List of BACnetTimeValues, and with no Exception_Schedules at a higher priority. $D_1$ represents the date and time of the first of these two BACnetTimeValues, with corresponding value $V_1$, while $D_2$ and $V_2$ (a value distinguishable from $V_1$) represent the second BACnetTimeValue. A time $D_t$ is defined to occur between $D_1$ and $D_2$.

Test Steps:

1. (TRANSMIT TimeSynchronization-Request, 'Time' = ~~$D_1$~~ *$D_t$*) |
   (TRANSMIT UTCTimeSynchronization-Request, 'Time' = ~~$D_1$~~ *$D_t$*) |
   MAKE (the local date and time = ~~$D_1$~~ *$D_t$*)
2. WAIT Schedule Evaluation Fail Time
3. VERIFY Present_Value = $V_1$
4. VERIFY (value of referenced property in IUT) = $V_1$
5. (TRANSMIT TimeSynchronization-Request, 'Time' = $D_2$) |
   (TRANSMIT UTCTimeSynchronization-Request, 'Time' = $D_2$) |

```
        MAKE (the local date and time = $D_2$)
6.  WAIT Schedule Evaluation Fail Time
7.  VERIFY Present_Value = $V_2$
8.  VERIFY (value of referenced property in IUT) = $V_2$
```

### 7.3.2.23.X1 Write_Every_Scheduled_Action TRUE Test

Reason for Change: There is no test for this functionality.

Purpose: To verify the functionality of the Write_Every_Scheduled_Action property of the Schedule object when the value is TRUE.

Test Concept: The IUT is configured as specified in Configuration Requirements. The members of the List_Of_Object_Property_References are written to V1 by setting the IUT's clock to D1. The clock is advanced to D2 and writes of V1 to the members are verified. The members of the List_Of_Object_Property_References are written to the value of the Schedule_Default property by setting the IUT's clock to D3. The clock is advanced to D4 and writes to the members with the value of Schedule_Default is verified.

Configuration Requirements: The IUT shall be configured with a Schedule object (S1) containing a List_Of_Object_Property_References property that contains members internal to the IUT and, if supported, external members. The Schedule object shall be configured with Write_Every_Scheduled_Action equal to TRUE, the Effective_Period is active, and the Weekly Schedule or one or more Exception_Schedules are configured to create two consecutive BACnetTimeValues (D1 and D2) to occur that contain the same value (V1). The Weekly Schedule or an Exception_Schedule is configured with two consecutive BACnetTimeValues (D3 and D4) with D3 containing the same value as Schedule_Default and D4 containing a NULL such that the Schedule_Default comes into effect. PFW is the value of S1's Priority_For_Writing property.

Test Steps:

```
-- Set the value of all members to V1
1.  (TRANSMIT TimeSynchronization-Request, 'Time' = D1) |
        (TRANSMIT UTCTimeSynchronization-Request, 'Time' = D1) |
        MAKE (the local date and time = D1)
2.  BEFORE Schedule Evaluation Fail Time {
3.      REPEAT X = (every member in List_Of_Object_Property_References) DO {
4.          IF (X.device-identifier is present AND X.device-identifier <> IUT) THEN
5.              RECEIVE WriteProperty-Request,
                    'Object Identifier' =       (X.object-identifier),
                    'Property Identifier' =     (X.property-identifier),
                    'Property Value' =          V1,
                    'Priority' =                (PFW)
6.              TRANSMIT BACnet-SimpleACK-PDU
            ELSE
7.              VERIFY X.object-identifier, X.property-identifier = V1
        }
    }

-- Test the same value is written to all internal and external members
8.  (TRANSMIT TimeSynchronization-Request, 'Time' = D2) |
        (TRANSMIT UTCTimeSynchronization-Request, 'Time' = D2) |
        MAKE (the local date and time = D2)
9.  BEFORE Schedule Evaluation Fail Time {
10.     REPEAT X = (every member in List_Of_Object_Property_References) DO {
11.         IF (X.device-identifier is present AND X.device-identifier <> IUT) THEN
```

```
12.                 RECEIVE WriteProperty-Request,
                        'Object Identifier' =       (X.object-identifier),
                        'Property Identifier' =     (X.property-identifier),
                        'Property Value' =          V1,
                        'Priority' =                (PFW)
13.                 TRANSMIT BACnet-SimpleACK-PDU
            ELSE
14.                 IF (X.property-identifier = Present-Value and X.object-identifier contains a Priority_Array) THEN
15.                     VERIFY X.object-identifier, Priority_Array = V1, ARRAY INDEX = PFW
                    ELSE
16.                     VERIFY X.object-identifier, X.property-identifier = V1
        }
    }

-- Set the value of all members to the value of Schedule_Default
17. (TRANSMIT TimeSynchronization-Request, 'Time' = D3) |
        (TRANSMIT UTCTimeSynchronization-Request, 'Time' = D3) |
        MAKE (the local date and time = D3)
18. BEFORE Schedule Evaluation Fail Time {
19.     REPEAT X = (every member in List_Of_Object_Property_References) DO {
20.         IF (X.device-identifier is present OR X.device-identifier <> IUT) THEN
21.                 RECEIVE WriteProperty-Request,
                        'Object Identifier' =       (X.object-identifier),
                        'Property Identifier' =     (X.property-identifier),
                        'Property Value' =          Schedule_Default,
                        'Priority' =                (PFW)
22.                 TRANSMIT BACnet-SimpleACK-PDU
            ELSE
23.                 IF (X.property-identifier = Present-Value and X.object-identifier contains a Priority_Array) THEN
24.                     VERIFY X.object-identifier, Priority_Array = Schedule_Default, ARRAY INDEX = PFW
                    ELSE
25.                     IF (Schedule_Default <> NULL) THEN
26.                       VERIFY X.object-identifier, X.property-identifier = Schedule_Default
        }
    }

-- Test that when Schedule_Default comes into effect, it is written to all internal and external members
27. (TRANSMIT TimeSynchronization-Request, 'Time' = D4) |
        (TRANSMIT UTCTimeSynchronization-Request, 'Time' = D4) |
        MAKE (the local date and time = D4)
28. BEFORE Schedule Evaluation Fail Time {
29.     REPEAT X = (every member in List_Of_Object_Property_References) DO {
30.         IF (X.device-identifier is present OR X.device-identifier <> IUT) THEN
31.                 RECEIVE WriteProperty-Request,
                        'Object Identifier' =       (X.object-identifier),
                        'Property Identifier' =     (X.property-identifier),
                        'Property Value' =          Schedule_Default,
                        'Priority' =                (PFW)
32.                 TRANSMIT BACnet-SimpleACK-PDU
            ELSE
33.                 IF (X.property-identifier = Present-Value and X.object-identifier contains a Priority_Array) THEN
```

```
34.                 VERIFY X.object-identifier, Priority_Array = Schedule_Default, ARRAY INDEX
= PFW
            ELSE
35.                 IF (Schedule_Default <> NULL) THEN
36.                     VERIFY X.object-identifier, X.property-identifier = Schedule_Default
        }
    }
```

**7.3.2.23.X2 Write_Every_Scheduled_Action FALSE Test**

Reason for Change: There is no test for this functionality.

Purpose: To verify that when Write_Every_Scheduled_Action property is FALSE or not present, the schedule does not write to members of the List_Of_Object_Property_References when the Present_Value does not change when a new time-value pair comes into effect.

Test Concept: The IUT's Schedule object is configured to write to internal and, if supported, external objects. The Schedule object is configured with two sequential dates and times (D1 and D2) that contain the same value (V1). To set the value of all members to V1, the IUT's clock is set to D1, and the members are verified to contain V1. The clock is advanced to D2, and it is checked that no writes occurred.

Configuration Requirements: The IUT shall be configured with a Schedule object (S1) containing a List_Of_Object_Property_References property that contains members internal to the IUT and, if supported, external members. The Schedule object shall be configured with Write_Every_Scheduled_Action equal to FALSE, if present, the Effective_Period is active, and the Weekly Schedule or one or more Exception_Schedules are configured to create two consecutive BACnetTimeValues (D1 and D2) to occur that contain the same value (V1). PFW is the value of S1's Priority_For_Writing property.

Test Steps:

```
-- Set the value of all members to V1
1.  (TRANSMIT TimeSynchronization-Request, 'Time' = D1) |
        (TRANSMIT UTCTimeSynchronization-Request, 'Time' = D1) |
        MAKE (the local date and time = D1)
2.  BEFORE Schedule Evaluation Fail Time {
3.      REPEAT X = (every member in List_Of_Object_Property_References) DO {
            -- verify internal and external members are equal to V1
4.          IF (X.property-identifier = Present-Value and X.object-identifier contains a Priority_Array)
THEN
5.              VERIFY X.object-identifier, Priority_Array = V1, ARRAY INDEX = PFW
            ELSE
6.              VERIFY X.object-identifier, X.property-identifier = V1
        }
    }

-- Test the same value is not written to any external members
7.  (TRANSMIT TimeSynchronization-Request, 'Time' = D2) |
        (TRANSMIT UTCTimeSynchronization-Request, 'Time' = D2) |
        MAKE (the local date and time = D2)
8.  BEFORE Schedule Evaluation Fail Time {
9.      REPEAT X = (every member in List_Of_Object_Property_References) DO {
10.         IF (X.device-identifier is present OR X.device-identifier <> IUT) THEN
11.             CHECK (the IUT does not initiate a Write_Property request to X)
        }
    }
```

### 7.3.2.24 Logging Object Tests

### 7.3.2.24.13 Log-Status Test

Reason for Change: The test here supersedes the version in 135.1-2023, with a completely different, less prescriptive approach.

~~Dependencies: ReadRange Service Execution Tests, 9.21; WriteProperty Service Execution Tests, 9.18.~~

~~BACnet Reference Clause: 12.25.14*, 12.27.13, 12.30.19*~~

Purpose: To verify proper logging of log-disabled*,* ~~and~~ buffer-purged*, and log-interrupted* 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.~~

*Logging is disabled and the log buffer is purged by writing Record_Count = 0. The Log_Buffer is then checked to verify it has a single record and it is a buffer-purged log entry. Logging is then enabled and disabled and the Log_Buffer is checked to verify the log-disabled events were logged. Logging is enabled and a power cycle (or some other vendor specified action that will generate a log_interrupted entry) is performed on the IUT. After the IUT restarts, the Log_Buffer is checked to verify that a log-interrupted event was logged.*

~~Test Configuration:~~ *Configuration Requirements:* 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’ = O1,~~
~~‘Property Identifier’ = Log_Buffer,~~
~~‘Reference Index’ = 1,~~
~~‘Count’ = 1~~
~~5. RECEIVE ReadRange-Ack~~
~~‘Object Identifier’ = O1,~~
~~‘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’ = O1,~~
~~‘Property Identifier’ = Log_Buffer,~~
~~‘Reference Index’ = 1,~~
~~‘Count’ = 2~~
~~9. RECEIVE ReadRangeAck~~

~~'Object Identifier' = O1,~~
~~'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' = O1,~~
~~'Property Identifier' = Log_Buffer,~~
~~'Reference Time' = (2154-12-31, 23:59:59.99),~~
~~'Count' = -1~~
~~11. RECEIVE ReadRangeAck~~
~~'Object Identifier' = O1,~~
~~'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)*
*7. IF (Protocol_Revision >=7) THEN*
*8. WRITE Enable = TRUE*
*9. MAKE (power cycle the IUT or take some other vendor specified action as required to generate a Log_Interrupted entry)*
*10. VERIFY (Log_Buffer contains an entry for the log-interrupted event)*

### 7.3.2.30 Notification Forwarder Object Tests

### 7.3.2.30.6 Out_Of_Service Property Test

Reason for Change: Corrected inconsistencies in test steps with Base Setup 2.

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.

Configuration Requirements: The selected object is configured such that its Out_Of_Service shall be set to FALSE and Reliability set to NO_FAULT_DETECTED. 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_ID~~2~~, -- Recipient ~~D2~~*D1*
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)
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~~
13.     RECEIVE DESTINATION = D1, UnconfirmedEventNotification-Request
~~          RECEIVE DESTINATION = D2, UnconfirmedEventNotification-Request~~
14. MAKE (Out_Of_Service = TRUE)
15. VERIFY Out_Of_Service = TRUE
16. VERIFY Status_Flags = (FALSE, FALSE, FALSE, TRUE)
17. IF (Reliability is writable) THEN
18.     REPEAT X = (all values of the Reliability enumeration appropriate to the object type except
                    NO_FAULT_DETECTED) DO {
19.         WRITE Reliability = X
20.         VERIFY Reliability = X
21.         VERIFY Status_Flags =(?, TRUE, ?, TRUE)
22.         WRITE Reliability = NO_FAULT_DETECTED
23.         VERIFY Reliability = NO_FAULT_DETECTED
24.         VERIFY Status_Flags = (?, FALSE, ?, TRUE)
        }
25. IF (Out_Of_Service is writable) THEN
26.     WRITE Out_Of_Service = FALSE
    ELSE
27.     MAKE (Out_Of_Service = FALSE)
28. VERIFY Out_Of_Service = FALSE
29. VERIFY Status_Flags = ( ?, FALSE, ?, FALSE)
```

### 7.3.2.39 Lighting Output Object Tests

### 7.3.2.39.X1 Tracking_Value Clamping Test

Reason for Change: No test exists for this functionality.

Purpose: To ensure that Tracking_Value is properly clamped when Present_Value is written to a value above High_End_Trim or below Low_End_Trim and In_Progress is equal TRIM_ACTIVE.

Test Concept: Throughout this test, the Present_Value shall be written at a priority between 3 and 16 and is the active priority. If High_End_Trim is present, write the Present_Value to a value greater than High_End_Trim. Verify In_Progress is equal to TRIM_ACTIVE, Track_Value is equal to the High_End_Trim, and Present_Value is equal to the value written. If Low_End_Trim is present, write the Present_Value to a value less than Low_End_Trim. Verify In_Progress is equal to TRIM_ACTIVE, Track_Value is equal to the Low_End_Trim, and Present_Value is equal to the value written.

Test Configuration: The Lighting Output object, O1, shall be configured such that In_Progress is IDLE and no processes are writing to the Present_Value. The High_End_Trim, if present, shall be less than 99. The Low_End_Trim, if present, shall be greater than 2.

Test Steps:

```
1.  VERIFY In_Progress = IDLE
2.  IF (High_End_Trim is present) THEN {
3.      READ HET = High_End_Trim
4.      WRITE Present_Value = (PV, a valid value > HET)
5.      WHILE (In_Progress <> TRIM_ACTIVE) {}
6.      VERIFY Tracking_Value = HET
7.      VERIFY Present_Value = PV
    }
8.  IF (Low_End_Trim is present) THEN {
9.      READ LET = Low_End_Trim
10.     IF (In_Progress = TRIM_ACTIVE) THEN {
11.         WRITE Present_Value = (PV, a valid value < HET and > LET)
12.         WHILE (In_Progress <> IDLE) {}
        }
13.     WRITE Present_Value = (PV, a valid value < LET)
14.     WHILE (In_Progress <> TRIM_ACTIVE) {}
15.     VERIFY Tracking_Value = LET
16.     VERIFY Present_Value = PV
    }
```

### 7.3.2.39.X2 Priority 1 and 2 Clamping Test

Reason for Change: No test exists for this functionality.

Purpose: To verify that Tracking_Value is not clamped to High_End_Trim or Low_End_Trim when Present_Value is written at priorities 1 and 2.

Test Concept: Throughout this test, the Present_Value shall be written at a priority 1 and 2 and is the active priority. If High_End_Trim is present, write the Present_Value to a value greater than High_End_Trim. Verify In_Progress is equal to IDLE and Track_Value and Present_Value are equal to the value written. If Low_End_Trim is present, write the Present_Value to a value less than Low_End_Trim. Verify In_Progress is equal to IDLE and Track_Value and Present_Value are equal to the value written.

Test Configuration: The Lighting Output object, O1, shall be configured such that In_Progress is IDLE and no processes are writing to the Present_Value. The High_End_Trim, if present, shall be less than 99. The Low_End_Trim, if present, shall be greater than 2.

Test Steps:

```
1.  VERIFY In_Progress = IDLE
2.  IF (High_End_Trim is present) THEN {
3.      READ HET = High_End_Trim
4.      WRITE Present_Value, = (PV, a valid value > HET), PRIORITY = 1
5.      WHILE (In_Progress <> IDLE) {}
6.      VERIFY Tracking_Value = PV
7.      VERIFY Present_Value = PV
    }
8.  IF (Low_End_Trim is present) THEN {
9.      READ LET = Low_End_Trim
10.     WRITE Present_Value, = (NULL), PRIORITY = 1
11.     WHILE (In_Progress <> IDLE) {}
12.     WRITE Present_Value = (PV, a valid value < LET), PRIORITY = 2
13.     WHILE (In_Progress <> IDLE) {}
14.     VERIFY Tracking_Value = PV
15.     VERIFY Present_Value = PV
    }
```

### 7.3.2.39.X3 Trim_Fade_Time Test

Reason for Change: No test exists for this functionality.

Purpose: This test verifies the IUT's Lighting Output object will fade using Trim_Fade_Time when the High_End_Trim or Low_End_Trim are changed such that the current value of Present_Value is outside of the Operating Range.

Test Concept: The Present_Value is made to be within the Operating Range. If the High_End_Trim property is present, it is changed to be a lower value such that Present_Value is now outside the Operating range and it is verified that it takes Trim_Fade_Time milliseconds for the Tracking_Value to fade to the new High_End_Trim value. The same steps are repeated using Low_End_Trim, if present.

Configuration Requirements: The IUT shall not be performing any fades at the start of this test and Present_Value shall be within the Operating Range. Default_Fade_Time and Trim_Fade_Time shall be configured to different values, if possible.

Test Steps:

```
1.  VERIFY In_Progress = IDLE
2.  READ DFT = Default_Fade_Time
3.  READ TFT = Trim_Fade_Time
4.  IF (High_End_Trim is present) THEN {
5.      READ HET1 = High_End_Trim
6.      WRITE Present_Value = HET1
7.      WRITE High_End_Trim = (a new value HET2, such that HET2 < PV)
8.      IF (TFT <= DFT) THEN {
9.          WAIT (TFT milliseconds)
10.         VERIFY Tracking_Value = HET2
11.         VERIFY In_Progress = TRIM_ACTIVE
12.         VERIFY Present_Value = HET1
        }
13.     IF (TFT > DFT) THEN {
```

```
14.             WAIT (DFT milliseconds)
15.             VERIFY Tracking_Value <> HET2
16.             WAIT (TFT - DFT milliseconds)
17.             VERIFY Tracking_Value = HET2
18.             VERIFY In_Progress = TRIM_ACTIVE
19.             VERIFY Present_Value = HET1
            }
        -- Reset test setup in case of Low_End_Trim being present
20.         WRITE Present_Value = HET2
21.         VERIFY In_Progress = IDLE
    }
22. IF (Low_End_Trim is present) THEN {
23.         READ LET1 = Low_End_Trim
24.         WRITE Present_Value = LET1
25.         WRITE Low_End_Trim = (a new value LET2, such that LET2 > PV)
26.         IF (TFT <= DFT) THEN {
27.             WAIT (TFT milliseconds)
28.             VERIFY Tracking_Value = HET2
29.             VERIFY In_Progress = TRIM_ACTIVE
30.             VERIFY Present_Value = HET1
            }
31.         IF (TFT > DFT) THEN {
32.             WAIT (DFT milliseconds)
33.             VERIFY Tracking_Value <> HET2
34.             WAIT (TFT - DFT milliseconds)
35.             VERIFY Tracking_Value = HET2
36.             VERIFY In_Progress = TRIM_ACTIVE
37.             VERIFY Present_Value = HET1
            }
    }
```

### 7.3.2.40 Access Door Object Tests

#### 7.3.2.40.6 Door_Unlock_Delay_Time Test

Reason for Change: Step 22 required the door to be in the incorrect state.

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
4.      IF (Lock_Status is present) THEN
5.          VERIFY Lock_Status = LOCKED
6.      CHECK (that the door control output is in a state that would cause the door to be locked)

7.  IF (Lock_Status is present) THEN
8.      VERIFY Lock_Status = UNLOCKED
9.  CHECK (that the door control output is in a state that would cause the door to be unlocked)
10. WAIT (Door_Pulse_Time)
11. VERIFY Present_Value = LOCK
12. IF (Lock_Status is present) THEN
13.     VERIFY Lock_Status = LOCKED
14.. CHECK (that the door control output is in a state that would cause the door to be locked)

-- Test EXTENDED_PULSE_UNLOCK
15. WRITE Present_Value = EXTENDED_PULSE_UNLOCK
16. WAIT (Internal Processing Fail Time)
17. BEFORE Door_Unlock_Delay_Time
18.     IF (Lock_Status is present) THEN
19.         VERIFY Lock_Status = LOCKED
20.     CHECK (that the door control output is in a state that would cause the door to be locked)

21. IF (Lock_Status is present) THEN
22.     VERIFY Lock_Status = UNLOCKED
23. CHECK (that the door control output is in a state that would cause the door to be unlocked)
24. WAIT (Door_Extended_Pulse_Time)
25. VERIFY Present_Value = LOCK
26. IF (Lock_Status is present) THEN
27.     VERIFY Lock_Status = LOCKED
28. CHECK (that the door control output is in a state that would cause the door to be locked)

-- Test UNLOCK
29. WRITE Present_Value = UNLOCK
30. WAIT (Internal Processing Fail Time)
31. IF (Lock_Status is present) THEN
32      VERIFY Lock_Status = UNLOCKED
33. CHECK (that the door control output is in a state that would cause the door to be ~~locked~~ *unlocked*)
```

### 7.3.2.44 Access Credential Object Tests

#### 7.3.2.44.X1 Absentee Limit Property Test with Zero and 65535 Value

Reason for Change: No test exists for this functionality.  This test is not in any SSPC proposal.

Purpose: To verify the absentee limit credential behaviors for values of 0 and 65535.

Configuration Requirements:
See 7.3.2.*44*~~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 > 0.
d) Last_Use_Time shall be set to a valid date and time.

Test Steps:

```
1.  VERIFY Credential_Status = ACTIVE
2.  VERIFY Reason_For_Disable = (empty list)
3.  MAKE (Absentee_Limit = 0)
4.  VERIFY Absentee_Limit = 0
5.  READ T1 = Local_Date
6.  (TRANSMIT TimeSynchronization-Request,
        'Time' = (T1 + 1 days)) |
    (TRANSMIT UTCTimeSynchronization-Request,
        'Time' = (T1 + 1 days)) |
    MAKE (Local_Date = T1 + 1 days)
7.  VERIFY Credential_Status = INACTIVE
8.  VERIFY Reason_For_Disable = (DISABLED_INACTIVITY)
9.  MAKE (Absentee_Limit = 65535)
10. VERIFY Absentee_Limit = 65535
11. MAKE Credential_Status = ACTIVE
12. VERIFY Credential_Status = ACTIVE
13. READ T1 = Local_Date
14. (TRANSMIT TimeSynchronization-Request,
        'Time' = (T1 + 2 days)) |
    (TRANSMIT UTCTimeSynchronization-Request,
        'Time' = (T1 + 2 days)) |
    MAKE (Local_Date = T1 + 2 days)
15. VERIFY Credential_Status = ACTIVE
```

### 7.3.2.46 Network Port Object Tests

### 7.3.2.46.1 Network Port Configuration Tests

### 7.3.2.46.1.3 Network Port Non-Volatility Properties Test

Reason for Change: Modify the test to save changes via normal means before verifying the Power Cycle keeps the changes.

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 {
2.      WRITE P = (a new value different from the property’s current value)
    }
3.  IF (Changes_Pending is TRUE) ~~any of the properties utilize the pending change functionality~~ THEN
~~        VERIFY Changes_Pending = TRUE~~
4.      TRANSMIT ReinitializeDevice-Request
```

```
            'Reinitialized State of Device' = WARMSTART | ACTIVATE_CHANGES
            'Password' = (any valid password)
5.      RECEIVE BACnet-SimpleACK-PDU
6.      MAKE(reconfigure the TD and other devices on the network to the new network settings)
7.      WAIT Activate Changes Fail Time
~~    ELSE~~
8.      VERIFY Changes_Pending = FALSE
9.  REPEAT P = P1 ... PN {
10.     VERIFY P = (the new value for the property)
    }
11. MAKE (the IUT power cycle~~ to reinitialize~~)
12. REPEAT P = P1 ... PN {
13.     VERIFY P = (the new value for the property)
14.     CHECK (that the value for P is in use by the network port)
    }
```

### 7.3.2.46.1.X1 Network Port Object Not Writable Property Test

Reason for Change: New test for read-only NPO properties.

Purpose: This test verifies that writes to read-only properties of a Network Port object do not affect the Changes_Pending property and, if one is defined, contains the prescribed default value.

Test Concept: The properties of a Network Port object, NPO1, are written and rejected and the Changes_Pending is checked.

Configuration Requirements: P1 through PN are Network Port properties that are supported by the Network_Type and Protocol_Level and are not writable in the IUT.

Test Steps:

```
1. VERIFY Changes_Pending = FALSE
2. REPEAT P = P1 ... PN {
3.  TRANSMIT WriteProperty-Request,
        'Object Identifier' = NPO1,
        'Property Identifier' = P,
        'Property Value' = (any valid value)
4.  RECEIVE BACnet-Error-PDU
        'Error Class' = PROPERTY,
        'Error Code' = WRITE_ACCESS_DENIED
5.  CHECK (P = prescribed default value)
    }
6. VERIFY Changes_Pending = FALSE
```

### 7.3.2.46.3 Network Port Command Tests

### 7.3.2.46.3.2 DISCARD_CHANGES Command Tests

### 7.3.2.46.3.2.X1 DISCARD_CHANGES Command Test

Reason for change: No change, renumbered clause only and changed test step numbering. This was needed to add the DISCARD_CHANGES failure 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.

Configuration Requirements: 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) DO {
2.        V[I] = READ P[I]
3.        WRITE P[I] = (a value different than V[I], if possible)
    }

-- discard the changes
4.  WRITE Command = DISCARD_CHANGES
5.  WAIT Activate Changes Fail Time

-- verify that no changes are pending any more
6.  VERIFY Changes_Pending = FALSE
7.  VERIFY Command = IDLE

-- verify that the properties have reverted in value, and that the old value remains in use by the port
8.  REPEAT I = (in the range 1 through the number of properties being written) DO {
9.        VERIFY P[I] = V[I]
10.       CHECK(the value V[I] is in use by the network port)
    }

-- command the device to activate any changes which should have no effect
11. TRANSMIT ReinitializeDevice-Request
          'Reinitialized State of Device' = WARMSTART | ACTIVATE_CHANGES
          'Password' = (any valid password)
12. RECEIVE BACnet-SimpleACK-PDU
13. MAKE(reconfigure the TD and other devices on the network to the new network settings)
14. WAIT Activate Changes Fail Time
15. VERIFY Command = IDLE

-- verify that the properties retain their original values, and that that value remains in use by the port
16. REPEAT I = (in the range 1 through the number of properties being written) DO {
17.       VERIFY P[I] = V[I]
18.       CHECK(the value V[I] is in use by the network port)
    }
```

### 7.3.2.46.3.2.X2 DISCARD_CHANGES Command Failure Test

Reason for change: No test existed.

Purpose: To verify that Network Port object responds to DISCARD_CHANGES commands when the command is not supported.

Test Concept: Attempt to command a Network Port which does not support the DISCARD_CHANGES. Verify that the attempt fails with an Error Class of PROPERTY and an error code of VALUE_OUT_OF_RANGE.

Configuration Requirements: Select a Network Port which supports writable properties that set the Changes_Pending property to TRUE.

Test Steps:

```
1.  TRANSMIT WriteProperty-Request,
        'Object Identifier' =   (the Network Port object),
        'Property Identifier' = (any writable property that results in Changes_Pending = TRUE),
        'Property Value' =      (any valid value)
2.  RECEIVE BACnet-SimpleACK-PDU
3.  TRANSMIT WriteProperty-Request,
        'Object Identifier' =   (the Network Port object),
        'Property Identifier' = Command,
        'Property Value' =      DISCARD_CHANGES,
4.  RECEIVE BACnet-Error-PDU
        'Error Class' =   PROPERTY,
        'Error Code' =    VALUE_OUT_OF_RANGE
5.  VERIFY Command = IDLE
```

### 7.3.2.46.3.2.X3 DISCARD_CHANGES Command With File Object References Test

Reason for Change: New test per Addendum 135-2020cc Clauses 12.56.Y24 and 12.56.Y25.

Purpose: To verify that the Network Port object and linked File objects discard pending changes when the Command DISCARD_CHANGES is received.

Test Concept: Write the File object linked to the Network Port object, verify the write was successful, write DISCARD_CHANGES to the Command property of the Network Port object, and verify that the File object and Network Port object properties revert to their previous values. Repeat the test writing the File object, File_Size to zero.

Configuration Requirements: Execute this test on a Network Port object, NP1, with Network_Type = SECURE_CONNECT and supports the DISCARD_CHANGES command. F1 is the File object referenced by a property of NP1. When performing the AtomicWriteFile service, a Maximum Write Data Length (MWDL) shall be calculated before starting the test. MWDL shall be 21 octets less than the minimum of the TD’s maximum transmittable APDU length and the IUT’s Max_APDU_Length_Accepted.

Test Steps:

```
-- write to the File object
1.  VERIFY NP1, Changes_Pending = FALSE
2.  VERIFY NP1, Command = IDLE
3.  READ FS1 = F1, File_size
4.  READ MD1 = F1, Modification_Date
5.  REPEAT Z = (0 through the file size, FS2, in increments of MWDL) DO {
6.      TRANSMIT AtomicWriteFile-Request
            ‘File Identifier’ = F1
            ‘File Start Position’ = Z
            ‘Record Data’ = (any valid file content, the number of octets being the lesser of
                (file size – Z) and MWDL)
7.      RECEIVE AtomicWriteFile-ACK
            ‘File Start Position’ = Z
    }
8.  VERIFY NP1, Changes_Pending = TRUE
9.  VERIFY F1, File_Size = FS2
10. VERIFY F1, Modification_Date = (the current local date and time)
```

```
-- discard changes
11.  WRITE NP1, Command = DISCARD_CHANGES
12.  WAIT Activate Changes Fail Time

-- verify the Network Port object was successfully reverted
13.  VERIFY NP1, Changes_Pending = FALSE
14.  VERIFY NP1, Command = IDLE

-- verify the File object was successfully reverted
15.  VERIFY F1, File_Size = FS1
16.  VERIFY F1, Modification_Date = (MD1 or the current local date and time)
17.  VERIFY F1, Archive = FALSE

-- write File_Size = 0
18.  WRITE F1, File_Size = 0
19.  VERIFY NP1, Changes_Pending = TRUE

-- verify the File object was successfully written
20.  VERIFY F1, File_Size = 0
21.  VERIFY F1, Modification_Date = (the current local date and time)

-- discard changes
22.  WRITE NP1, Command = DISCARD_CHANGES
23.  WAIT Activate Changes Fail Time

-- verify the Network Port object was successfully reverted
24.  VERIFY NP1, Changes_Pending = FALSE
25.  VERIFY NP1, Command = IDLE

-- verify the File object was successfully reverted
26.  VERIFY F1, File_Size = FS1
27.  VERIFY F1, Modification_Date = (MD1 or the current local date and time)
28.  VERIFY F1, Archive = FALSE
```

**7.3.2.46.3.9 No Commands if Changes_Pending Test**

Reason for Change: Modified per Addendum 135-2020cc-1.

Purpose: To verify that the Network Port disallows commands, except DISCARD_CHANGES *and VALIDATE_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.

Configuration Requirements: Execute the test on a Network Port object which supports the Command property.

Test Steps:

```
-- write some properties
1.   REPEAT P = (P1 .. Px) DO{
2.        WRITE NP, P = (any valid value)
     }

-- verify that changes are pending
3.   VERIFY Changes_Pending = TRUE
```

```
-- write each supported Command value, except DISCARD_CHANGES *and VALIDATE_CHANGES*
4.  REPEAT CMD = (all ~~non-IDLE~~ valid values that NP supports except DISCARD_CHANGES *and*
            *VALIDATE_CHANGES*) *DO* {
5.      TRANSMIT WriteProperty-Request
            'Object Identifier' = NP
            'Property' = Command,
            'Property Value' = CMD
6.      RECEIVE BACnet-Error-PDU
            'Error Class' =         PROPERTY,
            'Error Code' =          INVALID_VALUE_IN_THIS_STATE
    }

-- revert the Network Port object
7.  IF the IUT supports DISCARD_CHANGES THEN
8.      WRITE Command = DISCARD_CHANGES
    ELSE
9.      MAKE (the IUT discard its changes)
```

### 7.3.2.46.3.X Certificate Configuration Tests

### 7.3.2.46.3.X.1 GENERATE_CSR_FILE Command Test

Reason for Change: New test per Addendum 135-2020cc-1. Added File object File_Size and Modification_Date checks.

Purpose: To verify that the Network Port object generates a new CSR file when commanded to.

Test Concept: Using a Network Port object, NP1, which supports the GENERATE_CSR_FILE command, the port is commanded to GENERATE_CSR_FILE. Test the referenced CSR file has been updated using the Modification_Date property.

Configuration Requirements: Execute the test on a Network Port object which supports the Command property and property Changes_Pending = FALSE.

Test Steps:

```
1.  WRITE NP1, Command = GENERATE_CSR_FILE
2.  WHILE (NP1, Command <> IDLE) DO {}
3.  VERIFY Changes_Pending = FALSE
4.  READ CSR = NP1, Certificate_Signing_Request_File
5.  VERIFY CSR, Modification_Date = (the current local date and time)
6.  VERIFY CSR, File_Size > 0
```

### 7.3.2.46.3.X.2 GENERATE_CSR_FILE Command Failure Test

Reason for Change: New test per Addendum 135-2020cc-1.

Purpose: To verify that Network Port objects respond to the GENERATE_CSR_FILE 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 GENERATE_CSR_FILE. Verify that the correct error code is returned.

Configuration Requirements: property Changes Pending = FALSE.

Test Steps:

```
1.  TRANSMIT WriteProperty-Request,
        'Object Identifier' =    (the Network Port object),
        'Property Identifier' =  Command,
        'Property Value' =       GENERATE_CSR_FILE
2.  RECEIVE BACnet-Error-PDU
        'Error Class' =          PROPERTY,
        'Error Code' =           OPTIONAL_FUNCTIONALITY_NOT_SUPPORTED
```

### 7.3.2.46.3.X.3 VALIDATE_CHANGES Command Test

Reason for Change: New test per Addendum 135-2020cc-1.

Purpose: To verify that the Network Port attempts to perform the required validations on this property when commanded to.

Test Concept: Starting with a Network Port object which supports the VALIDATE_CHANGES command. The port is commanded to VALIDATE_CHANGES. This command shall initiate a validation of the values of the properties of this port as specified in each property. If a property is present but not used, based on the Network_Type, it shall not be validated. The value of the Command_Validation_Result property shall be updated to indicate the validation result.

Test Steps:

```
1.  READ V1 = Command_Validation_Result
2.  READ CP = Changes_Pending
-- request a VALIDATE_CHANGES command, and wait for it to timeout
3.  WRITE Command = VALIDATE_CHANGES
4.  VERIFY Changes_Pending = CP
5.  WHILE (Command <> IDLE) DO {}
6.  VERIFY Command_Validation_Result = Any value different from V1
```

### 7.3.2.46.3.X.4 VALIDATE_CHANGES Command Failure Test

Reason for Change: New test per Addendum 135-2020cc-1.

Purpose: To verify that Network Port objects respond to the VALIDATE_CHANGES 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 VALIDATE_CHANGES. Verify that the correct error code is returned.

Test Steps:

```
1.  TRANSMIT WriteProperty-Request,
        'Object Identifier' =    (the Network Port object),
        'Property Identifier' =  Command,
        'Property Value' =       VALIDATE_CHANGES
2.  RECEIVE BACnet-Error-PDU
        'Error Class' =          PROPERTY,
        'Error Code' =           OPTIONAL_FUNCTIONALITY_NOT_SUPPORTED
```

### 7.3.2.46.4 Hierarchical Network Port Tests

#### 7.3.2.46.4.1 Valid Hierarchy Test

Reason for Change: The test no longer needs to test all NPOs at BACNET_APPLICATION as the test now referenced in each DLL. Modified per Addendum 135-2020cc-1.

Purpose: To verify that the set of network port objects in the IUT are organized in a valid hierarchy.

Test Concept: *Starting with the* ~~Visit each~~ Network Port object *(NP)* 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~~ *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 ) DO {~~
~~        REPEAT NPx = (object id of each Network Port object in NP's hierarchy) DO {~~
~~            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~~
~~        }~~
~~    }~~
1.  REPEAT NPx = (object id of each Network Port object, Reference_Port in NP's hierarchy) DO {
2.      PL = READ (Network Port, NPx), Protocol_Level
3.      IF (PL is BACNET_APPLICATION or NON_BACNET_APPLICATION) THEN
4.          ERROR Invalid Protocol_Level in child Network Port object
5.      IF (PL is PHYSICAL) THEN
6.          VERIFY (Network Port, NPx), Reference_Port = 4194303
    }
7.  IF (Protocol_Revision >= 24 and Additional_Reference_Ports is present) THEN {
8.      IF (NP, Reference_Port property is not present) THEN
9.          ERROR missing Reference_Port property
10.     REPEAT (for each entry Network Port object, Additional_Reference_Ports) DO {
11.         REPEAT NPx = (object id of each Network Port object, Additional_Reference_Ports in NP's hierarchy) DO {
12.             PL = READ (Network Port, NPx), Protocol_Level
13.             IF PL is BACNET_APPLICATION or NON_BACNET_APPLICATION THEN
14                  ERROR Invalid Protocol_Level in child Network Port object
15.             IF PL is PHYSICAL THEN
16.                 VERIFY (Network Port, NPx), Additional_Reference_Ports = (empty list)
        }
    }
```

#### 7.3.2.46.4.2 Properties in Referenced Network Port Reflected in Top Network Port Object

Reason for Change: The test no longer needs to test all NPOs at BACNET_APPLICATION is the test now referenced in each DLL

Purpose: To verify that properties in referenced Network Port objects are reflected in the top Network Port object.

Test Concept: *The* ~~Visit each~~ Network Port object *(NP)* 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) DO {~~
~~         -- verify that the required properties exist for this Network Port object based~~
~~         -- on its Network_Type~~
~~         REPEAT P = (each required property for NP's Network_Type, see Table 12-72) DO {~~
~~              VERIFY (Network Port, NP), P = (any valid value)~~
~~         }~~
~~         REPEAT NPx = (object id of each Network Port object in NP's hierarchy) DO {~~
~~              -- 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) DO {~~
~~                   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~~
~~              }~~
~~         }~~
~~    }~~
-- verify that the required properties exist for this Network Port object based on its Network_Type
1.   REPEAT P = (each required property for NP's Network_Type, see Table 12-72) DO {
2.        VERIFY (Network Port, NP), P = (any valid value)
     }
3.   REPEAT NPx = (object id of each Network Port object in NP's hierarchy) DO {
          -- 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)
4.        REPEAT P = (each expected property in NPx based on its Network_Type and
                    Protocol_Level as defined in Table 12-73) DO {
5.             V1 = READ (Network Port, NPx), P
6.             IF (P is not in a higher Network Port object in this hierarchy) THEN
7.                  VERIFY (Network Port, NP), P = V1
          }
     }
```

### 7.3.2.46.5 APDU_Length Test

Reason for Change: Test had an invalid conditional.

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) DO {
3.          IF NP.APDU_Length < 50 THEN
4.              ERROR "APDU_Length must not be less than 50."
5.          IF NP.APDU_Length > (the maximum allowable for the Network_Type) THEN
6.              ERROR "APDU_Length is too large for the connected Network_Type"
            ~~IF MAX_APDU <> NP.APDU_Length THEN~~
*7.         IF MAX_APDU < NP.APDU_Length THEN*
8.              MAX_APDU = NP.APDU_Length
    }
9.  VERIFY (Device, 4194303), Max_APDU_Length_Supported <= MAX_APDU
```

**7.3.2.47 Timer Object Tests**

**7.3.2.47.1 Positive Tests**

**7.3.2.47.1.7 Already Running Timer Restarted with Default_Timeout**

Reason for Change: Fix the property references.

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~~ *Timeout* different from Default_~~Value~~ *Timeout*. 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~~ *Timeout* different from Default_~~Value~~*Timeout*.

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.X67 Color Object Tests**

**7.3.2.X67.1 Color Object Present_Value Startup Test**

Reason for Change: No test exists for this functionality.

Purpose: This test verifies that the Color object's Present_Value goes to either the last Tracking_Value or a default color on startup, depending on the color in the Default_Color property.

Test Concept: The IUT is restarted and Present_Value is verified to go either to Default_Color or the previous color in effect prior to restart if Default_Color is (0,0). The color output in Tracking_Value is verified to go to either Present_Value or the previous color before the restart.

Configuration Requirements: The IUT is not performing any color commands or fades at the beginning of this test. The starting Present_Value, PV1, shall be set to something other than the Default_Color, DC.

Test Steps:

```
1.  VERIFY In_Progress = IDLE
2.  READ PV1 = Present_Value
3.  READ DC = Default_Color
4.  CHECK (PV1 does not equal DC)
5.  MAKE (the IUT restart)
6.  WAIT (for the IUT to restart)
7.  IF (DC = (0,0)) THEN
8.          VERIFY Present_Value = PV1
9.          VERIFY In_Progress = IDLE
    ELSE
10.         VERIFY Present_Value = DC
11. IF (the IUT's color output is updated on startup) THEN
12.         VERIFY Tracking_Value = (the Present_Value read in the previous step)
13.         VERIFY In_Progress = IDLE
    ELSE
14.         VERIFY In_Progress = NOT_CONTROLLED
```

### 7.3.2.X67.2 Transition NONE Test

Reason for Change: No test exists for this functionality.

Purpose: This test verifies that when Transition is NONE or not supported, writing to the Present_Value is set to the target color immediately.

Test Concept: Transition is verified as NONE or not supported. Tracking_Value is read. A different value is written to Present_Value and Tracking_Value is read back as equal to Present_Value.

Configuration Requirements: The IUT is not performing any color commands or fades at the start of this test.

Test Steps:

```
1.  IF (Transition property is supported) THEN
2.          VERIFY Transition = NONE
3.  READ TV = Tracking_Value
4.  WRITE Present_Value = (C1: any valid color supported by the IUT, other than TV)
5.  VERIFY Tracking_Value = C1
6.  VERIFY In_Progress = IDLE
```

### 7.3.2.X67.3 Color Object Present_Value Out Of Range Test

Reason for Change: No test exists for this functionality.

Purpose: This test verifies that the IUT responds with the correct behavior when a color which is out of range is written to its Present_Value, depending on the color value written.

Test Concept: Present_Value is read, then written to with a value outside the allowed range of (0,0) to (1,1). An error is received with Error Class = PROPERTY, and Error Code = VALUE_OUT_OF_RANGE. Then a value which is within the allowed range, but outside of the range supported by the IUT, is written to Present_Value. Either an error is returned and Present_Value unchanged, or the value is accepted but the Present_Value is changed to the closest color supported. This is repeated if the IUT does not support the entire CIE chromaticity curve using a value within the curve, but outside of the range supported by the IUT.

Configuration Requirements: There are no configuration requirements for this test.

Test Steps:

```
1.  READ PV1 = Present_Value
2.  TRANSMIT WriteProperty-Request,
        'Property Identifier' = Present_Value,
        'Property Value' = (any value outside of the range (0,0) to (1,1))
3.  RECEIVE BACnet-Error-PDU,
        Error Class = PROPERTY
        Error Code = VALUE_OUT_OF_RANGE
4.  VERIFY Present_Value = PV1
5.  TRANSMIT WriteProperty-Request,
        'Property Identifier' = Present_Value,
        'Property Value' = (any value outside of the curved space of the CIE chromaticity diagram and within the range (0,0) to (1,1))
6.  IF (the IUT clamps color values outside the CIE chromaticity curve but within the range (0,0) to (1,1)) THEN
7.      RECEIVE BACnet-SimpleACK-PDU
8.      READ PV1 = Present_Value
9.      IF (Transition is present and set to FADE) THEN
10.         WAIT (Default_Fade_Time milliseconds)
11.     CHECK (that PV1 and Tracking_Value are the closest supported color to what was written)
    ELSE
12.     RECEIVE BACnet-Error-PDU,
            Error Class = PROPERTY
            Error Code = VALUE_OUT_OF_RANGE
13.     VERIFY Present_Value = PV1
14. IF (the IUT does not support all color values within the CIE chromaticity curve) THEN
15.     TRANSMIT WriteProperty-Request,
            'Property Identifier' = Present_Value,
            'Property Value' = (any value unsupported by the IUT and within the curved space of the CIE chromaticity curve)
16.     IF (the IUT clamps unsupported color values) THEN
17.         RECEIVE BACnet-SimpleACK-PDU
18.         READ PV1 = Present_Value
19.         IF (Transition is present and set to FADE) THEN
20.             WAIT (Default_Fade_Time milliseconds)
21.         CHECK (that PV1 and Tracking_Value are the closest supported color to what was written)
        ELSE
22.         RECEIVE BACnet-Error-PDU,
                Error Class = PROPERTY
                Error Code = VALUE_OUT_OF_RANGE
23.         VERIFY Present_Value = PV1
```

### 7.3.2.X67.4 Color Object Color_Command Out Of Range Test

Reason for Change: No test exists for this functionality.

Purpose: This test verifies that the IUT responds with the correct behavior when a target color which is out of range is written to its Color_Command, depending on the color written.

Test Concept: Present_Value is read, then Color_Command is written to with a target value outside the allowed range of (0,0) to (1,1). An error is received with Error Class = PROPERTY, and Error Code = VALUE_OUT_OF_RANGE. Then a Color_Command with a target value which is within the allowed range, but outside of the range supported by the IUT, is written to the IUT. Either an error is returned and Present_Value unchanged, or the value is accepted but the Present_Value is changed to the closest color supported. This is repeated if the IUT does not support the entire CIE chromaticity curve.

Configuration Requirements: The IUT is not performing any color commands or fades.

Test Steps:

```
1.  READ PV1 = Present_Value
2.  TRANSMIT WriteProperty-Request,
        'Property Identifier' = Color_Command,
        'Property Value' = (FADE_TO_COLOR, any value outside of the range (0,0) to (1,1))
3.  RECEIVE BACnet-Error-PDU,
        Error Class = PROPERTY
        Error Code = VALUE_OUT_OF_RANGE
4.  VERIFY Present_Value = PV1
5.  TRANSMIT WriteProperty-Request,
        'Property Identifier' = Color_Command,
        'Property Value' = (FADE_TO_COLOR, any value outside of the curved space of the CIE chromaticity diagram and within the range (0,0) to (1,1), 100)
6.  IF (the IUT clamps color values outside the CIE chromaticity curve but within the range (0,0) to (1,1)) THEN
7.      RECEIVE BACnet-SimpleACK-PDU
8.      WAIT (100 milliseconds)
9.      CHECK (Present_Value and Tracking_Value are equal to each other and within the range of colors supported by the IUT)
    ELSE
10.     RECEIVE BACnet-Error-PDU,
            Error Class = PROPERTY
            Error Code = VALUE_OUT_OF_RANGE
11.     VERIFY Present_Value = PV1
12. IF (the IUT does not support all color values within the CIE chromaticity curve) THEN
13.     READ PV2 = Present_Value
14.     TRANSMIT WriteProperty-Request,
            'Property Identifier' = Color_Command,
            'Property Value' = (FADE_TO_COLOR, any value unsupported by the IUT and within the curved space of the CIE chromaticity diagram, 100)
15.     IF (the IUT clamps unsupported color values) THEN
16.         RECEIVE BACnet-SimpleACK-PDU
17.         WAIT (100 milliseconds)
18.         CHECK (Present_Value and Tracking_Value are equal to each other and within the range of colors supported by the IUT)
        ELSE
19.         RECEIVE BACnet-Error-PDU,
                Error Class = PROPERTY
                Error Code = VALUE_OUT_OF_RANGE
20.         VERIFY Present_Value = PV2
```

### 7.3.2.X67.5 Invalid Color_Command Operations Test

Reason for Change: No test exists for this functionality.

Purpose: This test verifies that the IUT responds with the correct error when invalid color commands are written to the Color_Command property of the Color object.

Test Concept: Present_Value and Color_Command are read, then Color_Command is written to with each unsupported CCT color command. An error is received each time, with Error Class = PROPERTY and Error Code = VALUE_OUT_OF_RANGE. When the error is received, TD verifies the Present_Value has not changed.

Configuration Requirements: There are no configuration requirements for this test.

Test Steps:

```
1   READ CC = Color_Command
2.  READ PV = Present_Value
3.  REPEAT X = (each invalid Color_Command operation, including NONE and a value not defined) DO
    {
4.      TRANSMIT WriteProperty-Request,
            'Property Identifier' = Color_Command,
            'Property Value' = (X)
5.      RECEIVE BACnet-Error-PDU,
            Error Class = PROPERTY
            Error Code = VALUE_OUT_OF_RANGE
6.      VERIFY Present_Value = PV
7.      VERIFY Color_Command = CC
    }
```

### 7.3.2.X67.6 FADE_TO_COLOR Color Command Test

Reason for Change: No test exists for this functionality.

Purpose: This test verifies that the IUT will accept a FADE_TO_COLOR color command when sent with minimum and maximum fade-times and without a specified fade-time.

Test Concept: Color_Command is written to with Operation = FADE_TO_COLOR and a valid target color, and the minimum fade-time allowed by the standard. Then another Color Command is written with the maximum fade-time allowed by the standard. TD verifies the color fade has started. A final color command is written without a fade-time parameter. TD verifies Present_Value and if Default_Fade_Time is large enough, also verifies In_Progress and Tracking_Value during the fade. After Default_Fade_Time has elapsed, TD verifies the fade is completed.

Configuration Requirements: There are no configuration requirements for this test.

Test Steps:

```
1.  WRITE Color_Command = (FADE_TO_COLOR, (C1: any valid color supported by the IUT), 100)
2.  WAIT (100 milliseconds)
3.  VERIFY In_Progress = IDLE
4.  VERIFY Present_Value = C1
5.  WRITE Color_Command = (FADE_TO_COLOR, (C2: any valid color supported by the IUT other
than C1), 86400000)
6.  VERIFY In_Progress = FADE_ACTIVE
7.  VERIFY Present_Value = C2
-- Send a color command without a fade-time, overwriting the previous one, to verify usage of
Default_Fade_Time
8.  READ FT = Default_Fade_Time
9.  WRITE Color_Command = (FADE_TO_COLOR, C1)
```

```
10. IF (FT is large enough for TD to read properties before elapsing) THEN
11.     BEFORE (Default_Fade_Time milliseconds)
12.         VERIFY In_Progress = FADE_ACTIVE
13.         VERIFY Tracking_Value <> C1
14. VERIFY Present_Value = C1
15. WAIT (Default_Fade_Time milliseconds)
16. VERIFY Tracking_Value = C1
17. VERIFY In_Progress = IDLE
```

### 7.3.2.X67.8 Interrupting a Fade In Progress

Reason for Change: No test exists for this functionality.

Purpose: This test verifies that the IUT will stop a fade in progress when Present_Value is written to, when a new color command is written, or when STOP is written to the Color_Command property.

Test Concept: TD writes a color command to begin a fade to color with a specified fade-time, then it interrupts the fade by writing to the Present_Value property. The fade should immediately stop and go to the color written in Present_Value, depending on the presence, and value of, the Transition property. Then TD writes the same color command to begin another fade to color. Before this fade elapses, TD interrupts the fade with a color command to go to a different color. TD then interrupts this final color command by writing STOP to the Color_Command property. TD verifies that the final state of Present_Value, In_Progress, and Color_Command.

Configuration Requirements: The IUT should not have a fade in progress at the beginning of this test. If Transition is configurable, it shall be configured to FADE.

Notes to Tester: This test can be made easier by selecting three distinct colors that the IUT supports.

Test Steps:

```
1.  READ C1 = Present_Value
2.  VERIFY In_Progress = IDLE
3.  WRITE Color_Command = (FADE_TO_COLOR, (C2: any valid color supported by the IUT other than C1), 86400000)
4.  VERIFY In_Progress = FADE_ACTIVE
5.  VERIFY Tracking_Value <> C2
6.  VERIFY Present_Value = C2
-- Interrupt the color command fade by writing to Present_Value
7.  WRITE Present_Value = (C3: any valid color supported by the IUT other than C1 and C2)
8.  IF (Transition property is not present, or set to NONE) THEN
9.      VERIFY Tracking_Value = C3
10.     VERIFY In_Progress = IDLE
    ELSE
11.     VERIFY In_Progress = FADE_ACTIVE
-- Interrupt the fade or start a new one depending on Transition's value
12. WRITE Color_Command = (FADE_TO_COLOR, C1, 86400000)
13. VERIFY Present_Value = C1
14. VERIFY In_Progress = FADE_ACTIVE
15. VERIFY Tracking_Value <> C1
-- Send a different color command, to interrupt the previous one
16. WRITE Color_Command = (FADE_TO_COLOR, C2, 86400000)
-- Interrupt the fade with the STOP command
17. READ TV = Tracking_Value
18. WRITE Color_Command = STOP
19. VERIFY Present_Value = (a value approximately equal to, or equal to, TV)
```

20. VERIFY In_Progress = IDLE
21. VERIFY Color_Command = STOP

### 7.3.2.X67.11 Color_Command Fade-time Out Of Range Test

Reason for Change: No test exists for this functionality.

Purpose: This test verifies that the IUT responds with the correct behavior when a fade-time which is out of range is written to its Color_Command.

Test Concept: Present_Value is read, then a color command is written with a fade-time smaller than the minimum allowed. An error is received with Error Class = PROPERTY, and Error Code = VALUE_OUT_OF_RANGE. Then another color command is written with a target value which is larger than the maximum allowed. An error is received with Error Class = PROPERTY, and Error Code = VALUE_OUT_OF_RANGE. After each write, TD verifies the Present_Value is unchanged.

Configuration Requirements: The IUT is not performing any color commands or fades.

Test Steps:

```
1.  READ PV1 = Present_Value
2.  TRANSMIT WriteProperty-Request,
        'Property Identifier' = Color_Command,
        'Property Value' = (FADE_TO_COLOR, C1: any valid color, 99)
3.  RECEIVE BACnet-Error-PDU,
        Error Class = PROPERTY
        Error Code = VALUE_OUT_OF_RANGE
4.  VERIFY Present_Value = PV1
5.  TRANSMIT WriteProperty-Request,
        'Property Identifier' = Color_Command,
        'Property Value' = (FADE_TO_COLOR, , C1, 86400001)
6.  RECEIVE BACnet-Error-PDU,
        Error Class = PROPERTY
        Error Code = VALUE_OUT_OF_RANGE
7.  VERIFY Present_Value = PV1
```

### 7.3.2.X67.22 Color Commands Ignore Transition When Fade-Time is Specified

Reason for Change: No test exists for this functionality.

Purpose: This test verifies that when the IUT supports the Transition property, writes to Color_Command with a specified fade-time will use that fade time instead of the fade time that is specified by Transition.

Test Concept: TD writes a Color_Command with a fade-time, that is different from Default_Fade_Time and verifies the color fade did not end after Default_Fade_Time milliseconds.

Configuration Requirements: Default_Fade_Time must not be set to 86400000.

Test Steps:

```
1.  VERIFY Transition = FADE | NONE
2.  READ C1 = Present_Value
3.  WRITE Color_Command = (FADE_TO_COLOR, (C2: any valid color supported by the IUT other than C1), 86400000)
4.  WAIT (Default_Fade_Time milliseconds)
5.  VERIFY In_Progress = FADE_ACTIVE
```

```
6.  VERIFY Tracking_Value <> C1
7.  VERIFY Tracking_Value <> C2
8.  WRITE Color_Command = STOP
```

### 7.3.2.X67.31 Configuring Default_Fade_Time Within Allowable Range

Reason for Change: No test exists for this functionality.

Purpose: This test verifies that the IUT supports a configurable Default_Fade_Time.

Test Concept: The IUT is configured with a different Default_Fade_Time, FT1. If Transition is supported and set to FADE, TD writes to Present_Value and the Tracking_Value is verified to only be equal to the written color after FT1 milliseconds have passed. Otherwise, a color command with fade-time = FT2 is written and Tracking_Value is verified to only be equal to the written color after FT2 milliseconds have passed.

Configuration Requirements: There are no configuration requirements for this test.
Notes to Tester: Sufficiently large fade times should be used when selecting FT1 or FT2, in order to allow TD to read the Tracking_Value after FT0 but before FT1 or FT2 has passed.

Test Steps:

```
1.  READ FT0 = Default_Fade_Time
2.  MAKE (configure the IUT such that Default_Fade_Time = FT1: a different fade time longer than FT0)
3.  VERIFY FT1 = Default_Fade_Time
4.  READ C1 = Present_Value
-- Write to Present_Value to verify Default_Fade_Time gets applied
5.  IF (Transition is present and equal to FADE) THEN
6.      WRITE Present_Value = (C2, a different color than C1)
7.      VERIFY In_Progress = FADE_ACTIVE
8.      WAIT (FT0 milliseconds)
9.      VERIFY Tracking_Value <> C2
10.     WAIT (FT1 - FT0 milliseconds)
11.     VERIFY Tracking_Value = C2
12.     VERIFY In_Progress = IDLE
    ELSE
13.     WRITE Color_Command = (FADE_TO_COLOR, (C2: any valid color supported by the IUT
other than C1), (FT2: a different fade time longer than FT1))
14.     VERIFY In_Progress = FADE_ACTIVE
15.     WAIT (FT1 milliseconds)
16.     VERIFY Tracking_Value <> C2
17.     WAIT (FT2 - FT1 milliseconds)
18.     VERIFY Tracking_Value = C2
19.     VERIFY In_Progress = IDLE
```

### 7.3.2.X67.32 Writing Default_Fade_Time Positive Test

Reason for Change: No test exists for this functionality.

Purpose: This test verifies that the IUT can be configured with a Default_Fade_Time at the bounds of the allowable value range using BACnet services.

Test Concept: Default_Fade_Time is written with a value equal to 100 milliseconds. Then Default_Fade_Time is written with a value equal to 86400000 milliseconds.

Configuration Requirements: There are no configuration requirements for this test.

Test Steps:

```
1.  WRITE Default_Fade_Time = 100
2.  VERIFY Default_Fade_Time = 100
3.  WRITE Default_Fade_Time = 86400000
4.  VERIFY Default_Fade_Time = 86400000
```

### 7.3.2.X68 Color Temperature Object Tests

### 7.3.2.X68.1 Color Temperature Object Present_Value Startup Test

Reason for Change: No test exists for this functionality.

Purpose: This test verifies that the Color Temperature object’s Present_Value goes to either the last Present_Value or a default color temperature on startup, depending on the color in the Default_Color_Temperature property.

Test Concept: The IUT is restarted and Present_Value is verified to go either to Default_Color_Temperature or the previous color temperature in effect prior to restart if Default_Color_Temperature is 0. The color temperature output in Tracking_Value is verified to go to either Present_Value or the previous color temperature before the restart.

Configuration Requirements: The IUT is not performing any color temperature commands or fades at the beginning of this test. The starting Present_Value, PV1, shall be set to something other than the Default_Color_Temperature,.

Test Steps:

```
1.  VERIFY In_Progress = IDLE
2.  READ PV1 = Present_Value
3.  READ DCT = Default_Color_Temperature
4.  CHECK (PV1 <> DCT)
5.  MAKE (the IUT restart)
6.  WAIT (for the IUT to restart)
7.  IF (DCT = 0) THEN {
8.      IF (Present_Value is preserved over a power cycle) THEN {
9.          VERIFY Present_Value = PV1
10.         VERIFY In_Progress = IDLE
        }
        ELSE {
11.         VERIFY In_Progress = NOT_CONTROLLED
        }
    }
    ELSE {
12.     VERIFY Present_Value = DCT
13.     VERIFY Tracking_Value = DCT
14.     VERIFY In_Progress = IDLE
    }
```

### 7.3.2.X68.2 Transition NONE Test

Reason for Change: No test exists for this functionality.

Purpose: This test verifies that when Transition is NONE or not supported, writing to the Present_Value is set to the target color temperature immediately.

Test Concept: Transition is verified as NONE or not supported. Tracking_Value is read. A different value is written to Present_Value and Tracking_Value is read back as equal to Present_Value.

Configuration Requirements: The IUT is not performing any color temperature commands or fades at the start of this test.

Test Steps:

```
1.  IF (Transition property is supported) THEN
2.      VERIFY Transition = NONE
3.  READ TV = Tracking_Value
4.  WRITE Present_Value = (C1: any valid color temperature supported by the IUT, other than TV)
5.  VERIFY Tracking_Value = C1
6.  VERIFY In_Progress = IDLE
```

**7.3.2.X68.3 Color Temperature Object Present_Value Clamping Test**

Reason for Change: No test exists for this functionality.

Purpose: This test verifies that the IUT responds with the correct behavior when a color temperature which is within the standard range for Present_Value but out of range of Min_Pres_Value and Max_Pres_Value is written.

Test Concept: A Color Temperature object's (O1) Present_Value is read, then written to using values T1 and T2, where T1 is a value between 1000 Kelvin and Min_Pres_Value and T2 is a value between Max_Pres_Value and 30000 Kelvin.
Configuration Requirements: The color temperature object should not be executing any fades.

Notes to Tester: Configuring Transition to NONE, or minimizing Default_Fade_Time and maximizing Default_Ramp_Rate will assist in reducing the time it takes to execute this test.

Test Steps:

```
1.  READ PV1 = Present_Value
2.  TRANSMIT WriteProperty-Request,
        'Object Identifier' = O1,
        'Property Identifier' = Present_Value,
        'Property Value' = 999
3.  RECEIVE BACnet-Error-PDU,
        'Error Class' = PROPERTY,
        'Error Code' = VALUE_OUT_OF_RANGE
4.  VERIFY Present_Value = PV1
5.  TRANSMIT WriteProperty-Request,
        'Object Identifier' = O1,
        'Property Identifier' = Present_Value,
        'Property Value' = 30001
6.  RECEIVE BACnet-Error-PDU,
        'Error Class' = PROPERTY,
        'Error Code' = VALUE_OUT_OF_RANGE
7.  VERIFY Present_Value = PV1
8.  IF (the IUT supports Min_Pres_Value and Max_Pres_Value) THEN {
9.      IF (Min_Pres_Value > 1000) THEN {
10.         WRITE Present_Value = T1
11.         VERIFY Present_Value = Min_Pres_Value
12.         IF (Transition is present and set to FADE) THEN {
```

```
13.                 WAIT (Default_Fade_Time milliseconds)
                }
14.             IF (Transition is present and set to RAMP) THEN {
15.                 WAIT ( ((PV1 - Min_Pres_Value) / Default_Ramp_Rate) seconds)
                }
16.             VERIFY Tracking_Value = Min_Pres_Value
            }
17.         IF (Max_Pres_Value < 30000) THEN {
18.             READ PV1 = Present_Value
19.             WRITE Present_Value = T2
20.             VERIFY Present_Value = Max_Pres_Value
21.             IF (Transition is present and set to FADE) THEN {
22.                 WAIT (Default_Fade_Time milliseconds)
                }
23.             IF (Transition is present and set to RAMP) THEN {
24.                 WAIT ( ((Max_Pres_Value - PV1) / Default_Ramp_Rate) seconds)
                }
25.             VERIFY Tracking_Value = Max_Pres_Value
            }
        }
```

### 7.3.2.X68.4 Color Temperature Object Color_Command Out Of Range Test

Reason for Change: No test exists for this functionality.

Purpose: This test verifies that the IUT responds with the correct behavior when a target color temperature which is out of range is written to its Color_Command, depending on the color temperature written.

Test Concept: A Color Temperature object's (O1) Present_Value is read, then Color_Command is written to with a target value outside the allowed range of 1000 to 30000. An error is received with Error Class = PROPERTY, and Error Code = VALUE_OUT_OF_RANGE. Then, if the IUT supports Min_Pres_Value and Max_Pres_Value, a Color_Command with a target value which is within the allowed range but outside of the range supported by the IUT, is written to the IUT. An error is returned and Present_Value is clamped to the Min_Pres_Value or Max_Pres_Value.

Configuration Requirements: The IUT is not performing any color commands or fades.

Test Steps:

```
1.  REPEAT X = (each valid Color_Command operation) DO {
2.      READ PV1 = Present_Value
3.      TRANSMIT WriteProperty-Request,
            'Object Identifier' = O1,
            'Property Identifier' = Color_Command,
            'Property Value' = (X, additional parameters which would result in a color temperature below
1000K)
4.      RECEIVE BACnet-Error-PDU,
            'Error Class' = PROPERTY,
            'Error Code' = VALUE_OUT_OF_RANGE
5.      READ PV1 = Present_Value
6.      TRANSMIT WriteProperty-Request,
            'Object Identifier' = O1,
            'Property Identifier' = Color_Command,
            'Property Value' = (X, additional parameters which would result in a color temperature above
30000K)
7.      RECEIVE BACnet-Error-PDU,
            'Error Class' = PROPERTY,
            'Error Code' = VALUE_OUT_OF_RANGE
8.      VERIFY Present_Value = PV1
9.      IF (the IUT supports Min_Pres_Value and Max_Pres_Value) THEN {
10.         IF (Min_Pres_Value > 1000) THEN {
11.             WRITE Color_Command = (X, additional parameters which would result in a color temperature between 1000 and Min_Pres_Value)
12.             VERIFY Present_Value = Min_Pres_Value
13.             WHILE (In_Progress <> IDLE) { }  -- Do nothing
14.             VERIFY Tracking_Value = Min_Pres_Value
            }
15.         IF (Max_Pres_Value < 30000) THEN {
16.             WRITE Color_Command = (X, additional parameters which would result in a color temperature between Max_Pres_Value and 30000)
17.             VERIFY Present_Value = Max_Pres_Value
18.             WHILE (In_Progress <> IDLE) { -- Do nothing }
19.             VERIFY Tracking_Value = Max_Pres_Value
            }
        }
    }
```

### 7.3.2.X68.5 Invalid Color_Command Operations Test

Reason for Change: No test exists for this functionality.

Purpose: This test verifies that the IUT responds with the correct error when invalid color commands are written to the Color_Command property of the Color Temperature object.

Test Concept: A Color Temperature object's (O1) Present_Value and Color_Command are read, then Color_Command is written to with each unsupported CCT color command. An error is received each time, with Error Class = PROPERTY and Error Code = VALUE_OUT_OF_RANGE. When the error is received, TD verifies the Present_Value has not changed.

Configuration Requirements: There are no configuration requirements for this test.

Test Steps:

```
1   READ CC = Color_Command
2.  READ PV = Present_Value
3.  REPEAT X = (each invalid Color_Command operation, including NONE and a value not defined) DO {
4.      TRANSMIT WriteProperty-Request,
            'Object Identifier' = O1,
            'Property Identifier' = Color_Command,
            'Property Value' = (X)
5.      RECEIVE BACnet-Error-PDU,
            'Error Class' = PROPERTY,
            'Error Code' = VALUE_OUT_OF_RANGE
6.      VERIFY Present_Value = PV
7.      VERIFY Color_Command = CC
    }
```

### 7.3.2.X68.6 Valid Color Command Test

Reason for Change: No test exists for this functionality.

Purpose: This test verifies that the IUT will accept all valid color commands when sent with minimum and maximum parameters and without optional parameters present.

Test Concept: Each valid Color Command Operation is written with valid target color temperatures, exercising the optional fields of each operation. TD also verifies that when writing a Color Command without the optional field, that the default parameter is used. TD verifies the fade is completed once enough time has elapsed. This process is repeated for all remaining valid Color Commands.

Configuration Requirements: There are no configuration requirements for this test.

Test Steps:

```
1.  REPEAT X = (each valid Color_Command operation) DO {
2.      WRITE Color_Command = (X, (C1: any valid target color temperature supported by the IUT, or absent if X does not support it), MIN: the minimum allowable value for this parameter for X)
3.      WAIT (until the color command has finished)
4.      VERIFY In_Progress = IDLE
5.      VERIFY Present_Value = (the color output determined by X and C1's presence)
6.      WRITE Color_Command = (X, (C2: any valid target color temperature supported by the IUT other than C1, or absent if X does not support it), MAX: the maximum allowable value for this parameter for X)
7.      VERIFY In_Progress = (an appropriate state for X)
8.      VERIFY Present_Value = (the color output determined by X and C2's presence)
-- Write Color Command without the optional parameter, interrupting the last one to verify use of the default property corresponding to the optional parameter in the Color Command
9.      READ T1 = (the 'default' value corresponding to the optional parameter in X)
10.     WRITE Color_Command = (X, (C1, or absent if X does not support it))
11.     VERIFY Present_Value = (the color output determined by X and C1's presence)
12.     IF (the color command will finish within a reasonable timeframe based on X and T1) THEN {
13.         WAIT (for the fade to finish based on T1)
14.         VERIFY Tracking_Value = (the color output determined by X and C1's presence)
15.         VERIFY In_Progress = IDLE
        }
16.     ELSE {
17.         WRITE Color_Command = STOP
        }
    }
```

### 7.3.2.X68.8 Interrupting a Fade In Progress

Reason for Change: No test exists for this functionality.

Purpose: This test verifies that the IUT will stop a fade in progress when Present_Value is written to, when a new color command is written, or when STOP is written to the Color_Command property.

Test Concept: TD writes a color command to a Color Temperature object's (O1) which starts a fade to a color temperature (C2) with a specified fade-time, then it interrupts the fade by writing to the Present_Value property. The fade should immediately stop and go to the color temperature written in Present_Value, depending on the presence, and value of, the Transition property. Then for each valid color command, TD writes a color command to begin a fade to color temperature. Before the operation completes, TD interrupts the fade with a different color command. TD verifies that Color_Command matches the command that was written and that Present_Value and In_Progress have appropriate values.

Configuration Requirements: The IUT should not have a fade or ramp in progress at the beginning of this test. If Transition is configurable, it shall not be configured to NONE at the start of this test.

Test Steps:

```
1.  READ C1 = Present_Value
2.  VERIFY In_Progress = IDLE
3.  WRITE Color_Command = (FADE_TO_CCT, (C2: any valid color temperature supported by the IUT other than C1), 86400000)
4.  VERIFY In_Progress = FADE_ACTIVE
5.  VERIFY Present_Value = C2
-- Interrupt the color command by writing to Present_Value
6.  WRITE Present_Value = C1
7.  REPEAT X = (each valid Color_Command operation including FADE_TO_CCT and STOP) DO {
8.        WRITE Color_Command = (FADE_TO_CCT, C2, 86400000)
9.        WRITE Color_Command = (X, (C1, or absent if X does not support a target color temperature), (the maximum value for this parameter, or absent if X does not support a fade-time or ramp-rate))
10.       VERIFY Present_Value =  (a value appropriate to X)
11.       VERIFY In_Progress = (a value appropriate to X)
12.       VERIFY Color_Command = (X)
    }
```

### 7.3.2.X68.9 Interrupting a Ramp In Progress

Reason for Change: No test exists for this functionality.

Purpose: This test verifies that the IUT will stop a ramp in progress when Present_Value is written to, when a new color command is written, or when STOP is written to the Color_Command property.

Test Concept: TD writes a color command to a Color Temperature object's (O1) which starts a ramp to a color temperature C2 with a specified ramp-time, then it interrupts the ramp by writing to the Present_Value property. The ramp should immediately stop and go to the color temperature written in Present_Value, depending on the presence, and value of, the Transition property. Then for each valid color command, TD writes a color command operation to begin a ramp to color temperature. Before the operation completes, TD interrupts the ramp with a different color command. TD verifies that Color_Command matches the command that was written and that Present_Value and In_Progress have appropriate values.

Configuration Requirements: The IUT should not have a ramp in progress at the beginning of this test. If Transition is configurable, it shall not be configured to NONE at the start of this test.

Test Steps:

```
1.  READ C1 = Present_Value
2.  VERIFY In_Progress = IDLE
3.  WRITE Color_Command = (RAMP_TO_CCT, (C2: any valid color temperature supported by the IUT other than C1), 1)
4.  VERIFY In_Progress = RAMP_ACTIVE
5.  VERIFY Present_Value = C2
-- Interrupt the color command by writing to Present_Value
6.  WRITE Present_Value = C1
7.  REPEAT X = (each valid Color_Command operation including RAMP_TO_CCT and STOP) DO {
8.        WRITE Color_Command = (RAMP_TO_CCT, C2, 1)
9.        WRITE Color_Command = (X, (C2, or absent if X does not support a target color temperature), (the maximum value for this parameter, or absent if X does not support a fade-time or ramp-rate)
10.       VERIFY Present_Value = (a value appropriate to X)
11.       VERIFY In_Progress = (a value appropriate to X)
12.       VERIFY Color_Command = (X)
    }
```

### 7.3.2.X68.11 Color_Command Optional Parameter Out Of Range Test

Reason for Change: No test exists for this functionality.

Purpose: This test verifies that the IUT responds with the correct behavior when Color Command is written which contains an optional parameter whose value is outside the allowed range.

Test Concept: Present_Value of a Color Temperature object (O1) is read, then for each valid color command operation, a color command is written with the optional parameter smaller than the minimum allowed. An error is received with Error Class = PROPERTY, and Error Code = VALUE_OUT_OF_RANGE. Then another color command is written with the optional parameter larger than the maximum allowed. An error is received with Error Class = PROPERTY, and Error Code = VALUE_OUT_OF_RANGE. After each write, TD verifies the Present_Value is unchanged.

Configuration Requirements: The IUT is not performing any color commands. If Transition is present and configurable, it shall be configured to NONE. If Min_Pres_Value and Max_Pres_Value are configurable, they shall not be configured to 1000 and 30000 respectively.

Test Steps:

```
1.  READ Cx = Present_Value
2.  REPEAT X = (each valid Color Command operation other than STOP) DO {
3.      TRANSMIT WriteProperty-Request,
            'Object Identifier' = O1,
            'Property Identifier' = Color_Command,
            'Property Value' = (X, (C1: any valid target color temperature other than Cx or absent), (a value smaller than the minimum allowed for this parameter))
4.      RECEIVE BACnet-Error-PDU,
            'Error Class' = PROPERTY,
            'Error Code' = VALUE_OUT_OF_RANGE
5.      VERIFY Present_Value = Cx
6.      TRANSMIT WriteProperty-Request,
            'Object Identifier' = O1,
            'Property Identifier' = Color_Command,
            'Property Value' = (X, (C1 or absent), (a value larger than the maximum allowed for this parameter))
7.      RECEIVE BACnet-Error-PDU,
            'Error Class' = PROPERTY,
            'Error Code' = VALUE_OUT_OF_RANGE
8.      VERIFY Present_Value = Cx
    }
9.  IF (the IUT supports the Min_Pres_Value and Max_Pres_Value properties) THEN {
10.     IF (Min_Pres_Value > 1000) THEN {
11.         WRITE Color_Command = (STEP_DOWN_CCT, (S1: a value such that 1000 <= Cx - S1 < Min_Pres_Value))
12.         VERIFY Present_Value = Min_Pres_Value
        }
13.     IF (Max_Pres_Value < 30000) THEN {
14.         WRITE Color_Command = (STEP_UP_CCT, (S2: a value such that Max_Pres_Value < Present_Value + S2 <=  30000))
15.         VERIFY Present_Value = Max_Pres_Value
        }
    }
```

### 7.3.2.X68.22 Default_Fade_Time Test

Reason for Change: No test exists for this functionality.

Purpose: This test verifies that writes to Color_Command with a specified fade-time will use that fade-time instead of the Default_Fade_Time and when fade-time is not specified, Default_Fade_Time is used

Test Concept: TD writes a Color_Command with a fade-time, that is different from Default_Fade_Time and verifies the color temperature fade did not end after Default_Fade_Time milliseconds. A second color command is written without the fade-time parameter and TD verifies that the fade ends after Default_Fade_Time milliseconds, if Default_Fade_Time is a reasonable value.

Configuration Requirements: There are no configuration requirements for this test.

Test Steps:

```
1.  READ C1 = Present_Value
2.  WRITE Color_Command = (FADE_TO_CCT, (C2: any valid color temperature supported by the IUT other than C1), (Ft: a valid fade-time that is different than Default_Fade_Time and long enough for the next step to be executed))
3.  IF (Ft > Default_Fade_Time) THEN {
4.          WAIT (Default_Fade_Time milliseconds)
5.          VERIFY In_Progress = FADE_ACTIVE
6.          WAIT (Ft milliseconds - Default_Fade_Time)
7.          VERIFY In_Progress = IDLE
    }
    ELSE { -- (Ft < Default_Fade_Time)
8.          WAIT (Ft milliseconds)
9.          VERIFY In_Progress = IDLE
    }
10. WRITE Color_Command = (FADE_TO_CCT, C1)
11. IF (Default_Fade_Time is not excessively long) THEN {
12.         WAIT (Default_Fade_Time milliseconds)
13.         VERIFY Tracking_Value = C2
14.         VERIFY In_Progress = IDLE
    }
```

### 7.3.2.X68.23 Default_Ramp_Rate Test

Reason for Change: No test exists for this functionality.

Purpose: This test verifies that writes to Color_Command with a specified ramp-rate will use that ramp rate instead of the Default_Ramp_Rate and when ramp-rate is not specified, Default_Ramp_Rate is used.
Test Concept: TD writes a Color_Command with a ramp-rate, that is different from Default_Ramp_Rate and verifies the color temperature ramp did not end after Default_Ramp_Rate. A second color command is written without the ramp-rate parameter and TD verifies that the ramp ends at the appropriate time.

Configuration Requirements: Default_Ramp_Time must not be set to 1. Rr shall be a valid ramp-rate that is different than Default_Ramp_Rate and large enough to allow the test to be executed. T1 shall be the ((absolute value of C1-C2/Default_Ramp_Rate). T2 shall be the ((absolute value of C1-C2/Rr).

Test Steps:

```
1.  READ C1 = Present_Value
2.  WRITE Color_Command = (RAMP_TO_CCT, (C2: any valid color temperature supported by the IUT other than C1), Rr)
3.  IF (Rr > Default_Ramp_Rate) THEN {
4.          WAIT (T1 seconds)
5.          VERIFY In_Progress = RAMP_ACTIVE
6.          WAIT (T1 - T2 seconds)
7.          VERIFY In_Progress = IDLE
      }
      ELSE { -- (Rr < Default_Ramp_Rate)
8.          WAIT (T2 seconds)
9.          VERIFY In_Progress = IDLE\
      }
10.   WRITE Color_Command = (RAMP_TO_CCT, C1)
11.   IF (T1 is not excessively long) THEN {
12          WAIT (T1)
13          VERIFY Tracking_Value = C1
14          VERIFY In_Progress = IDLE
      }
```

### 7.3.2.X68.24 Default_Step_Size Test

Reason for Change: No test exists for this functionality.

Purpose: This test verifies that writes to Color_Command with a specified step-size will use that step size instead of the Default_Step_Size.

Test Concept: TD writes a Color_Command with a step-size, that is different from Default_Step_Size and verifies the color temperature did not change by Default_Step_Size.

Configuration Requirements: Present_Value must not be at the maximum allowable value for the Color Temperature object at the start of the test.

Test Steps:

1. READ C1 = Present_Value
2 WRITE Color_Command = (STEP_UP_CCT, (S1: any valid step size supported by the IUT other than Default_Step_Size, and will not cause clamping of Present_Value))
3. VERIFY In_Progress = IDLE
4. VERIFY Tracking_Value = C1 + S1
5. VERIFY Present_Value = C1 + S1
6 WRITE Color_Command = (STEP_DOWN_CCT, S1)
7. VERIFY In_Progress = IDLE
8. VERIFY Tracking_Value = C1
9. VERIFY Present_Value = C1

### 7.3.2.X68.25 Transition FADE Test

Reason for Change: No test exists for this functionality.

Purpose: This test verifies that when the IUT supports the Transition property set to FADE, writes to Present_Value will use the Default_Fade_Time.

Test Concept: TD writes a Present_Value, verifies the color temperature fade is not continuing after Default_Fade_Time milliseconds.

Configuration Requirements: Transition shall be configured to FADE at the beginning of this test.

Test Steps:

1. VERIFY Transition = FADE
2. READ C1 = Present_Value

```
3.  WRITE Present_Value = (C2: any valid color temperature supported by the IUT other than C1)
4.  IF (TD can read In_Progress before Default_Fade_Time elapses) THEN {
5.      VERIFY In_Progress = FADE_ACTIVE
    }
6.  WAIT (Default_Fade_Time milliseconds)
7.  VERIFY In_Progress = IDLE
8.  VERIFY Tracking_Value = C2
```

### 7.3.2.X68.26 Transition RAMP Test

Reason for Change: No test exists for this functionality.

Purpose: This test verifies that when the IUT supports the Transition property set to RAMP, writes to Present_Value will use the Default_Ramp_Rate.

Test Concept: TD writes a Present_Value, verifies the color temperature ramp is not continuing after Default_Ramp_Rate milliseconds.

Configuration Requirements: Transition shall be configured to RAMP at the beginning of this test.

Test Steps:

```
1.  VERIFY Transition = RAMP
2.  READ C1 = Present_Value
3.  WRITE Present_Value = (C2: any valid color temperature supported by the IUT other than C1)
4.  IF(TD can read In_Progress before the transition completes) THEN {
5.      VERIFY In_Progress = RAMP_ACTIVE
    }
6.  WAIT ((absolute value of C1-C2)/Rr seconds)
7.  VERIFY In_Progress = IDLE
8.  VERIFY Tracking_Value = C2
```

### 7.3.2.X68.35 Configuring Default_Step_Increment Within Allowable Range

Reason for Change: No test exists for this functionality.

Purpose: This test verifies that the IUT supports a configurable Default_Step_Increment.

Test Concept: The IUT is configured with a different Default_Step_Increment, S1. A color command with step-increment S2 is written and Present_Value is verified to only be equal to the written color temperature after that ramp is completed.

Configuration Requirements: Present_Value should not be set to the minimum or maximum color temperature supported by the IUT at the start of the test.

Test Steps:

```
1.  READ S0 = Default_Step_Increment
2.  MAKE (configure the IUT such that Default_Step_Increment = S1: a different step increment than S0)
3.  VERIFY S1 = Default_Step_Increment
4.  READ C1 = Present_Value
5.  WRITE Color_Command = STEP_UP_CCT
6.  VERIFY Present_Value = (C1 + S1)
7.  WRITE Color_Command = STEP_DOWN_CCT
8.  VERIFY Present_Value = C1
```

### 135.1-2023*u*-3 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 135.1-2023. In addition, the test coverage is increased with the addition of new tests.

## 8. APPLICATION SERVICE INITIATION TESTS

### 8.2 ConfirmedCOVNotification Service Initiation Tests

### 8.2.11 ConfirmedCOVNotification Pulse Converter changing Status_Flags

Reason for Change: Removed requirement for configurable Out_Of_Service.

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~~*made to change* 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.~~ *which the Status_Flags property cannot be made to change, 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 is 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**
7.      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)
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
~~10. IF (Out_Of_Service is writable) THEN~~
~~        WRITE Out_Of_Service = FALSE~~
```

### 8.2.12 Change of Value Notification from an Access Door object Present_Value, Status_Flags and Door_Alarm_State property

Reason for Change: Removed requirement for configurable Out_Of_Service.

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*, Door_Alarm_State, and Status_Flags* of the monitored object ~~is~~*are* changed, and a notification shall be received. The Present_Value *and Door_Alarm_State* 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, 11~~ *11,12,13* shall be skipped. For implementations where it is not possible ~~to write Out_Of_Service or~~ to 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 > 0chosen by the TD),
        'Monitored Object Identifier' = X,
        'Issue Confirmed Notifications' =    TRUE,
        'Lifetime' =                 L
2.  RECEIVE BACnet-SimpleACK-PDU
3.  BEFORE **Notification Fail Time**
4.      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)
5.  TRANSMIT BACnet-SimpleACK-PDU
~~5.  IF (Out_Of_Service is writable) THEN~~
```

```
        ~~WRITE Out_Of_Service = TRUE~~
    ~~ELSE~~

6.  MAKE (Status_Flags = any value that differs from initial Status_Flags)
7.  BEFORE **Notification Fail Time**
8.      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)
9.  TRANSMIT BACnet-SimpleACK-PDU
10. IF (Present_Value is writable) THEN
11.     WRITE X,Present_Value = (any value that differs from ReportedPV)
    ELSE
12.     MAKE (Present_Value = any value that differs from ReportedPV)
13. BEFORE **Notification Fail Time**
14.     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)
15. TRANSMIT BACnet-SimpleACK-PDU
16. IF (Door_Alarm_State is now writable) THEN
17.     WRITE Door_Alarm_State = (any value that differs from its initial Door_Alarm_State)
    ELSE
18.     MAKE (Door_Alarm_State = any value that differs from its initial Door_Alarm_State)
19. BEFORE **Notification Fail Time**
20.     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)
21. TRANSMIT BACnet-SimpleACK-PDU

22. TRANSMIT SubscribeCOV-Request,
        'Subscriber Process Identifier' =   (the same value used in the SubscribeCOV-Request),
        'Monitored Object Identifier' = X
23. RECEIVE BACnet-SimpleACK-PDU
~~16. IF (Out_Of_Service is writable) THEN~~

        ~~WRITE Out_Of_Service = FALSE~~

}
```

**8.2.13 Change of Value Notification from an Access Point object**

Reason for Change: Removed requirement for configurable Out_Of_Service..

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~~*are* changed, and a notification shall be received. The ~~properties~~*Access_Event_Time property* 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: None~~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**
4.      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)
5.  TRANSMIT BACnet-SimpleACK-PDU
~~5.  IF (Out_Of_Service is writable) THEN~~
~~        WRITE Out_Of_Service = TRUE~~
~~    ELSE~~
6.  MAKE (Status_Flags = any value that differs from initial Status_Flags)
7.  BEFORE **Notification Fail Time**
8       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 an
                                       Access_Event_Authentication_Factor property)
9.  TRANSMIT BACnet-SimpleACK-PDU
10. IF (Access_Event_Time is now writable) THEN
11.     WRITE Access_Event_Time = (any value that differs from initial Access_Event_Time)
    ELSE
12.     MAKE (Access_Event_Time = any value that differs from initial Access_Event_Time)
13.. BEFORE **Notification Fail Time**
14.     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)
15. TRANSMIT BACnet-SimpleACK-PDU
16. TRANSMIT SubscribeCOV-Request,
'Subscriber Process Identifier' = PI,
'Monitored Object Identifier' = X
17. RECEIVE BACnet-SimpleACK-PDU
~~13. IF (Out_Of_Service is writable) THEN~~
~~WRITE Out_Of_Service = FALSE~~
18. CHECK (verify that no notification message has been transmitted)

**8.2.14 Change of Value Notification from a Credential Data Input Object**

Reason for Change: Removed requirement for configurable Out_Of_Service.

Purpose: To verify that the IUT can initiate ConfirmedCOVNotification service requests conveying a change of the Status_Flags and Update_Time properties of Credential Data Input 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 and Update_Time properties of the monitored object ~~is~~*are* changed, and a notification shall be received. The ~~properties~~ *Present_Value property* 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: *None.*~~At the beginning of the test, the Out_Of_Service property shall have a value of FALSE.~~

Test Steps:

*1.* ~~13.~~ TRANSMIT SubscribeCOV-Request,
'Subscriber Process Identifier' = (PI: any value > 0chosen by the TD),
'Monitored Object Identifier' = X,
'Issue Confirmed Notifications' = TRUE,
'Lifetime' = L
2. RECEIVE BACnet-SimpleACK-PDU
3. BEFORE **Notification Fail Time**
4. 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 Present_Value, initial Status_Flags, and
Update_Time (most recent update time when the Present_Value
was updated))
5. TRANSMIT BACnet-SimpleACK-PDU
~~5. IF (Out_Of_Service is writable) THEN~~
~~WRITE X, Out_Of_Service = TRUE~~
~~ELSE~~
6. MAKE (Status_Flags = any value that differs from initial Status_Flags)

```
7.  BEFORE **Notification Fail Time**
8.      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 Present_Value, new Status_Flags, and Update_Time
                                        (most recent update time when the Present_Value was updated))
9.  TRANSMIT BACnet-SimpleACK-PDU
10. IF (Present_Value is now writable) THEN
        WRITE X, Present_Value = (any value that differs from initial Present_Value)
    ELSE
        MAKE (Present_Value = any value that differs from initial Present_Value)
11. BEFORE **Notification Fail Time**
12.     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))

13. TRANSMIT BACnet-SimpleACK-PDU
14. Verify Update_Time received in step 7.
15. TRANSMIT SubscribeCOV-Request,
        'Subscriber Process Identifier' = PI,
        'Monitored Object Identifier' = X
16. RECEIVE BACnet-SimpleACK-PDU
~~14. IF (Out_Of_Service is writable) THEN~~
~~        WRITE Out_Of_Service = FALSE~~
17. CHECK (verify that no notification message has been transmitted)
```

### 8.2.X1 Change of Value Notification from a Load Control Object (ConfirmedCOVNotification)

Reason for Change: No test for this functionality.

Purpose: To verify that the IUT can initiate ConfirmedCOVNotification service requests conveying a change of value for a Load Control object (O1) when the properties specified in the standard changes.

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 value of each property is changed and it is verified that a COV notification is received.

Configuration Requirements: None.

Test Steps:

```
1.  TRANSMIT SubscribeCOV-Request,
        'Subscriber Process Identifier' =       (P1, 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**
4.      RECEIVE ConfirmedCOVNotification-Request,
            'Subscriber Process Identifier' =   (P1),
            'Initiating Device Identifier' =     IUT,
            'Monitored Object Identifier' =      O1,
            'Time Remaining' =                   (any value appropriate for the Lifetime selected),
            'List of Values' = (the initial values for Present_Value, Status_Flags, Requested_Shed_Level,
                Start_Time, Shed_Duration, and Duty_Window)
5.  TRANSMIT BACnet-SimpleACK-PDU
6.  MAKE (Present_Value change)
7.  BEFORE **Notification Fail Time**
8.      RECEIVE ConfirmedCOVNotification-Request,
            'Subscriber Process Identifier' =    (P1),
            'Initiating Device Identifier' =     IUT,
            'Monitored Object Identifier' =      O1,
            'Time Remaining' =                   (any value appropriate for the Lifetime selected),
            'List of Values' = (updated values for Present_Value, Status_Flags, Requested_Shed_Level,
                Start_Time, Shed_Duration, and Duty_Window)
9.  IF (Status_Flags can be changed by some action) THEN {
10.     MAKE (Status_Flags change)
11.     BEFORE **Notification Fail Time**
12.         RECEIVE ConfirmedCOVNotification-Request,
                'Subscriber Process Identifier' =    (P1),
                'Initiating Device Identifier' =     IUT,
                'Monitored Object Identifier' =      O1,
                'Time Remaining' =                   (any value appropriate for the Lifetime selected),
                'List of Values' = (updated values for Present_Value, Status_Flags, Requested,
                    Shed_Level, Start_Time, Shed_Duration, and Duty_Window)
13.     TRANSMIT BACnet-SimpleACK-PDU
    }
14. REPEAT PROP1 = (Request_Shed_Level, Start_Time, Shed_Duration, Duty_Window) DO {
15.     WRITE O1, PROP1 = (any value that differs from the value of PROP1 in the last COV
Notification)
16.     BEFORE **Notification Fail Time**
17.         RECEIVE ConfirmedCOVNotification-Request,
                'Subscriber Process Identifier' =    (P1),
                'Initiating Device Identifier' =     IUT,
                'Monitored Object Identifier' =      O1,
                'Time Remaining' =                   (any value appropriate for the Lifetime selected),
                'List of Values' = (updated values for Present_Value, Status_Flags,
Requested_Shed_Level,
                    Start_Time, Shed_Duration, and Duty_Window)
    }
```

### 8.2.X2 Change of Value Notification from Other Standard Object Types

Reason for Change: No test for this functionality.

Purpose: To verify that the IUT can initiate ConfirmedCOVNotification service requests conveying a change of value for an object (O1) not listed in 135-2020 Table 13-1.

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 value of the Present_Value is changed, and it is verified that a COV notification is received. If the Status_Flags is present and can be made to change, it is verified that when Status_Flags changes a COV notification is received.

Configuration Requirements: None.

Test Steps:

```
1.  TRANSMIT SubscribeCOV-Request,
        'Subscriber Process Identifier' =     (P1, 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
4.      RECEIVE ConfirmedCOVNotification-Request,
            'Subscriber Process Identifier' =     (P1),
            'Initiating Device Identifier' =      IUT,
            'Monitored Object Identifier' =       O1,
            'Time Remaining' =                    (any value appropriate for the Lifetime selected),
            'List of Values' = (the initial values for Present_Value and Status_Flags (if O1 supports Status_Flags))
5.  TRANSMIT BACnet-SimpleACK-PDU
6.  MAKE (Present_Value change)
7.  BEFORE Notification Fail Time
8.      RECEIVE ConfirmedCOVNotification-Request,
            'Subscriber Process Identifier' =     (P1),
            'Initiating Device Identifier' =      IUT,
            'Monitored Object Identifier' =       O1,
            'Time Remaining' =                    (any value appropriate for the Lifetime selected),
            'List of Values' = (updated values for Present_Value and Status_Flags (if O1 supports Status_Flags))
9.  IF (Status_Flags are present and can be changed by some action) THEN {
10.     MAKE (Status_Flags change)
11.     BEFORE Notification Fail Time
12.         RECEIVE ConfirmedCOVNotification-Request,
                'Subscriber Process Identifier' =     (P1),
                'Initiating Device Identifier' =      IUT,
                'Monitored Object Identifier' =       O1,
                'Time Remaining' =                    (any value appropriate for the Lifetime selected),
                'List of Values' = (updated values for Present_Value, Status_Flags)
13.     TRANSMIT BACnet-SimpleACK-PDU
    }
```

### 8.3 UnconfirmedCOVNotification Service Initiation Tests

#### 8.3.9 Unsubscribed Change of Value Notifications

Reason for Change: Add Process ID Requirement and Abort Conditionality to test.

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~~*shall have* a value of zero.

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**
3. RECEIVE UnconfirmedCOVNotification-Request,
   'Subscriber Process Identifier' = ~~(any valid process ID)~~*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)

**8.3.X1 Change of Value Notification from a Load Control Object (UnconfirmedCOVNotification)**
Reason for Change: No test for this functionality.

Purpose: To verify that the IUT can initiate UnconfirmedCOVNotification service requests conveying a change of the value for a Load Control object when the properties specified in the standard change.

Test Steps: The steps for this test case are identical to the test steps in 8.2.X1 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.

**8.3.X2 Change of Value Notification from Other Standard Object Types**
Reason for Change: No test for this functionality.

Purpose: To verify that the IUT can initiate UnconfirmedCOVNotification service requests conveying a change of value for an object (O1) not listed in 135-2020 Table 13-1.

Test Steps: The steps for this test case are identical to the test steps in 8.2.X2 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**

**8.4.8 CHANGE_OF_LIFE_SAFETY Tests (ConfirmedEventNotifications)**

**8.4.8.X1 Re-alerting CHANGE_OF_LIFE_SAFETY Transitions**
Reason for Change: No test exists for this functionality.

Purpose: This test case verifies the correct operation of the CHANGE_OF_LIFE_SAFETY event algorithm for objects re-alerting OFFNORMAL or LIFE_SAFETY_ALARM event states.

Test Concept: The object begins the test in the NORMAL state. The conditions are made which would cause a transition to a LIFE_SAFETY_ALARM or OFFNORMAL state, E1. Then, the conditions are made which cause the device to re-alert its current state (usually resulting from no alarm acknowledgement or transition away from the current state within some vendor defined time period). If the device does not support re-alerting of OFFNORMAL nor LIFE_SAFETY_ALARM states, then this test shall be skipped.

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

Test Steps:

```
1.  VERIFY Event_State = NORMAL
2.  MAKE (the conditions exist which will cause a transition to E1)
3.  BEFORE Notification Fail Time
4.      RECEIVE ConfirmedEventNotification-Request,
            'Process Identifier' =            (P1: any valid process ID),
            'Initiating Device Identifier' =  IUT,
            'Event Object Identifier' =       (O1: object being tested),
            'Time Stamp' =                    (T1: the IUT's current time),
            'Notification Class' =            (NC1: the configured notification class),
            'Priority' =                      (PTY1: 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' =                   (NT1: the configured notify type),
            'AckRequired' =                   (AR1: TRUE | FALSE),
            'From State' =                    NORMAL,
            'To State' =                      E1,
            'Event Values' =                  (EVS1: pMonitoredValue, pMode, pStatusFlags,
                                              pOperationExpected)
5.  MAKE(wait for IUT to send each of its Number_Of_APDU_Retries)
6.  MAKE (the conditions exist which will cause the re-alert of the current state E1)
7.  BEFORE Notification Fail Time
8.      RECEIVE ConfirmedEventNotification-Request,
            'Process Identifier' =            P1,
            'Initiating Device Identifier' =  IUT,
            'Event Object Identifier' =       O1,
            'Time Stamp' =                    (T2: any valid time stamp > T1),
            'Notification Class' =            NC1,
            'Priority' =                      PTY1,
            'Event Type' =                    CHANGE_OF_LIFE_SAFETY,
            'Message Text' =                  (S2: if provided in the original notification),
            'Notify Type' =                   NT1,
            'AckRequired' =                   AR1,
            'From State' =                    E1,
            'To State' =                      E1,
            'Event Values' =                  EVS1
9.  TRANSMIT BACnet-SimpleACK-PDU
```

### 8.4.X19 Multiple Access Events with either Denied or Granted Access Event Value (ConfirmedEventNotification)

Reason for Change: No test exists for this functionality.

Purpose: To verify that the last access event generated by the Access Point object is either denied or granted event value.
Test Concept: The test concept corresponds to 8.5.X19.
Configuration Requirements: The configuration requirements are identical to those in 8.5.X19.

Test Steps: The test steps for this test case are identical to the test steps in 8.5.X19, 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.X19, except that the event notifications shall be conveyed using a ConfirmedEventNotification service request.

## 8.5 UnconfirmedEventNotification Service Initiation Tests

### 8.5.17 CHANGE_OF_RELIABILITY Tests (UnconfirmedEventNotification)

#### 8.5.17.10 After FAULT-to-NORMAL, Re-Notification of OFFNORMAL (UnconfirmedEventNotifications)

Reason for Change: Clarifying that the same offnormal state should be used throughout the test. Account for cases where pTimeDelay=0.

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~~offnormal conditions. Make O1 transition to an ~~OFFNORMAL~~*offnormal* state (*S1)* 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~~*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. *If writable or configurable, set the value of the Time_Delay property in O1 to a value greater than zero.* 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 offnormal state~~*S1*)
4. WAIT pTimeDelay
5. BEFORE **Notification Fail Time**
6. 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~~*S1*,
   'Event Values' = (property-values appropriate for O1 )
7. VERIFY pCurrentState = ~~OFFNORMAL~~*S1*
8. MAKE(O1 enter a fault state)
9. BEFORE **Notification Fail Time**
10. RECEIVE UnconfirmedEventNotification-Request
    'Process Identifier' = (any valid process identifier),
    'Initiating Device Identifier' = IUT,
    'Event Object Identifier' = O1,
    'Time Stamp' = (any valid timestamp),
    '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~~S1,
          'To State' =      FAULT,
          'Event Values' =  ((R1 any valid BACnetReliability other than
                NO_FAULT_DETECTED),
                (~~?~~T, T, ?, ?),
                (A list of valid values for properties required to be reported for O1,
                and 0 or more other properties of O1))
11. MAKE(O1 clear the fault condition)
12. BEFORE Notification Fail Time
13.     RECEIVE UnconfirmedEventNotification-Request
          'Process Identifier' =   (any valid process identifier),
          'Initiating Device Identifier' = IUT,
          'Event Object Identifier' =    O1,
          'Time Stamp' =    (TS1, 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))
14. IF (pTimeDelay > 0) THEN {
15.         VERIFY pCurrentReliability = NO_FAULT_DETECTED
16.     VERIFY pCurrentState = NORMAL
17.     WAIT ~~until TS1 + ~~pTimeDelay
    }
18. BEFORE Notification Fail Time
19.     RECEIVE UnconfirmedEventNotification-Request
          'Process Identifier' =   (any valid process identifier),
          'Initiating Device Identifier' = IUT,
          'Event Object Identifier' =    O1,
          'Time Stamp' =    (any valid time stamp),
          'Notification Class' =  (the notification class configured for O1),
          'Priority' =    (the value configured for the transition),
          'Event Type' =    ET1,
          'Message Text' =  (optional, any valid message text),
          'Notify Type' =   ALARM | EVENT,
          'AckRequired' =   TRUE | FALSE,
          'From State' =    NORMAL,
          'To State' =      ~~OFFNORMAL~~S1,
          'Event Values' =  (property-values appropriate for O1)
20. VERIFY pCurrentReliability = NO_FAULT_DETECTED
21. VERIFY pCurrentState = ~~OFFNORMAL~~S1
```

### 8.5.X19 Multiple Access Events with either Denied or Granted Access Event Value (UnconfirmedEventNotification)

Reason for Change: No test exists for this functionality.

Purpose: To verify that the last access event generated by the Access Point object is either denied or granted event value.

Test Concept: The IUT’s Access Point object is configured such that a single access transaction generates multiple Access Events with either one of the event values being denied or granted.

Configuration Requirements: This test uses the standard configuration for Access Point objects, refer to 7.3.2.44 for further information. This test also requires the following additional configuration:
a) The IUT shall be configured with at least one instance of Access Point object which generates ConfirmedEventNotification and UnconfirmedEventNotification service requests describing multiple Access Events with denied or granted event values.

Test Steps:

```
1.  READ EventTag = Access_Event_Tag
2.  MAKE (the IUT perform a single access transaction, generating multiple access events which includes
denied or granted event value)

3.  REPEAT X = (1 .. number of expected events) DO {
4.      BEFORE Notification Fail Time
5.          RECEIVE UnconfirmedEventNotification-Request,
                'Process Identifier' =          (the configured process ID),
                'Initiating Device Identifier' =    IUT,
                'Event Object Identifier' =         O1,
                'Time Stamp' =                  (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' =            {pAccessEvent stored in N[X]}
6.      VERIFY pAccessEventTag = EventTag + 1
    }
7.  CHECK (N[last entry] is GRANTED or any valid denied reason)
```

### 8.24 DeviceCommunicationControl Service Initiation Tests

### 8.24.2 Indefinite Duration, Disable, Password

Reason For Change: The test is requesting the deprecated value DISABLE for the ‘Enable/Disable’ parameter.

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~~ *DISABLE-INITIATION*,
    'Password' = (a password of up to 20 characters)
2. TRANSMIT BACnet-SimpleACK-PDU

### 8.24.3 Time Duration, Disable, Password

Reason For Change: The test is requesting the deprecated value DISABLE for the ‘Enable/Disable’ parameter.

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~~ *DISABLE-INITIATION*,
        'Password' = (a password of up to 20 characters)
2. TRANSMIT BACnet-SimpleACK-PDU

### 8.24.6 Time Duration, Disable, No Password

Reason For Change: The test is requesting the deprecated value DISABLE for the ‘Enable/Disable’ parameter.

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~~ *DISABLE-INITIATION*,
2. TRANSMIT BACnet-SimpleACK-PDU

## 8.32 Who-Has Service Initiation Tests

### 8.32.1 Object Identifier Selection with no Device Instance Range

Reason for Change: The BACnet standard (per addendum 135-2012ar-5) now allows the IUT to send and receive a unicast response.

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 = *TD* | LOCAL BROADCAST | GLOBAL BROADCAST
        SOURCE = IUT,
        Who-Has-Request,
        'Object Identifier' = Object1
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)

### 8.32.2 Object Name Selection with no Device Instance Range

Reason for Change: The BACnet standard (per addendum 135-2012ar-5) now allows the IUT to send and receive a unicast response.

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 = *TD* | LOCAL BROADCAST | GLOBAL BROADCAST,
   SOURCE = IUT,
   Who-Has-Request,
   'Object Name' = V1
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)

### 8.32.3 Object Identifier Selection with a Device Instance Range

Reason for Change: The allowance for Unicast I-Have is added.

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 = *TD* | LOCAL BROADCAST | GLOBAL BROADCAST,
   SOURCE = IUT,
   Who-Has-Request,
   'Device Instance Range Low Limit' = (any integer X: ~~1~~*0* <= X <= 'Device Instance Range High Limit'),
   'Device Instance Range High Limit' = (any integer Y: 'Device Instance Range Low Limit' <= Y <= 4,194,303),
   'Object Identifier' = Object1
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)

### 8.32.4 Object Name Selection with a Device Instance Range

Reason for Change: The allowance for Unicast I-Have is added.

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 = *TD* | LOCAL BROADCAST | GLOBAL BROADCAST,
    SOURCE = IUT,
    Who-Has-Request,
    'Device Instance Range Low Limit' = (any integer X: ~~1~~*0* <= X <= 'Device Instance Range High Limit'),
    'Device Instance Range High Limit' = (any integer Y: 'Device Instance Range Low Limit' <= Y <= 4,194,303),
    'Object Name' = V1
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.X35 Who-Am-I Service Initiation Tests

### 8.X35.1 Responds to Who-Is With Who-Am-I While in the Unconfigured State

Reason for Change: No test exists for this functionality.

Purpose: To verify the IUT will send a Who-Am-I while in the unconfigured state when a Who-Is is received.

Test Concept: TD sends a Who-Is using the wildcard instance of 4194303 and the IUT responds with a Who-Am-I Request.

Test Configuration: The IUT's Device object is not configured with an object instance number. The IUT is configured with a MAC address. If the IUT does not support having a MAC address but no configured Device object instance number, this test shall be skipped.

Notes to Tester: The destination address used by TD shall be selected such that the IUT will receive the messages.

Test Steps:

1. TRANSMIT
    DESTINATION = IUT | LOCAL BROADCAST | GLOBAL BROADCAST | REMOTE BROADCAST,
    Who-Is Request,
    'Device Instance Range Low Limit' = 4194303,
    'Device Instance Range High Limit' = 4194303

```
2.  BEFORE **Unconfirmed Response Fail Time**
        RECEIVE
            DESTINATION = TD | LOCAL BROADCAST | GLOBAL BROADCAST | REMOTE BROADCAST,
            Who-Am-I Request,
            'Vendor Identifier' = (the IUT's Vendor_Identifier),
            'Model Name' = (the IUT's Model_Name),
            'Serial Number' = (the IUT's Serial_Number)
```

### 8.X36 You-Are Service Initiation Tests

#### 8.X36.1 Configures Other Device's MAC Address

Reason for Change: No test exists for this functionality.

Purpose: To verify the IUT can configure another device's MAC address using the You-Are service without relying on the Who-Am-I service.

Test Concept: The IUT configures TD with an appropriate MAC address without TD first sending a Who-Am-I.

Configuration Requirements: TD's Device object is configured with a known Device object instance number and no configured MAC address.

Notes to Tester: The IUT may require the tester to specify all the parameters needed to configure TD with You-Are, using the IUT's software. The destination address used by TD shall be selected such that the IUT will receive the messages.

Test Steps:

```
1.  MAKE (the IUT configure TD)
2.  RECEIVE
        DESTINATION = LOCAL BROADCAST | GLOBAL BROADCAST | REMOTE BROADCAST,
        You-Are Request,
        'Vendor Identifier' = (TD's Vendor_Identifier),
        'Model Name' = (TD's Model_Name),
        'Serial Number' = (TD's Serial_Number),
        'Device Identifier' = (TD's Device object),
        'Device MAC Address' = (an appropriate MAC address)
3.  IF (TD is not an MS/TP subordinate node)
4.      TRANSMIT
            DESTINATION = IUT | LOCAL BROADCAST | GLOBAL BROADCAST | REMOTE BROADCAST,
            I-Am Request,
            'Device Identifier' = (TD's Device object)
            'Max APDU Length Accepted' = (any valid value),
            'Segmentation Supported' = (any valid value),
            'Vendor Identifier' = (TD's Vendor_Identifier)
```

#### 8.X36.2 Configures Other Device's Object Instance Number and MAC Address

Reason for Change: No test exists for this functionality.

Purpose: To verify the IUT can configure another Device's object instance number and MAC address using the You-Are service without relying on the Who-Am-I service.

Test Concept: The IUT configures TD with an appropriate Device object instance number (X) and MAC address without TD first sending a Who-Am-I.

Configuration Requirements: TD is not configured with a Device object instance number and has no configured MAC address.

Notes to Tester: The IUT will require the tester to specify all the parameters needed to configure TD with You-Are, using the IUT's software. The destination address used by TD shall be selected such that the IUT will receive the messages.

Test Steps:

```
1.  MAKE (the IUT configure TD)
2.  BEFORE **Internal Processing Fail Time**
3.      RECEIVE
            DESTINATION = LOCAL BROADCAST | GLOBAL BROADCAST | REMOTE BROADCAST,
            You-Are Request,
            'Vendor Identifier' = (TD's Vendor_Identifier),
            'Model Name' = (TD's Model_Name),
            'Serial Number' = (TD's Serial_Number),
            'Device Identifier' = (Device, X),
            'Device MAC Address' = (any valid MAC address)
4.  IF (TD is not an MS/TP subordinate node)
5.      TRANSMIT
            DESTINATION = IUT | LOCAL BROADCAST | GLOBAL BROADCAST | REMOTE BROADCAST,
            I-Am Request,
            'Device Identifier' = (Device, X),
            'Max APDU Length Accepted' = (any valid value),
            'Segmentation Supported' = (any valid value),
            'Vendor Identifier' = (TD's Vendor_Identifier)
```

### 8.X36.3 Can Unconfigure Devices

Reason for Change: No test exists for this functionality.

Purpose: To verify the IUT can unconfigure a device using the You-Are service.

Test Concept: The IUT transmits a You-Are with 'Device Identifier' = (Device, 4194303) in order to unconfigure TD.

Test Steps:

```
1.  MAKE (the IUT unconfigure TD)
2.  BEFORE **Internal Processing Fail Time**
3.      RECEIVE
            DESTINATION = TD | LOCAL BROADCAST | GLOBAL BROADCAST | REMOTE BROADCAST,
            You-Are Request,
            'Vendor Identifier' = (TD's Vendor_Identifier),
            'Model Name' = (TD's Model_Name),
            'Serial Number' = (TD's Serial_Number),
            'Device Identifier' = (Device, 4194303),
            'Device MAC Address' = (any valid MAC address, or absent)
```

## 135.1-2023*u*-4 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 135.1-2023. In addition, the test coverage is increased with the addition of new tests.

## 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

### 9.1.2 Negative AcknowledgeAlarm Service Execution Tests

### 9.1.2.1 Unsuccessful Alarm Acknowledgment of Confirmed Event Notifications Because the 'Time Stamp' is Too Old

Reason for Change: Changes required dur to 135-2016br-4 and deprecation of the 'time' form of the BACnetTimeStamp datatype.

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 (D1)
3. BEFORE **Notification Fail Time**
4. RECEIVE ConfirmedEventNotification-Request,
    'Process Identifier' = (*PID:* the process identifier configured for this event),
    'Initiating Device Identifier' = IUT,
    'Event Object Identifier' = (*O1:* the object detecting the alarm),
    'Time Stamp' = (T1: any valid time stamp),
    'Notification Class' = (*NC:* the notification class configured for this event),
    'Priority' = (*P1:* the priority configured for this event type),
    'Event Type' = (*E1:* any valid event type),
    'Message Text' = (*MT:* optional, any valid message text),
    'Notify Type' = (*NT:* the notify type configured for the event),
    'AckRequired' = TRUE,
    'From State' = (*S1:* any appropriate event state),
    'To State' = (~~S1~~*S2*: any appropriate event state),
    'Event Values' = (the values appropriate to the event type)
5. TRANSMIT BACnet-SimpleACK-PDU
6. RECEIVE
    DESTINATION = (a device other than the TD),
    SOURCE = IUT,
    ConfirmedEventNotification-Request,
    'Process Identifier' = (*PID*~~the process identifier configured for this event~~),
    'Initiating Device Identifier' = IUT,
    'Event Object Identifier' = (*O1*~~the object detecting the alarm~~),
    'Time Stamp' = (T1),
    'Notification Class' = (*NC*~~the notification class configured for this event~~),
    'Priority' = (*P1*~~the priority configured for this event type~~),
    'Event Type' = (*E1*~~E2: any valid event type~~),
    'Message Text' = (*MT* optional~~, any valid message text~~),
    'Notify Type' = (*NT*~~the notify type configured for the event~~),
    'AckRequired' = TRUE,
    'From State' = (*S1*~~any appropriate event state~~),
    'To State' = (S2~~: any appropriate event state~~),

```
            'Event Values' =                  (the values appropriate to the event type)
7.  TRANSMIT BACnet-SimpleACK-PDU
8.  VERIFY ~~(the 'Event Object Identifier' from the event notification)~~*O1*,
        Acked_Transitions = (appropriate bit FALSE, the others TRUE)
9.  TRANSMIT AcknowledgeAlarm-Request,
        'Acknowledging Process Identifier' =  (P1:~~the value of the 'Process Identifier' parameter in the event notification~~),
        'Event Object Identifier' =           (*O1*~~the 'Event Object Identifier' from the event notification~~),
        'Event State Acknowledged' =          (*S2*~~the state specified in the 'To State' parameter of the notification~~),
        'Time Stamp' =                        (any valid time stamp older than T1),
        'Acknowledgment Source' =             (any valid value)
        'Time of Acknowledgment' =            (the current time ~~using a Time format~~)
10. RECEIVE BACnet-Error-PDU
            Error Class =       SERVICES,
            Error Code =        INVALID_TIME_STAMP
11. VERIFY ~~(the 'Event Object Identifier' from the event notification)~~*O1*,
            Acked_Transitions = (appropriate bit FALSE, the others TRUE)
12. TRANSMIT AcknowledgeAlarm-Request,
            'Acknowledging Process Identifier' =  (*PID*~~the process identifier configured for this event~~),
            'Event Object Identifier' =           (*E1*~~the 'Event Object Identifier' from the event notification~~),
            'Event State Acknowledged' =          (*S2*~~the state specified in the 'To State' parameter of the notification~~),
            'Time Stamp' =                        T1,
            'Time of Acknowledgment' =            (the current time ~~using a Time format~~)
13. RECEIVE BACnet-SimpleACK-PDU
~~14. IF (Protocol_Revision is present AND Protocol_Revision >= 1) THEN~~
14. BEFORE **Notification Fail Time**
15. RECEIVE
            ConfirmedEventNotification-Request,
                'Process Identifier' =            (*PID*~~the process identifier configured for this event~~),
                'Initiating Device Identifier' =  IUT,
                'Event Object Identifier' =       (*O1*~~the object detecting the alarm~~),
                'Time Stamp' =                    (T2: any valid time stamp),
                'Notification Class' =            (*NC*~~the notification class configured for this event~~),
                'Priority' =                      (*P1*~~the priority configured for this event type~~),
                'Event Type' =                    (*E1*~~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~~
~~                ConfirmedEventNotification-Request,~~
~~                'Process Identifier' =            (the process identifier configured for this event),~~
~~                'Initiating Device Identifier' =  IUT,~~
~~                'Event Object Identifier' =       (O1the 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' =                    (*E1*any valid event type),~~
~~                'Message Text' =                  (optional, any valid message text),~~
~~                'Notify Type' =                   ACK_NOTIFICATION~~
16. TRANSMIT BACnet-SimpleACK-PDU
```

```
~~15. IF (Protocol_Revision is present AND Protocol_Revision >= 1) THEN~~
17. RECEIVE
        DESTINATION = (a device other than the TD),
        SOURCE = IUT,
        ConfirmedEventNotification-Request,
            'Process Identifier' =               (PID~~the process identifier configured for this event~~),
            'Initiating Device Identifier' =     IUT,
            'Event Object Identifier' =          (O1~~the object detecting the alarm~~),
            'Time Stamp' =                       (T2),
            'Notification Class' =                   (NC ~~the notification class configured for this event~~),
            'Priority' =                         (P1~~the priority configured for this event type~~),
            'Event Type' =                       (E1~~any valid event type~~),
            'Message Text' =                     (optional, any valid message text),
            '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' =          (O1the object detecting the alarm),~~
~~            'Time Stamp' =                       (T2),~~
~~            'Notification Class' =               (NC the notification class configured for this event),~~
~~            'Priority' =                         (the priority configured for this event type),~~
~~            'Event Type' =                       (E1any valid event type),~~
~~            'Message Text' =                     (optional, any valid message text),~~
~~            'Notify Type' =                      ACK_NOTIFICATION~~
18. TRANSMIT BACnet-SimpleACK-PDU
19. VERIFY ~~(the 'Event Object Identifier' from the event notification)~~O1, Acked_Transitions =
(TRUE,TRUE,TRUE)
```

### 9.2 ConfirmedCOVNotification Service Execution Tests

### 9.2.1 Positive ConfirmedCOVNotification Service Execution Tests

### 9.2.1.X4 Change of Value Notification from Proprietary Objects

*This test has not been developed and shall be skipped.*

### 9.3 UnconfirmedCOVNotification Service Execution Tests

### 9.3.X9 Change of Value Notification from Proprietary Objects

*This test has not been developed and shall be skipped.*

### 9.9 LifeSafetyOperation Service Execution Test

### 9.9.1 Positive LifeSafetyOperation Execution Tests

### 9.9.1.3 Silencing/Unsilencing Execution Tests

Reason for Change: Allow the IUT to use some other mechanism to indicate audible and visual notifications.

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 life safety object, O1, enters a non-normal state and an audible and/or visual indication, I1, occurs.* A LifeSafetyOperation service request is transmitted to silence *I1 and I1 is verified to be silenced. A second LifeSafetyOperation service request is transmitted to unsilence I1 and I1 is verified to be unsilenced. The Silenced property is also validated.*~~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.

| ***Audible Only Indication*** | | | | |
|---|---|---|---|---|
| ***Silence Request (S)*** | ***Allowable Silenced State (S_State)*** | | ***Unsilenced Request (U)*** | ***Allowable Silenced State (U_STATE)*** |
| *SILENCE* | *ALL_SILENCED, AUDIBLE_SILENCED* | | *UNSILENCE* | *UNSILENCED, proprietary* |
| *SILENCE_AUDIBLE* | *ALL_SILENCED, AUDIBLE_SILENCED* | | *UNSILENCE_AUDIBLE* | *UNSILENCED, proprietary* |
| *SILENCE_VISUAL* | *N/A* | | *UNSILENCE_VISUAL* | *N/A* |

| ***Visual Only Indication*** | | | | |
|---|---|---|---|---|
| ***Silence Request (S)*** | ***Allowable Silenced State (S_State)*** | | ***Unsilenced Request (U)*** | ***Allowable Silenced State (U_State)*** |
| *SILENCE* | *ALL_SILENCED, VISUAL_SILENCED* | | *UNSILENCE* | *UNSILENCED, proprietary* |
| *SILENCE_AUDIBLE* | *N/A* | | *UNSILENCE_AUDIBLE* | *N/A* |
| *SILENCE_VISUAL* | *ALL_SILENCED, VISUAL_SILENCED* | | *UNSILENCE_VISUAL* | *UNSILENCED, proprietary* |

| ***Audible and Visual Indication*** | | | | |
|---|---|---|---|---|
| ***Silence Request (S)*** | ***Allowable Silenced State (S_State)*** | | ***Unsilenced Request (U)*** | ***Allowable Silenced State (U_State)*** |
| *SILENCE* | *ALL_SILENCED* | | *UNSILENCE* | *UNSILENCED, proprietary* |
| *SILENCE_AUDIBLE* | *AUDIBLE_SILENCED* | | *UNSILENCE_AUDIBLE (all silenced)* | *SILENCED_VISUAL, proprietary* |
| | | | *UNSILENCE_AUDIBLE (audible silenced, visual active)* | *UNSILENCED, proprietary* |
| | | | *UNSILENCE_AUDIBLE* | *N/A* |

| | | | |
|---|---|---|---|
| | | *(audible active, visual silenced)* | |
| *SILENCE_VISUAL* | *VISUAL_SILENCED* | *UNSILENCE_VISUAL (all silenced)* | *SILENCED_AUDIBLE, proprietary* |
| | | *UNSILENCE_VISUAL (audible silenced, visual active)* | *N/A* |
| | | *UNSILENCE_VISUAL (audible active, visual silenced)* | *UNSILENCED, proprietary* |

| ~~**Only Sounder Attached**~~ | | | |
|---|---|---|---|
| ~~Silence Operation~~ | ~~Allowable Silenced State~~ | ~~Unsilenced Operation~~ | ~~Allowable Silenced State~~ |
| ~~SILENCE~~ | ~~ALL_SILENCED, AUDIBLE_SILENCED, proprietary~~ | ~~UNSILENCE~~ | ~~UNSILENCED, proprietary~~ |
| ~~SILENCE_AUDIBLE~~ | ~~ALL_SILENCED, AUDIBLE_SILENCED, proprietary~~ | ~~UNSILENCE_AUDIBLE~~ | ~~UNSILENCED, proprietary~~ |
| ~~SILENCE_VISUAL~~ | ~~N/A~~ | ~~UNSILENCE_VISUAL~~ | ~~N/A~~ |

| ~~**Only Strobe Attached**~~ | | | |
|---|---|---|---|
| ~~Silence Operation~~ | ~~Allowable Silenced State~~ | ~~Unsilenced Operation~~ | ~~Allowable Silenced State~~ |
| ~~SILENCE~~ | ~~ALL_SILENCED, VISUAL_SILENCED, proprietary~~ | ~~UNSILENCE~~ | ~~UNSILENCED, proprietary~~ |
| ~~SILENCE_AUDIBLE~~ | ~~N/A~~ | ~~UNSILENCE_AUDIBLE~~ | ~~N/A~~ |
| ~~SILENCE_VISUAL~~ | ~~ALL_SILENCED, VISUAL_SILENCED, proprietary~~ | ~~UNSILENCE_VISUAL~~ | ~~UNSILENCED, proprietary~~ |

| ~~**Sounder And Strobe Attached**~~ | | | |
|---|---|---|---|
| ~~Silence Operation~~ | ~~Allowable Silenced State~~ | ~~Unsilenced Operation~~ | ~~Allowable Silenced State~~ |
| ~~SILENCE~~ | ~~ALL_SILENCED, proprietary~~ | ~~UNSILENCE~~ | ~~UNSILENCED, proprietary~~ |
| ~~SILENCE_AUDIBLE~~ | ~~AUDIBLE_SILENCED, proprietary~~ | ~~UNSILENCE_AUDIBLE (all silenced)~~ | ~~SILENCED_VISUAL, proprietary~~ |
| | | ~~UNSILENCE_AUDIBLE~~ | ~~UNSILENCED, proprietary~~ |

| | | | |
|---|---|---|---|
| | | ~~(audible silenced, visual active)~~ | |
| | | ~~UNSILENCE_AUDIBLE (audible active, visual silenced)~~ | ~~N/A~~ |
| ~~SILENCE_VISUAL~~ | ~~VISUAL_SILENCED, proprietary~~ | ~~UNSILENCE_VISUAL (all silenced)~~ | ~~SILENCED_AUDIBLE, proprietary~~ |
| | | ~~UNSILENCE_VISUAL (audible silenced, visual active)~~ | ~~N/A~~ |
| | | ~~UNSILENCE_VISUAL (audible active, visual silenced)~~ | ~~UNSILENCED, proprietary~~ |

Configuration Requirements: The IUT must *support an indication that audible and/or visual equipment has been the silenced and unsilenced*~~be fitted with needed audible and visual equipment~~.

Notes to Tester: *The indication of silencing and unsilencing an audible and visual change is vendor specific but must be detectable by the lab during testing*~~Source object needs to get silence only for configured objects~~.

Test Steps:

```
REPEAT S = (for each supported LifeSafetyOperation service Request specified in the above table) DO {
1.  MAKE (O1 enter a condition, that S can silence)
2.  VERIFY Silenced = (UNSILENCED or a proprietary value with a similar semantic)
3.  TRANSMIT LifeSafetyOperation-Request,
        'Requesting Process Identifier' = (any valid identifier),
        'Requesting Source' =             (any valid character string),
        'Request' =                           S,
        'Object Identifier' =             (O1 or absent)
4.  RECEIVE BACnet-SimpleACK-PDU
5.  CHECK (I1 is silenced)
6.  VERIFY Silenced = (S_State)
7.  TRANSMIT LifeSafetyOperation-Request,
        'Requesting Process Identifier' = (any valid identifier),
        'Requesting Source' =             (any valid character string),
        'Request' =                           U,
        'Object Identifier' =             (O1 or absent)
8.  RECEIVE BACnet-SimpleACK-PDU
9.  CHECK (I1 is unsilenced)
10. VERIFY Silenced = (U_State)
}
```

~~REPEAT X = (All supported enumerations that silence the object) DO {~~
~~1. MAKE (the selected object enter a state where enumeration X will silence the sounder/strobe)~~
~~2. VERIFY Silenced = (Unsilenced or a proprietary value with a similar semantic)~~
~~3. TRANSMIT LifeSafetyOperation-Request,~~
~~'Requesting Process Identifier' = (any valid identifier),~~
~~'Requesting Source' = any valid character string),~~
~~'Request' = X,~~
~~'Object Identifier' = (absent or the selected object)~~
~~4. RECEIVE BACnet-SimpleACK-PDU~~
~~5. CHECK (that the sounder/strobe is inactive)~~

~~6. VERIFY Silenced = (an allowable silenced state based on the IUT setup and operation request X)~~
~~7. 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)~~
~~8. RECEIVE BACnet-SimpleACK-PDU~~
~~9. CHECK (the sounder / strobe active again, as appropriate to the operation)~~
~~10. VERIFY Silenced = (the appropriate state based on the operation and IUT condition)~~
~~}~~

### 9.9.2 Negative LifeSafetyOperation Execution Tests

### 9.9.2.X1 LifeSafetyOperation on an Object Which Does Not Support the Operation Specified in the 'Request' Parameter

Reason for Change: No test exists for this functionality. This test is not in any SSPC proposal.

Purpose: To verify that the IUT correctly responds when a LifeSafetyOperation is received that targets an object which does not support the operation specified in the 'Request' parameter.

Test Concept: Send a LifeSafetyOperation request, with an Object Identifier referencing an object in the IUT which does not support the operation specified in the 'Request' parameter.

Test Steps:

1. TRANSMIT LifeSafetyOperation-Request,
   'Requesting Process Identifier' = (any valid value),
   'Requesting Source' = (any valid value),
   'Request' = (any valid value not supported by the referenced Object Identifier),
   'Object Identifier' = (an object in the IUT which does not support the operation specified in the 'Request' parameter.)
2. RECEIVE BACnet-Error PDU,
   'Error Class' = OBJECT,
   'Error Code' = VALUE_OUT_OF_RANGE

### 9.10 SubscribeCOV Service Execution Tests

### 9.10.1 Positive SubscribeCOV Service Execution Tests

### 9.10.1.8 Updating Existing Subscriptions

Reason for Change: Lifetime was too restrictive.

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 *approximately 300 seconds but not* greater than ~~60 but less than~~ 300 seconds.

Test Steps:

1. TRANSMIT SubscribeCOV-Request,

```
        'Subscriber Process Identifier' =       (PID1, any valid process identifier),
        'Monitored Object Identifier' =         (O1, 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
4.      BEFORE Notification Fail Time
5.          RECEIVE ConfirmedCOVNotification-Request,
                'Subscriber Process Identifier' =   PID1,
                'Initiating Device Identifier' =    IUT,
                'Monitored Object Identifier' =     O1,
                'Time Remaining' =                  (~60, but not greater than 60~~60 or less than 60~~),
                'List of Values' =              (values appropriate to the object type of the monitored object)
6.      TRANSMIT BACnet-SimpleACK-PDU
    ELSE
7.      BEFORE Notification Fail Time
8.          RECEIVE UnconfirmedCOVNotification-Request,
                'Subscriber Process Identifier' =   PID1,
                'Initiating Device Identifier' =    IUT,
                'Monitored Object Identifier' =     O1,
                'Time Remaining' =                  (~60, but not greater than 60),
                'List of Values' =              (values appropriate to the object type of the monitored object)
9.  TRANSMIT SubscribeCOV-Request,
        'Subscriber Process Identifier' =       PID1,
        'Monitored Object Identifier' =         O1,
        'Issue Confirmed Notifications' =       TRUE | FALSE,
        'Lifetime' =        (300~~T1, a value between 180 and 300 seconds~~)
10. RECEIVE BACnet-SimpleACK-PDU
11. IF (the subscription was for confirmed notifications) THEN
12.     BEFORE Notification Fail Time
13.         RECEIVE ConfirmedCOVNotification-Request,
                'Subscriber Process Identifier' =   PID1,
                'Initiating Device Identifier' =    IUT,
                'Monitored Object Identifier' =     O1,
                'Time Remaining' =                  (~300, but not greater than 300~~~T1, but not greater than T1~~),
                'List of Values' =              (values appropriate to the object type of the monitored object)
14.     TRANSMIT BACnet-SimpleACK-PDU
    ELSE
15.     BEFORE Notification Fail Time
16.         RECEIVE UnconfirmedCOVNotification-Request,
                'Subscriber Process Identifier' =   PID1,
                'Initiating Device Identifier' =    IUT,
                'Monitored Object Identifier' =     O1,
                'Time Remaining' =                  (~300, but not greater than 300~~~T1, but not greater than T1~~),
                'List of Values' =              (values appropriate to the object type of the monitored object)
```

**9.10.1.11 Ensuring 5 Concurrent COV Subscribers**

Reason For Change: Clarified the Test Concept and added Test Conditionality.

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 $V_5$ *for monitored object O1. After each subscription verify a corresponding notification is sent. Change the monitored* *object and verify that all 5 notifications were sent.*~~, and then check to ensure that 5 notifications are sent when the monitored object changes.~~

*Test Conditionality: The IUT should not have any subscriptions at the start of this test.*

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 {
2.          TRANSMIT SubscribeCOV-Request,
                'Subscriber Process Identifier' =       X,
                'Monitored Object Identifier' =         O1~~(any object supporting COV notifications)~~,
                'Issue Confirmed Notifications' =       TRUE | FALSE,
                'Lifetime' =    (any valid value that will allow the subscription to outlast the test)
3.          RECEIVE BACnet-SimpleACK-PDU
4.  IF (~~if~~ confirmed notifications were requested) THEN
5.          BEFORE Notification Fail Time
6.                  RECEIVE ConfirmedCOVNotification-Request,
                        'Subscriber Process Identifier' =       X,
                        'Initiating Device Identifier' =  IUT,
                        'Monitored Object Identifier' = O1~~(the same object used in the subscription)~~,
                        'Time Remaining' =          (any valid value),
                        'List of Values' =          (the initial Present_Value and initial Status_Flags)
7.                  TRANSMIT BACnet-SimpleACK-PDU
    ELSE
8.          BEFORE Notification Fail Time
9.                  RECEIVE UnconfirmedCOVNotification-Request,
                        'Subscriber Process Identifier' =       X,
                        'Initiating Device Identifier' =  IUT,
                        'Monitored Object Identifier' = O1~~(the same object used in the subscription)~~,
                        'Time Remaining' =          (any valid value),
                        'List of Values' =          (the initial Present_Value and initial Status_Flags)
    }
10. MAKE (Present_Value = any value that differs from "initial Present_Value" such that a COV notification would be generated)
11. REPEAT (X=$V_1$ to $V_5$) DO {
12.         IF (~~if~~ confirmed notifications were requested) THEN
13.                 RECEIVE ConfirmedCOVNotification-Request,
                        'Subscriber Process Identifier' =       X,
                        'Initiating Device Identifier' =  IUT,
                        'Monitored Object Identifier' = O1~~(the same object used in the subscription)~~,
                        ‘Time Remaining' =          (any valid value),
                        'List of Values' =          (the new Present_Value and Status_Flags)
14.                 TRANSMIT BACnet-SimpleACK-PDU
            ELSE
15.                 RECEIVE UnconfirmedCOVNotification-Request,
                        'Subscriber Process Identifier' =       X,
                        'Initiating Device Identifier' =  IUT,
                        'Monitored Object Identifier' = O1~~(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

### 9.10.2.3 There Is No Space For A Subscription

Reason for Change: Remove requirement to read Active_COV_Subscriptions property so that devices that do not support segmentation can be tested.

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:

```
1.    REPEAT PID = (1 through the maximum number of subscriptions the IUT can accept plus 1, or until
the IUT  returns an Error-PDU) {
2.          TRANSMIT SubscribeCOV-Request,
                  'Subscriber Process Identifier' =       PID,
                  'Monitored Object Identifier' = (any object of that supports COV),
                  'Issue Confirmed Notifications' =    TRUE,
                  'Lifetime' =                         6000
3.          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-SimpleACKBACnet-SimpleACK-PDU was received in step 2) THEN~~
~~            CHECK (that the subscription is in ACS)~~
~~      ELSE~~
~~              CHECK (that the subscription is not in ACS)~~
      }
```

## 9.20 ReadPropertyMultiple Service Execution Tests

### 9.20.1 Positive ReadPropertyMultiple Service Execution Tests

### 9.20.1.X2 ReadPropertyMultiple Array Properties

Reason for Change: No test exists for this functionality. This test is not in any SSPC proposal.

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.

Test Concept: The TD reads the size of the array property, and then reads the first and last entries in the array. Finally, the TD reads past the end of the array and ensures that the IUT returns the correct error.

Configuration Requirement: O1 is any object in the IUT database having array property P1.

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 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
    | ReadPropertyMultiple-ACK,
            'Object Identifier' = O1,
            'Property Identifier' = P1,
            'Property Array Index' = X+1,
            'Property Access Error' = (
                'Error Class' = PROPERTY,
                'Error Code' = INVALID_ARRAY_INDEX
            )
```

## 9.21 ReadRange Service Execution Tests

### 9.21.1 Positive ReadRange Service Execution Tests

#### 9.21.1.6 Reading a Range of Items that do not Exist *by Position*

Reason for Change: Fix the Result Flags parameter in step 2.

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 all known not to be in the list property P. 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 object configured for this test),
        'Property Identifier' =  P,
```

'Reference Index' = (any value x: x > N),
'Count' = (any value y: y > 0)

2. RECEIVE Read-Range-ACK,
'Object Identifier' = (the object configured for this test),
'Property Identifier' = P,
'Result flags' = {~~TRUE~~*FALSE*, ~~TRUE~~*FALSE*, FALSE},
'Item Count' = 0,
'Item Data' = (an empty list)

### 9.21.2 Negative ReadRange Service Execution Tests

### 9.21.2.X7 Attempting to Read a List Property by Sequence That Does not Have Sequence Numbers

Reason for Change: No test exists for this functionality. Tests added as per 135-2016-bu1

References: 15.8.1.3.1, 18.3

Purpose: To verify the correct execution of the ReadRange service request when the requested property does not support sequence numbers.

Test Concept: A ReadRange request is transmitted by the TD requesting a specified sequence number and count of items. The IUT shall respond by returning the appropriate error code. This test is only applied to devices with a Protocol_Revision of 21 or higher.

Test Configuration: A list property that does not support sequence numbers must be selected for this test. If no suitable property exists in the device, this test shall be skipped.

Test Steps:

1. TRANSMIT Read-Range-Request,
'Object Identifier' = (the object configured for this test),
'Property Identifier' = (the list property configured for this test),
'Reference Sequence Number' = 42,
'Count' = (any valid value)
2. RECEIVE BACnet-Error-PDU,
'Error Class' = PROPERTY,
'Error Code' = LIST_ITEM_NOT_NUMBERED

### 9.21.2.X8 Attempting to Read a List Property by ByTime That Does not Have Timestamps

Reason for Change: No test exists for this functionality. Tests added as per 135-2016-bu1

References: 15.8.1.3.1, 18.3

Purpose: To verify the correct execution of the ReadRange service request when the requested property does not support timestamps.

Test Concept: A ReadRange request is transmitted by the TD requesting a specified reference time and count of items. The IUT shall respond with an error.

Test Configuration: A list property that does not support timestamps must be selected for this test.

Test Steps:

1. TRANSMIT Read-Range-Request,
'Object Identifier' = (the object configured for this test),
'Property Identifier' = (the list property configured for this test),

```
        'Reference Time' = (any valid specific BACnetDateTime)
        'Count' = (any valid value)
2.  RECEIVE BACnet-Error-PDU,
        'Error Class' = PROPERTY,
        'Error Code’ = LIST_ITEM_NOT_TIMESTAMPED
    | RECEIVE BACnet-Error-PDU,
            'Error Class' = E (any valid Error Class)
            'Error Code' = (Any valid Error Code for class E)
```

## 9.22 WriteProperty Service Execution Tests

### 9.22.1 Positive WriteProperty Service Execution Tests

#### 9.22.1.X3 Writing NULL to Non-commandable Properties

Reason for Change: The standard was changed in PR21 to require that devices not return errors when a NULL is written to non-commandable properties and no test exists for this functionality.

Purpose: This test case verifies that the IUT returns a Result(+) when an attempt is made to relinquish a non-commandable property.

Test Concept: Write NULL to a writable non-commandable property, P1 in object O1, and verify the IUT returns a Result(+) and does not modify the property.

Test Configuration: P1 shall be a property for which NULL is not an accepted value.

Test Steps:

```
1. READ X = (O1), P1
2. TRANSMIT WriteProperty-Request,
        'Object Identifier' =    O1,
        'Property Identifier' =  P1,
        'Property Value' =       NULL
3. RECEIVE BACnet-SimpleACK-PDU
4. VERIFY (O1), P1 = X
```

### 9.22.2 Negative WriteProperty Service Execution Tests

#### 9.22.2.10 Resizing a writable fixed size array property

Reason for Change: WRITE was changed to TRANSMIT since the expected response is an ERROR.

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)~~
2. *TRANSMIT WriteProperty-Request,*
   *'Object Identifier' = O1,*
   *'Property Identifier' = P1,*
   *'Property Value' = (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)~~
*5. TRANSMIT WriteProperty-Request,*
   *'Object Identifier' = O1,*
   *'Property Identifier' = P1,*
   *'Property Value' = (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~~
*8. TRANSMIT WriteProperty-Request,*
   *'Object Identifier' = O1,*
   *'Property Identifier' = P1,*
   *'Property Value' = (any valid value greater than Array Size X),*
   *'Property 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~~
*11. TRANSMIT WriteProperty-Request,*
   *'Object Identifier' = O1,*
   *'Property Identifier' = P1,*
   *'Property Value' = (any valid value less than Array Size X),*
   *'Property 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

### 9.23.2.13 Resizing a Writable Fixed Size Array Property Using WritePropertyMultiple Service

Reason for Change: Making this test be consistent with 9.22.2.10 by adding WRITE_ACCESS_DENIED as allowed error.

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 |
WRITE_ACCESS_DENIED,
        '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 |
WRITE_ACCESS_DENIED,
        'ObjectIdentifier' = O1,
        'PropertyIdentifier' = P1
        'Property Array Index'= 0
13. VERIFY (O1), P1= X, ARRAY INDEX = 0
```

### 9.23 WritePropertyMultiple Service Execution Tests

### 9.23.2 Negative WritePropertyMultiple Service Execution Tests

### 9.23.2.12 WritePropertyMultiple Reject Test

Reason for Change: Fix Test Concept to define O1 and O2.

Purpose: This test case verifies that the IUT does not send a Reject-PDU after applying part of a WritePropertyMultiple.

Test Concept: *Object, O1, containing writable property, P1 and object O2, containing writable property, P2*~~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
```

**9.23.2.16 WritePropertyMultiple Reject Test for First Element of 'List of Write Access Specifications'**

Reason for Change: Add description of O1 in Test Concept.

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: *Object, O1, contains a writable property, P1 and object O2, contains a writable property, P2.* ~~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~~*they have* not changed.

Test Steps:

```
1.  ~~VERIFY (O1), P1= X~~*READ X = (O1), P1*
2.  ~~VERIFY (O2), P2=Y~~*READ X = (O2), P2*
3.  TRANSMIT WritePropertyMultiple-Request,
        'Object Identifier' = O1,
        'Property Identifier' = P1,
        -- 'Property Value' = (this field is missing including the opening and closing tags)
        '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
```

### 9.23.2.19 Date Non-Pattern Properties Test using WritePropertyMultiple Service

Reason for Change: Update Test Concept to include meaning of O1.

Purpose: To verify that the property being tested does not accept special date field values.

Test Concept: *O1 is the object being tested.* 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 {
1.      TRANSMIT WritePropertyMultiple-Request
                'Object Identifier' = O1,
                'Property Identifier' = P1,
                'Property Value' = (V1 updated with the special value SV)
2.      RECEIVE WritePropertyMultiple-Error,
                'Error Class' = PROPERTY,
                'Error Code' = VALUE_OUT_OF_RANGE,
                'Object Identifier' = O1,
                'Property Identifier' = P1)
            | (BACnet-Reject-PDU
                'Reject Reason' = INVALID_PARAMETER_DATATYPE)
            | (BACnet-Reject-PDU
                'Reject Reason'= INVALID_TAG)
    }
```

### 9.23.2.20 Time Non-Pattern Properties Test using WritePropertyMultiple Service

Reason for Change: Update Test Concept to include meaning of O1.

Purpose: To verify that the property being tested does not accept special time field values.

Test Concept: *O1 is the object being tested.* 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 {
1.          TRANSMIT WritePropertyMultiple-Request
                'Object Identifier' = O1,
                'Property Identifier' = P1,
                'Property Value' = (V1 updated with the special value SV)
2.          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)
    }
```

**9.23.2.21 DateTime Non-Pattern Properties Test using WritePropertyMultiple Service**

Reason for Change: Update Test Concept to include meaning of O1.

Purpose: To verify that the property being tested does not accept special date field values.

Test Concept: *O1 is the object being tested*. The property being tested, $P_1$, is written with each of the special datetime field values to ensure that the property does not accept them. A datetime $DT_1$ is selected which is within the range that the IUT will accept for the property. The value, $V_1$, written to the property is the datetime $DT_1$ 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 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, hour unspecified, minute
unspecified, second unspecified, hundredths unspecified) DO {
1.          TRANSMIT WritePropertyMultiple-Request,
                'Object Identifier' = O1,
                'Property Identifier' = P1,
                'Property Value' = ($DT_1$ updated with the special value SV)
2.          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)
    }
```

### 9.23.2.22 BACnetDateRange Non-Pattern Properties Test using WritePropertyMultiple Service

Reason for Change: Update Test Concept to include meaning of O1.

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: *O1 is the object being tested.* 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:

```
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 {
1.      TRANSMIT WritePropertyMultiple-Request,
            'Object Identifier' = O1,
            'Property Identifier' = P1,
            'Property Value' = (DR1 with startDate updated with special value SV)
2.      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)
3.      TRANSMIT WritePropertyMultiple-Request,
            'Object Identifier' = O1,
            'Property Identifier' = P1,
            'Property Value' = (DR1 with endDate updated with special value SV)
4.      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 Tests

### 9.24.2 Negative DeviceCommunicationControl Service Execution Tests

#### 9.24.2.1 Invalid Password

Reason for Change: Added Time Duration to step 1 as finite time duration is required and infinite time duration is optional. Removed step 3 as not required for this test.

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 Concept: With the IUT and TD communicating, transmit a DeviceCommunicationControl service using DISABLE_INITIATION and an invalid password. Verify the IUT responds with the correct error class and code.*

Test Steps:

```
1.  TRANSMIT DeviceCommunicationControl-Request,
        'Time Duration'   = (a value T >= 1, in minutes)
        'Enable/Disable' = DISABLE_INITIATION,
        '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)~~
```

#### 9.24.2.2 Missing Password

Reason for Change: Added Time Duration to step 1 as finite time duration is required and infinite time duration is optional. Removed step 3 as not required for this test.

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 Concept: With the IUT and TD communicating, transmit a DeviceCommunicationControl service using DISABLE_INITIATION and no password. Verify the IUT responds with one of the valid error classes and codes.*

Test Steps:

```
1.  TRANSMIT DeviceCommunicationControl-Request,
        'Time Duration'   = (a value T >= 1, in minutes)
        'Enable/Disable' = DISABLE_INITIATION
2.  IF (Protocol_Revision >= 7) THEN
3.      RECEIVE BACnet-Error-PDU,
            Error Class = SECURITY,
            Error Code = PASSWORD_FAILURE
    ELSE
4.      (RECEIVE BACnet-Error-PDU,
            Error Class = SECURITY,
            Error Code = PASSWORD_FAILURE)
5.      | (RECEIVE BACnet-Error-PDU,
            Error Class = SERVICES,
            Error Code = MISSING_REQUIRED_PARAMETER)
~~3.  VERIFY (Device, X), System_Status = (any valid value)~~
```

### 9.X35 Who-Am-I Service Execution Tests

### 9.X35.1 Uses Who-Is to Configure Devices Supporting the Who-Am-I Service

Reason for Change: No test exists for this functionality.

Purpose: To verify that the IUT can configure a device when it receives a Who-Am-I in response to a Who-Is that it sent.

Test Concept: The IUT sends a Who-Is and TD responds with a Who-Am-I. The IUT then configures the TD's Device object instance number (X) using the You-Are service.

Configuration Requirements: TD has a MAC address but is not configured with a Device object instance number.

Notes to Tester: The IUT may require the tester to specify a Device object instance number to assign TD, using the IUT's software. If the IUT requires this step to occur at the beginning of step 3, then the amount of time it takes the tester to enter this should not be counted towards the fail timer. The destination address used by TD shall be selected such that the IUT will receive the messages.

Test Steps:

```
1.  MAKE (the IUT transmit a Who-Is request to discover devices needing configuration)
2.  RECEIVE
        DESTINATION = TD | LOCAL BROADCAST | GLOBAL BROADCAST | REMOTE BROADCAST,
        Who-Is Request,
        'Device Instance Range Low Limit' = 4194303,
        'Device Instance Range High Limit' = 4194303
3.  TRANSMIT
        DESTINATION = IUT | LOCAL BROADCAST | GLOBAL BROADCAST | REMOTE BROADCAST,
        Who-Am-I Request,
        'Vendor Identifier' = (TD's Vendor_Identifier),
        'Model Name' = (any valid Model Name),
        'Serial Number' = (any valid Serial Number)
4.  BEFORE **Internal Processing Fail Time**
5.      RECEIVE
            DESTINATION = TD | GLOBAL BROADCAST | LOCAL BROADCAST | REMOTE BROADCAST,
            You-Are Request,
            'Vendor Identifier' = (TD's Vendor_Identifier),
            'Model Name' = (the Model Name sent by TD in the previous step),
            'Serial Number' = (the Serial Number sent by TD in the previous step),
            'Device Identifier' = (Device, X),
            'Device MAC Address' = (TD's MAC address, or absent)
6.  IF (TD is not an MS/TP subordinate node)
7.      TRANSMIT
            DESTINATION = IUT | LOCAL BROADCAST | GLOBAL BROADCAST | REMOTE BROADCAST,
            I-Am Request,
            'Device Identifier' = (Device, X),
            'Max APDU Length Accepted' = (any valid value),
            'Segmentation Supported' = (any valid value),
            'Vendor Identifier' = (TD's Vendor_Identifier)
```

### 9.X35.2 Configures Other Device's Object Instance Number

Reason for Change: No test exists for this functionality.

Purpose: To verify the IUT can configure another Device's object instance number using the You-Are service.

Test Concept: TD sends a Who-Am-I, and the IUT configures it with an appropriate Device object instance number (X).

Configuration Requirements: TD is configured with a MAC address but no Device object instance number. The IUT is not actively discovering devices.

Notes to Tester: The IUT may require the tester to specify a Device object instance number to assign TD, using the IUT's software. If the IUT requires this step to occur between it receiving the Who-Am-I request and sending a You-Are request, then the amount of time it takes the tester to enter this should not be counted towards the fail timer. The destination address used by TD shall be selected such that the IUT will receive the messages.

Test Steps:

```
1.  TRANSMIT
        DESTINATION = IUT | LOCAL BROADCAST | GLOBAL BROADCAST | REMOTE BROADCAST,
        Who-Am-I Request,
        'Vendor Identifier' = (TD's Vendor_Identifier),
        'Model Name' = (any valid Model Name),
        'Serial Number' = (any valid Serial Number)
2.  BEFORE Internal Processing Fail Time
3.      RECEIVE
            DESTINATION = TD | LOCAL BROADCAST | GLOBAL BROADCAST | REMOTE BROADCAST,
            You-Are Request,
            'Vendor Identifier' = (TD's Vendor_Identifier),
            'Model Name' = (the Model Name sent by TD in the previous step),
            'Serial Number' = (the Serial Number sent by TD in the previous step),
            'Device Identifier' = (Device, X),
            'Device MAC Address' = (TD's MAC address, or absent)
4.  IF (TD is not an MS/TP subordinate node)
5.      TRANSMIT
            DESTINATION = IUT | LOCAL BROADCAST | GLOBAL BROADCAST | REMOTE BROADCAST,
            I-Am Request,
            'Device Identifier' = (Device, X)
            'Max APDU Length Accepted' = (any valid value),
            'Segmentation Supported' = (any valid value),
            'Vendor Identifier' = (TD's Vendor_Identifier)
```

### 9.X36 You-Are Service Execution Tests

### 9.X36.1 Positive Tests

### 9.X36.1.1 Configurable Device Object Instance Number

Reason for Change: No test exists for this functionality.

Purpose: To verify the IUT will send a Who-Am-I while in the unconfigured state.

Test Concept: The IUT is made to send a Who-Am-I while unconfigured. TD configures the IUT's Device object instance number (X).

Configuration Requirements: The IUT is not configured with a Device object instance number. The IUT is configured with a MAC address. If the IUT cannot be configured in this way, this test shall be skipped.

Notes to Tester: The destination address used by TD shall be selected such that the IUT will receive the messages.

Test Steps:

```
1.  MAKE (the IUT send a Who-Am-I)
2.  RECEIVE
        DESTINATION = TD | LOCAL BROADCAST | GLOBAL BROADCAST | REMOTE BROADCAST,
        Who-Am-I Request,
        'Vendor Identifier' = (the IUT's Vendor_Identifier),
        'Model Name' = (the IUT's Model_Name),
        'Serial Number' = (the IUT's Serial_Number)
3.  TRANSMIT
        DESTINATION = IUT | LOCAL BROADCAST | GLOBAL BROADCAST | REMOTE BROADCAST,
        You-Are Request,
        'Vendor Identifier' = (the IUT's Vendor_Identifier),
        'Model Name' = (the IUT's Model_Name),
        'Serial Number' = (the IUT's Serial_Number),
        'Device Identifier' = (Device, X),
        'Device MAC Address' = (IUT's MAC address, or absent)
4.  IF (the IUT is not an MS/TP subordinate node)
5.      BEFORE **Unconfirmed Response Fail Time**
6.          RECEIVE
                DESTINATION = TD | LOCAL BROADCAST | GLOBAL BROADCAST | REMOTE BROADCAST,
                I-Am Request,
                'Device Identifier' = (Device, X),
                'Max APDU Length Accepted' = (any valid value),
                'Segmentation Supported' = (any valid value),
                'Vendor Identifier' = (the IUT's Vendor_Identifier)
```

### 9.X36.1.2 Configurable MAC Address

Reason for Change: No test exists for this functionality.

Purpose: To verify the IUT can be configured with a MAC address using the You-Are service.

Test Concept: TD sends a You-Are request to configure TD's MAC address.

Configuration Requirements: The IUT is not configured with a MAC address. The IUT is configured with a Device object instance number (X). If the IUT cannot be configured in this way, this test shall be skipped.

Notes to Tester: The destination address used by TD shall be selected such that the IUT will receive the messages.

Test Steps:

```
1.  TRANSMIT
```

```
        DESTINATION = LOCAL BROADCAST | GLOBAL BROADCAST | REMOTE
BROADCAST,
        You-Are Request,
        'Vendor Identifier' = (the IUT's Vendor_Identifier),
        'Model Name' = (the IUT's Model_Name),
        'Serial Number' = (the IUT's Serial_Number),
        'Device Identifier' = (absent),
        'Device MAC Address' = (a valid MAC address)
2.  IF (the IUT is not an MS/TP subordinate node)
3.      BEFORE **Internal Processing Fail Time**
4.          RECEIVE
                DESTINATION = TD | LOCAL BROADCAST | GLOBAL BROADCAST | REMOTE
BROADCAST,
                I-Am Request,
                'Device Identifier' = (Device, X),
                'Max APDU Length Accepted' = (any valid value),
                'Segmentation Supported' = (any valid value),
                'Vendor Identifier' = (the IUT's Vendor_Identifier)
```

### 9.X36.1.3 Configurable Device Object Instance Number and MAC Address

Reason for Change: No test exists for this functionality.

Purpose: To verify the IUT can be configured with both a Device object instance number and MAC address using the You-Are service.

Test Concept: TD sends a You-Are request to configure the device with both a Device object instance number (X) and a MAC address.

Configuration Requirements: The IUT is not configured with a Device object instance number or a MAC address. If the IUT cannot be configured in this way, this test shall be skipped.

Notes to Tester: The destination address used by TD shall be selected such that the IUT will receive the messages.

Test Steps:

```
1.  TRANSMIT
        DESTINATION = LOCAL BROADCAST | GLOBAL BROADCAST | REMOTE
BROADCAST,
        You-Are Request,
        'Vendor Identifier' = (the IUT's Vendor_Identifier),
        'Model Name' = (the IUT's Model_Name),
        'Serial Number' = (the IUT's Serial_Number),
        'Device Identifier' = (Device, X),
        'Device MAC Address' = (any valid MAC address)
2.  IF (the IUT is not an MS/TP subordinate node)
3.      BEFORE **Internal Processing Fail Time**
4.          RECEIVE
                DESTINATION = TD | LOCAL BROADCAST | GLOBAL BROADCAST | REMOTE
BROADCAST,
                I-Am Request,
                'Device Identifier' = (Device, X),
                'Max APDU Length Accepted' = (any valid value),
                'Segmentation Supported' = (any valid value),
                'Vendor Identifier' = (the IUT's Vendor_Identifier)
```

### 9.X36.1.4 Device Object Instance Number is Configurable Even When MAC Address is Also Sent

Reason for Change: No test exists for this functionality.

Purpose: To verify the IUT can be configured with a Device object instance number even when a MAC address that does not match its own is also sent.

Test Concept: The IUT sends a Who-Am-I while unconfigured. TD assigns the IUT a Device object instance number (X) and also sends a MAC address (M1), which is different than the IUT's MAC address. The IUT will accept both values, but not change its MAC address to M1.

Configuration Requirements: The IUT is not configured with a Device ID. The IUT is configured with a MAC address. If the IUT cannot be configured in this way, this test shall be skipped. If the IUT cannot be configured with a MAC address that cannot be changed using You-Are, this test shall be skipped.

Notes to Tester: The destination address used by TD shall be selected such that the IUT will receive the messages.

Test Steps:

```
1.  MAKE (The IUT send a Who-Am-I)
2.  RECEIVE
        SOURCE = IUT
        DESTINATION = TD | LOCAL BROADCAST | GLOBAL BROADCAST | REMOTE BROADCAST,
        Who-Am-I Request,
        'Vendor Identifier' = (the IUT's Vendor_Identifier),
        'Model Name' = (the IUT's Model_Name),
        'Serial Number' = (the IUT's Serial_Number)
3.  TRANSMIT
        DESTINATION = IUT | LOCAL BROADCAST | GLOBAL BROADCAST | REMOTE BROADCAST,
        You-Are Request,
        'Vendor Identifier' = (the IUT's Vendor_Identifier),
        'Model Name' = (the IUT's Model_Name),
        'Serial Number' = (the IUT's Serial_Number),
        'Device Identifier' = (Device, X),
        'Device MAC Address' = (M1: a MAC address different than that of the IUT)
4.  IF (the IUT is not an MS/TP subordinate node)
5.      BEFORE **Unconfirmed Response Fail Time**
6.          RECEIVE
                SOURCE = IUT        -- ensure that the MAC address has not changed
                DESTINATION = TD | LOCAL BROADCAST | GLOBAL BROADCAST | REMOTE BROADCAST,
                I-Am Request,
                'Device Identifier' = (Device, X),
                'Max APDU Length Accepted' = (any valid value),
                'Segmentation Supported' = (any valid value),
                'Vendor Identifier' = (the IUT's Vendor_Identifier)
7.  CHECK (the IUT's MAC address did not change)
```

### 9.X36.1.5 MAC Address is Reconfigurable

Reason for Change: No test exists for this functionality.

Purpose: To verify the IUT can be configured with a MAC address different than the one it was configured with using the You-Are service, without having to be unconfigured first.

Test Concept: The IUT is configured and does not need a MAC address configured to it. TD assigns a different MAC address to the IUT using the You-Are service. The IUT changes its MAC address.

Configuration Requirements: The IUT is configured with a MAC address. If the IUT cannot be configured with a MAC address that can be changed using You-Are, this test shall be skipped.

Notes to Tester: The destination address used by TD shall be selected such that the IUT will receive the messages.

Test Steps:

```
1.  VERIFY (Device, IUT), Object_Type = DEVICE
2.  TRANSMIT
        DESTINATION = IUT | LOCAL BROADCAST | GLOBAL BROADCAST | REMOTE BROADCAST,
        You-Are Request,
        'Vendor Identifier' = (the IUT's Vendor_Identifier),
        'Model Name' = (the IUT's Model_Name),
        'Serial Number' = (the IUT's Serial_Number),
        'Device Identifier' = (the IUT's Device Object_Identifier),
        'Device MAC Address' = (a MAC address different than that of the IUT)
3.  IF (the IUT is not an MS/TP subordinate node)
4.      BEFORE Unconfirmed Response Fail Time
5.          RECEIVE
                SOURCE = IUT        -- ensure that the MAC address is what was set by TD
                DESTINATION = TD | LOCAL BROADCAST | GLOBAL BROADCAST | REMOTE BROADCAST,
                I-Am Request,
                'Device Identifier' = (the IUT's Device Object_Identifier),
                'Max APDU Length Accepted' = (any valid value),
                'Segmentation Supported' = (any valid value),
                'Vendor Identifier' = (the IUT's Vendor_Identifier)
6.  TRANSMIT
        DESTINATION = IUT -- using the newly configured MAC address
        ReadProperty-Request,
        'Object Identifier' = (Device, IUT),
        'Property Identifier' = Object_Type
7.  RECEIVE
        SOURCE = IUT                -- ensure that the MAC address is what was set by TD
        ReadProperty-Ack,
        'Object Identifier' = (Device, IUT),
        'Property Identifier' = Object_Type,
        'Property Value' = DEVICE
```

### 9.X36.1.6 Retains Configuration Through Restarts

Reason for Change: No test exists for this functionality.

Purpose: To verify the IUT will retain a configured Device object instance number and MAC address across a restart.

Test Concept: The IUT is configured with a Device object instance number (X) and MAC address (M1) using You-Are. The IUT is then restarted and the Device object instance number and MAC address are verified to be the same that were configured at the beginning of the test.

Configuration Requirements: The IUT is unconfigured.

Notes to Tester: If the IUT's MAC address cannot be changed using You-Are, use the IUT's MAC address in place of M1. The destination address used by TD shall be selected such that the IUT will receive the messages.

Test Steps:

```
1.  TRANSMIT
        DESTINATION = IUT | LOCAL BROADCAST | GLOBAL BROADCAST | REMOTE
BROADCAST,
        You-Are Request,
        'Vendor Identifier' = (the IUT's Vendor_Identifier),
        'Model Name' = (the IUT's Model Name),
        'Serial Number' = (the IUT's Serial Number),
        'Device Identifier' = (Device, X),
        'Device MAC Address' = (M1, or absent if the IUT does not support a configurable MAC address)
2.  RECEIVE
        DESTINATION = TD | LOCAL BROADCAST | GLOBAL BROADCAST | REMOTE
BROADCAST,
        I-Am Request,
        'Device Identifier' = (Device, X),
        'Max APDU Length Accepted' = (any valid value),
        'Segmentation Supported' = (any valid value),
        'Vendor Identifier' = (the IUT's Vendor_Identifier)
3.  MAKE (the IUT restart)
4.  WAIT (for the IUT to restart)
5.  TRANSMIT
        DESTINATION = IUT | LOCAL BROADCAST | GLOBAL BROADCAST | REMOTE
BROADCAST,
        Who-Is Request,
        'Device Instance Range Low Limit' = 4194303,
        'Device Instance Range High Limit' = 4194303
6.  WAIT Internal Processing Fail Time
7.  CHECK (the IUT did not send a Who-Am-I Request)
8.  TRANSMIT
        DESTINATION = IUT | LOCAL BROADCAST | GLOBAL BROADCAST | REMOTE
BROADCAST,
        Who-Is Request,
9.  RECEIVE
        DESTINATION = TD | LOCAL BROADCAST | GLOBAL BROADCAST | REMOTE
BROADCAST,
        I-Am Request,
        'Device Identifier' = (Device, X),
        'Max APDU Length Accepted' = (any valid value),
        'Segmentation Supported' = (any valid value),
        'Vendor Identifier' = (the IUT's Vendor_Identifier)
10. CHECK (the IUT's MAC address is M1 if it was changeable)
```

### 9.X36.1.7 Unconfigurable by You-Are

Reason for Change: No test exists for this functionality.

Purpose: To verify the IUT can be unconfigured with the You-Are service.

Test Concept: The IUT is configured with a Device object instance number and MAC address. TD then sends a You-Are request with 'Device Identifier' set to 4149303, and the IUT becomes unconfigured.

Configuration Requirements: The IUT is configured with both a Device object instance number and MAC address.

Notes to Tester: The destination address used by TD shall be selected such that the IUT will receive the messages.

Test Steps:

```
1.  TRANSMIT
        DESTINATION = IUT | LOCAL BROADCAST | GLOBAL BROADCAST | REMOTE BROADCAST,
        You-Are Request,
        'Vendor Identifier' = (the IUT's Vendor_Identifier),
        'Model Name' = (the IUT's Model Name),
        'Serial Number' = (the IUT's Serial Number),
        'Device Identifier' = (Device, 4194303)
        'MAC Address' = (the IUT's MAC address, or absent)
2.  CHECK (the IUT did not send any requests, except for an optional Who-Am-I if the MAC address cannot be unconfigured)
```

**9.X36.2 Negative Tests**

**9.X36.2.1 Only Supports Execution of the You-Are Service While Unconfigured**

Reason for Change: No test exists for this functionality.

Purpose: To verify the IUT does not initiate or execute any other services while the MAC address is unconfigured.

Test Concept: The IUT's MAC address is unconfigured and the tester attempts to attempts to execute any services it would otherwise execute.

Configuration Requirements: The IUT is configured with a MAC address. If the IUT's MAC address cannot be unconfigured, then this test shall be skipped.

Notes to Tester: The destination address used by TD shall be selected such that the IUT will receive the messages.

Test Steps:

```
1.  TRANSMIT
        DESTINATION = IUT | LOCAL BROADCAST | GLOBAL BROADCAST | REMOTE BROADCAST,
        You-Are-Request,
        'Vendor Identifier' = (the IUT's Vendor_Identifier),
        'Model Name' = (the IUT's Model_Name),
        'Serial Number' = (the IUT's Serial_Number),
        'Device Identifier' = (Device, 4194303)
        'MAC Address' = (the IUT's MAC address, or absent)
2.  WAIT Internal Processing Fail Time
3.  CHECK (the IUT did not transmit any requests)
3.  TRANSMIT
        DESTINATION = LOCAL BROADCAST | GLOBAL BROADCAST | REMOTE BROADCAST,
        Read-Property-Request,
        'Object Identifier' = (any object that would exist in the IUT, or (Device, 4194303)),
        'Property Identifier' = (any property in the object selected)
```

```
4.  WAIT Internal Processing Fail Time
5.  CHECK (the IUT did not respond with a Read-Property-ACK)
```

### 9.X36.2.2 Only Accepts Configuration When Received Parameters Match

Reason for Change: No test exists for this functionality.

Purpose: To verify the IUT will not configure or reconfigure itself when the parameters in a You-Are request do not match its vendor identifier, model name, and serial number.

Test Concept: The IUT is unconfigured and is sent a You-Are but with the wrong Vendor Identifier, Model Name, and Serial Number. The IUT does not accept the configuration and does not transmit an I-Am request indicating it has been configured.

Configuration Requirements: The IUT needs configuration of either Device object instance number or MAC address, or both. This test shall be skipped if the IUT is an MS/TP subordinate node.

Notes to Tester: If the IUT only supports configuration of either Device object instance number or MAC address but not both, TD shall use the IUT's Device Identifier or MAC address, whichever is configured, when sending You-Are requests. The destination address used by TD shall be selected such that the IUT will receive the messages.

Test Steps:

```
1.  TRANSMIT
        DESTINATION = IUT | LOCAL BROADCAST | GLOBAL BROADCAST | REMOTE BROADCAST,
        You-Are-Request,
        'Vendor Identifier' = (the IUT's Vendor_Identifier),
        'Model Name' = (the IUT's Model_Name),
        'Serial Number' = (any value other than the IUT's Serial_Number),
        'Device Identifier' = (any valid Device Identifier),
        'Device MAC Address' = (any valid MAC address, or absent)
2.  WAIT Unconfirmed Response Fail Time
3.  CHECK (the IUT did not transmit an I-Am-Request)
4.  TRANSMIT
        DESTINATION = IUT | LOCAL BROADCAST | GLOBAL BROADCAST | REMOTE BROADCAST,
        You-Are Request,
        'Vendor Identifier' = (the IUT's Vendor_Identifier),
        'Model Name' = (any value other than the IUT's Model_Name),
        'Serial Number' = (IUT's Serial_Number),
        'Device Identifier' = (any valid Device Identifier),
        'Device MAC Address' = (any valid MAC address, or absent)
5.  WAIT Unconfirmed Response Fail Time
6.  CHECK (the IUT did not transmit an I-Am-Request)
7.  TRANSMIT
        DESTINATION = IUT | LOCAL BROADCAST | GLOBAL BROADCAST | REMOTE BROADCAST,
        You-Are Request,
        'Vendor Identifier' = (any value other than the IUT's Vendor_Identifier),
        'Model Name' = (the IUT's Model_Name),
        'Serial Number' = (the IUT's Serial_Number),
        'Device Identifier' = (any valid Device Identifier),
        'Device MAC Address' = (any valid MAC address, or absent)
8.  WAIT Unconfirmed Response Fail Time
9.  CHECK (the IUT did not transmit an I-Am-Request)
```

### 135.1-2023*u*-5 Add new and correct existing tests for the Network Layer

**Rationale**

Errors have been identified in a number of network layer tests in ANSI/ASHRAE Standard 135.1-2023.
In addition, test coverage is increased with the addition of new tests.

## 10. NETWORK LAYER PROTOCOL TESTS

### 10.2 Router Functionality Tests

### 10.2.2 Processing Network Layer Messages

### 10.2.2.4 Router-Busy-To-Network

#### 10.2.2.4.2 Forwarding Router-Busy-To-Network Information for all DNETs

Reason for Change: Add new Notes to Tester.

Purpose: To verify that the IUT correctly forwards information indicating that all DNETs reachable through a particular router are temporarily unreachable because of traffic congestion.

*Notes to Tester: This test is to be run after test 10.2.2.2.5 such that R2-3 is known to be a router to network 6.*

Test Steps:

```
1.  TRANSMIT PORT B,
            DESTINATION = LOCAL BROADCAST,
            SOURCE = R2-3,
            Router-Busy-To-Network
2.  RECEIVE PORT A,
            DESTINATION = LOCAL BROADCAST,
            SOURCE = IUT,
            Router-Busy-To-Network,
            Network Numbers = 3,6 | 6,3 | (absent)
```

#### 10.2.2.4.4 Timeout

Reason for Change: Correct tests to specify correct SINFO information.

Purpose: To verify that the IUT restores the availability status of DNETs after the busy timer expires.

Test Steps:

```
1.  TRANSMIT PORT B,
        DESTINATION = LOCAL BROADCAST,
        SOURCE = R2-3,
        Router-Busy-To-Network,
        Network Numbers = 3
2.  RECEIVE PORT A,
        DESTINATION = LOCAL BROADCAST,
        SOURCE = IUT,
        Router-Busy-To-Network,
        Network Numbers = 3
```

```
3.  WAIT (30 seconds)
4.  TRANSMIT PORT A,
        DA = IUT,
        SOURCE = D1A,
        DNET = 3,
        DADR = D3D,
        Hop Count = 255,
        ReadProperty-Request,
        'Object Identifier' =    (any BACnet standard object),
        'Property Identifier' =  (any required property of the specified object)
5.  RECEIVE PORT B,
        DA = R2-3,
        SOURCE = IUT,
        SNET = 1,
        SADR = D1A,
        DNET = 3,
        DADR = D3D,
        Hop Count = (any integer x: 0 < x < 255),
        ReadProperty-Request,
        'Object Identifier' =    (the object identifier used in step 4),
        'Property Identifier' =  (the property identifier used in step 4)
```

### 10.2.2.5 Execute Router-Available-To-Network

### 10.2.2.5.1 Restoring Specific DNETs

Reason for Change: Correct test to specify correct SINFO information. Add new Notes to Tester.

Purpose: To verify that the IUT updates its network availability information when a Router-Available-To-Network message conveying specific DNETs is received.

*Notes to Tester: This test is to be run after test 10.2.2.2.5 such that R2-3 is known to be a router to network 6.*

Test Steps:

```
1.  TRANSMIT PORT B,
        DESTINATION = LOCAL BROADCAST,
        SOURCE = R2-3,
        Router-Busy-To-Network
2.  RECEIVE PORT A,
        DESTINATION = LOCAL BROADCAST,
        SOURCE = IUT,
        Router-Busy-To-Network,
        Network Numbers = 3, 6 | 6, 3
3.  TRANSMIT PORT B,
        DESTINATION = LOCAL BROADCAST,
        SOURCE = R2-3,
        Router-Available-To-Network,
        Network Numbers = 3
4.  RECEIVE PORT A,
        DESTINATION = LOCAL BROADCAST,
        SOURCE = IUT,
        Router-Available-To-Network,
        Network Numbers = 3
5.  TRANSMIT PORT A,
        DESTINATION = IUT,
        SOURCE = D1A,
        DNET = 3,
        DADR = D3D,
        Hop Count = 255,
        ReadProperty-Request,
        'Object Identifier' =    (any BACnet standard object),
        'Property Identifier' =  (any required property of the specified object)
6.  RECEIVE PORT B,
        DESTINATION = R2-3,
        SOURCE = IUT,
        SNET = 1,
        SADR = D1A,
        DNET = 3,
        DADR = D3D,
        Hop Count = (any integer x: 0 < x < 255),
        ReadProperty-Request,
        'Object Identifier' =    (the object identifier used in step 5),
        'Property Identifier' =  (the property identifier used in step 5)
7.  TRANSMIT PORT A,
        DESTINATION = IUT,
        SOURCE = D1A,
        DNET = 6,
        DADR = (any valid device address),
        Hop Count = 255,
        ReadProperty-Request,
        'Object Identifier' =    (any BACnet standard object),
        'Property Identifier' =  (any required property of the specified object)
8.  RECEIVE PORT A,
        DESTINATION = D1A,
        SOURCE = IUT,
        Reject-Message-To-Network,
        Reject Reason = 2 (router busy),
        DNET = 6
```

### 10.2.2.5.2 Restoring All DNETs

Reason for Change: Correct test to specify correct SINFO information. Add new Notes to Tester.

Purpose: To verify that the IUT updates its network availability information when a Router-Available-To-Network message conveying no DNETs is received.

*Notes to Tester: This test is to be run after test 10.2.2.2.5 such that R2-3 is known to be a router to network 6.*

Test Steps:

```
1.  TRANSMIT PORT B,
        DESTINATION = LOCAL BROADCAST,
        SOURCE = R2-3,
        Router-Busy-To-Network
2.  RECEIVE PORT A,
        DESTINATION = LOCAL BROADCAST,
        SOURCE = IUT,
        Router-Busy-To-Network,
        Network Numbers = 3, 6 | 6, 3
```

3. TRANSMIT PORT B,
    DESTINATION = LOCAL BROADCAST,
    SOURCE = R2-3,
    Router-Available-To-Network
4. RECEIVE PORT A,
    DESTINATION = LOCAL BROADCAST,
    SOURCE = IUT,
    Router-Available-To-Network,
    Network Numbers = 3, 6 | 6, 3
5. TRANSMIT PORT A,
    DA = IUT,
    SOURCE = D1A,
    DNET = 3,
    DADR = D3D,
    Hop Count = 255,
    ReadProperty-Request,
    'Object Identifier' = (any BACnet standard object),
    'Property Identifier' = (any required property of the specified object)
6. RECEIVE PORT B,
    DA = R2-3,
    SOURCE = IUT,
    *SNET = 1,*
    *SADR = D1A,*
    DNET = 3,
    DADR = D3D,
    Hop Count = (any integer x: 0 < x < 255),
    ReadProperty-Request,
    'Object Identifier' = (the object identifier used in step 5),
    'Property Identifier' = (the property identifier used in step 5)
7. TRANSMIT PORT A,
    DA = IUT,
    SOURCE = D1A,
    DNET = 6,
    DADR = (any valid device address),
    Hop Count = 255,
    ReadProperty-Request,
    'Object Identifier' = (any BACnet standard object),
    'Property Identifier' = (any required property of the specified object)
8. RECEIVE PORT B,
    DA = R2-3,
    SOURCE = IUT,
    *SNET = 1,*
    *SADR = D1A,*
    DNET = 6,
    DADR = (the address used in step 6),
    Hop Count = (any integer x: 0 < x < 255),
    ReadProperty-Request,
    'Object Identifier' = (the object identifier used in step 7),
    'Property Identifier' = (the property identifier used in step 7)

### 10.2.3 Routing of Unicast APDUs

### 10.2.3.6 Attempt to Locate Downstream Routers

### 10.2.3.6.2 Successful Attempt to Locate Router

Reason for Change: Correct test to specify correct DA information.

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.

TOmin: 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 TOmin

*4*. TRANSMIT PORT B,
    DESTINATION = LOCAL BROADCAST, SOURCE = R2-3, I-Am-Router-To-Network,
    Network Numbers = 3

*5*. RECEIVE PORT B,
    ~~SA~~ *DA* = 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)

### 10.2.6 Network Layer Priority

Reason for Change: Correct test to specify correct DINFO information.

Purpose: To verify that the IUT can process and forward messages with all network priorities.

Test Steps:

1. TRANSMIT PORT A,
   DA = IUT,
   SA = D1A,
   Priority = B'00',
   DNET = 2,
   DADR = D2C,
   Hop Count = 255,
   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,
   ~~SA~~*DA* = D2C,
   SA = IUT,
   Priority = B'00',
   SNET = 1,
   SDR = D1A,
   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)
3. TRANSMIT PORT A,
   DA = IUT,
   SA = D1A,
   Priority = B'01',
   DNET = 2,
   DADR = D2C,
   Hop Count = 255,
   BACnet-Confirmed-Request-PDU,
   'Service Choice' = ReadProperty-Request,
   'Object Identifier' = (any object identifier),
   'Property Identifier' = (any property of the specified object)
4. RECEIVE PORT B,
   DA = D2C,
   SA = IUT,
   Priority = B'01',
   SNET = 1,
   SDR = D1A,
   BACnet-Confirmed-Request-PDU,
   'Service Choice' = ReadProperty-Request,
   'Object Identifier' = (the object identifier used in step 3),
   'Property Identifier' = (the property identifier used in step 3)
5. TRANSMIT PORT A,
   DA = IUT,
   SA = D1A,
   Priority = B'10',
   DNET = 2,
   DADR = D2C,
   Hop Count = 255,

```
          BACnet-Confirmed-Request-PDU,
          'Service Choice' =       ReadProperty-Request,
          'Object Identifier' =    (any object identifier),
          'Property Identifier' =  (any property of the specified object)
6.   RECEIVE PORT B,
          DA = D2C,
          SA = IUT,
          Priority = B'10',
          SNET = 1,
          SDR = D1A,
          BACnet-Confirmed-Request-PDU,
          'Service Choice' =       ReadProperty-Request,
          'Object Identifier' =    (the object identifier used in step 5),
          'Property Identifier' =  (the property identifier used in step 5)
7.   TRANSMIT PORT A,
          DA = IUT,
          SA = D1A,
          Priority = B'11',
          DNET = 2,
          DADR = D2C,
          Hop Count = 255,
          BACnet-Confirmed-Request-PDU,
          'Service Choice' =       ReadProperty-Request,
          'Object Identifier' =    (any object identifier),
          'Property Identifier' =  (any property of the specified object)
8.   RECEIVE PORT B,
          DA = D2C,
          SA = IUT,
          Priority = B'11',
          SNET = 1,
          SDR = D1A,
          BACnet-Confirmed-Request-PDU,
          'Service Choice' =       ReadProperty-Request,
          'Object Identifier' =    (the object identifier used in step 7),
          'Property Identifier' =  (the property identifier used in step 7)
```

### 10.2.X7 Must Understand for Unknown Data Options Forwarding Test

Reason for Change: No test exists for this functionality.

Purpose: To verify that routers which connects multiple network supporting data attributes, correctly routes data attributes that include unknown data Options and a valid ‘Secure Path’ Data Option.

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 unknown data option data attributes in addition to a valid ‘Secure Path’ Data Option. 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.

Test Steps:

```
1.   TRANSMIT PORT A,
          DA = LOCAL BROADCAST,
          SOURCE = D1A,
          'Data Options' = ({X'C1' }, -- (more, M.U., no len, opt_type = 1 (Secure Path))
                                { X'5E' }), -- (not more, M.U., no len, opt_type = 30 (unknown header option type)),
          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' = ({ X'C1' }, -- (more, M.U., no len, opt_type = 1 (Secure Path))
                                { X'5E' }), -- (not more, M.U., no len, opt_type = 30 (unknown header option type)),
          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
3.  TRANSMIT PORT A,
          DA = LOCAL BROADCAST,
          SOURCE = D1A,
          'Data Options' = ({ X'C1' }, -- (more, M.U., no len, opt_type = 1 (Secure Path))
                                { X'1E' }), -- (not more, no M.U., no len, opt_type = 30 (unknown header option type)),
          DNET = GLOBAL BROADCAST,
          DLEN = 0,
          Hop Count = 255,
          BACnet-Unconfirmed-Request-PDU,
          'Service Choice' = Who-Is
4.  RECEIVE PORT B,
          DA = LOCAL BROADCAST,
          SA = IUT,
          'Data Options' = ({ X'C1' }, -- (more, M.U., no len, opt_type = 1 (Secure Path))
                            { X'1E' }), -- (not more,  no M.U., no len, opt_type = 30 (unknown header option type)),
          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.X8 Must Understand for Unknown Data Options Dropping Test

Reason for Change: No test exists for this functionality.

Purpose: To verify that routers correctly drop data attributes that include unknown Data Options and a valid ‘Secure Path’ Data Option.

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 that include unknown data Options and Must Understand = TRUE. Verify that message is correctly routed and the data_attributes are silently dropped. Repeat with Must Understand = FALSE and verify the message is correctly routed and the data_attributes are silently dropped.

Configuration Requirements: The IUT shall be configured as a router between 2 networks in which the destination network does not support data attributes.

Test Steps:

1. TRANSMIT PORT A,
    DA = LOCAL BROADCAST,
    SOURCE = D1A,
    'Data Options' = ({ X'C1' }, -- (more, M.U., no len, opt_type = 1 (Secure Path))
        { X'5E' }), -- (not more, M.U., no len, opt_type = 30 (unknown header option type)),
    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
3. TRANSMIT PORT A,
    DA = LOCAL BROADCAST,
    SOURCE = D1A,
    'Data Options' = ({ X'C1' }, -- (more, M.U., no len, opt_type = 1 (Secure Path))
        { X'1E' }), -- (not more, no M.U., no len, opt_type = 30 (unknown header option type)),
    DNET = GLOBAL BROADCAST,
    DLEN = 0,
    Hop Count = 255,
    BACnet-Unconfirmed-Request-PDU,
    'Service Choice' = Who-Is
4. 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

**10.8 Virtual Routing Functionality Tests**

**10.8.3 Routing of Unicast APDUs**

**10.8.3.5 Unicast Messages That Should Not Be Routed**

**10.8.3.5.X1 Silently Drop Messages to a Virtual Device that is Offline**

Reason for Change: No test exists for this functionality. This test is not in any SSPC proposal.

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 = V1DA
        '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)
```

**135.1-2023*u*-6 Add new and correct existing Data Link Layer Tests**

**Rationale**

Errors have been identified in a number of data link layer tests in ANSI/ASHRAE Standard 135.1-2023.
In addition, test coverage is increased with the addition of new tests.

## 12. DATA LINK LAYER PROTOCOLS TESTS

### 12.1 MS/TP State Machine Tests

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

### 12.1.3.X Ignores Unsupported Frame Types

Reason for Change: No test exists for this functionality.

Purpose: To verify that the IUT will quietly ignore unknown frame types.

Test Concept: The TD sends MSTP frames to the IUT with extended frame types (32 and 33). The IUT is observed to verify that it quietly ignores the unknown frame types and does not reset.

Test Configuration: None.

Test Steps:

```
1.  VERIFY System_Status = OPERATIONAL | OPERATIONAL_READ_ONLY
2.  TRANSMIT (any BACnet service choice, NPDU > 501 octets)
        Frame Type = 32 -- DER frame
3.  CHECK (verify that the IUT does not send a frame in response and does not reset)
4.  VERIFY System_Status = OPERATIONAL | OPERATIONAL_READ_ONLY
5.  TRANSMIT (any BACnet service choice, NPDU > 501 octets)
        Frame Type = 33 -- DNER frame
6.  CHECK (verify that the IUT does not send a frame in response and does not reset)
7.  VERIFY System_Status = OPERATIONAL | OPERATIONAL_READ_ONLY
```

### 12.1.3.X1 Verify MSTP Response Queue Test

Reason for Change: No test for this functionality.

Purpose: To verify the IUT can correctly respond to back-to-back requests where the second request contains Data_Expecting_Reply equal to TRUE.

Test Concept: Set the TD's Max_Info_Frames to a value greater than 1. Without passing the Token, the TD sends a Who-Is request followed immediately by a ReadProperty-Request. The IUT can either respond with a valid ReadProperty-ACK or a Reply Postponed.

Configuration Requirements: None.

Test Steps:

```
1.  TRANSMIT
      Who-Is-Request
2.  TRANSMIT
      ReadProperty-Request,
        'Frame Type' = BACnet Data Expecting Reply,
        'Object Identifier' = (Device, X),
        'Property Identifier' = Object_Identifier
3.  RECEIVE
      ReadProperty-ACK,
        'Frame Type' = BACnet Data Not Expecting Reply,
        'Object Identifier' = (Device, X),
        'Property Identifier' = Object_Identifier,
        'Property Value' = (Device, X) |
4.  RECEIVE
      Reply Postponed
```

**12.1.X MS/TP Router Reply Postponed Test**

Reason for Change: There are no tests for this functionality. Checking the hop count is unnecessary in step 3.

Purpose: To verify that the IUT sends 'reply postponed' when it receives a MS/TP packet with a data_expecting_reply parameter set to TRUE destined for another network.

Test Concept: The IUT is configured to route between Network 1 on port A (MS/TP network) and Network 2 on port B (any BACnet data link). D1A resides on Network 1 and D2C resides on Network 2. The IUT receives a message from D1A destined for D2C with the data expecting reply parameter set to TRUE. The IUT transmits ‘Reply Postponed’ to D1A before attempting to route the message.

Configuration Requirements: The IUT is actively routing between Network 1 and Network 2 and D1A has discovered D2C.

Test Steps:

```
1.  TRANSMIT PORT A
        SA = D1A
        DA = IUT
        DNET = Network 2
        DADR = D2C
        Hop Count = 255
        BACnet-Confirmed-Request-PDU
2.  RECEIVE PORT A
        Reply Postponed
3.  RECEIVE PORT B
        SA = IUT
        DA = D2C
        SNET = Network 1
        SADR = D1A
        BACnet-Confirmed-Request-PDU
```

**12.3 BACnet/IP Functionality Tests**

**12.3.8 Foreign Device Tests**

**12.3.8.3 Recurring Register-Foreign-Device Test**

Reason for Change: Purpose does not match with intention of 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.~~
*Purpose: Verify that the Register-Foreign-Device is re-sent before 'Time-to-Live' is expired.*

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)
5.          RECEIVE DA = BBMD1,
                Register-Foreign-Device
6.  TRANSMIT BVLC-Result,
        'Result Code' = Successful completion
7.  BEFORE (the time configured for the 'Time-to-Live' parameter used for Register-Foreign-Device
requests)
8.          RECEIVE DA = BBMD1,
                Register-Foreign-Device
9.  TRANSMIT BVLC-Result,
        'Result Code' = Successful completion
```

**12.3.8.X1 Register-Foreign-Device when NPOs Supported**

Reason for Change: No test for this configuration.

Purpose: To validate whether the Network Port Object can configure the dispatch and parameters of Register-Foreign-Device requests.

Test Concept: To enable and disable Register-Foreign-Device requests use the BACnet_IP_Mode property and configure the 'BBMD Address' and 'Time-to-Live' parameters through the FD_BBMD_Address and FD_Subscription_Lifetime properties in the Network Port Object.

Configuration Requirements: BBMD1 is the TD simulating a correctly functioning BBMD implementation. The IUT's Network Port object is initially configured for BACnet/IP or BACnet/IPv6 in NORMAL mode. Mode represents BACnet_IP_Mode for BACnet/IP and BACnet_IPv6_Mode for BACnet_IPv6.

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 Mode = NORMAL

-- update Mode, BBMD address and subscription lifetime
```

```
4.  IF (Mode is writable) THEN
5.      WRITE Mode = FOREIGN
    ELSE
6.      MAKE (Mode = FOREIGN)
7.  WRITE FD_BBMD_Address = BBMD1
8.  WRITE FD_Subscription_Lifetime = T1 (arbitrary value in seconds)
9.  VERIFY Changes_Pending = TRUE
10. TRANSMIT ReinitializeDevice-Request,
        'Reinitialized State of Device' = WARMSTART | ACTIVATE_CHANGES,
         'Password' = (any valid password)
11. RECEIVE BACnet-SimpleACK-PDU
12. WAIT Activate Changes Fail Time
13. VERIFY Mode = FOREIGN
14. VERIFY FD_BBMD_Address = BBMD1
15. VERIFY FD_Subscription_Lifetime = T1

-- verify Register-Foreign-Device request in TD
16. WAIT (T1 seconds)
17. RECEIVE DA = BBMD1,
        Register-Foreign-Device,
        'Time-to-Live' = T1
18. TRANSMIT BVLC-Result,
        'Result Code' = Successful completion

-- verify that the Register-Foreign-Device requests can be disabled
19. VERIFY Changes_Pending = FALSE
20.  IF (Mode is writable) THEN
21.     WRITE Mode = NORMAL
22.     VERIFY Changes_Pending = TRUE
23.     TRANSMIT ReinitializeDevice-Request,
            'Reinitialized State of Device' = WARMSTART | ACTIVATE_CHANGES,
            'Password' = (any valid password)
24.     RECEIVE BACnet-SimpleACK-PDU
25.     WAIT Activate Changes Fail Time
    ELSE
26.     MAKE (the IUT enter NORMAL mode)
27. VERIFY Mode = NORMAL
28. WAIT (more than T1 seconds)
29. CHECK (that the IUT did not send any Register-Foreign-Device requests)
```

### 12.3.8.X2 Forwarded-NPDU (Two-hop Distribution) in Foreign Mode

Reason For Change: The specific case of processing a Forwarded-NPDU message while in foreign mode did not exist.

Purpose: To verify that an IUT, configured in foreign mode, will process a Forwarded-NPDU message from a BBMD.

Configuration Requirements: The IUT is registered as a Foreign Device with the TD. The TD is configured as a BBMD, on a different IP subnet than the IUT.

Test Steps:

```
1.  TRANSMIT DA = SA = TD,
        Forwarded-NPDU,
        Originating-Device = D1,
        NPDU = Who-Is
```

```
2.  IF (the IUT responds with Unicast I-Am) THEN
3.      RECEIVE DA = D1,
            Original-Unicast-NPDU,
            NPDU = I-Am
    ELSE
4.      RECEIVE DA = TD, SOURCE = IUT,
            Distribute-Broadcast-To-Network-NPDU,
            NPDU = I-Am
5.  CHECK (The IUT shall not issue any Forwarded-NPDUs)
```

### 12.3.11 BBMD Configuration Tests - B Side

### 12.3.11.4 Broadcast Distribution Table Configuration via Hostname Entries

Reason for Change: With the advent of Network Port objects, BBMDs now need to accept hostname BDT entries.

Purpose: Verify that the IUT accepts and resolves hostname entries in the BBMD_Broadcast_Distribution_Table *and that the resolved IP address are shown in the result of a Read-Broadcast-Distribution-Table request*.

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*, HN1 that resolves to an IP address, IP2,* ~~(at IP address IP2),~~ and an entry with a non-resolvable hostname *(HN2)*. Send a broadcast that the IUT should distribute to its peer BBMDs and verify that it sends *them* 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 *(D1)* 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,
                                IP1 (an entry with an IP address),
                                HN1 (an entry with a resolvable hostname),
                                HN2 (an entry with a non-resolvable hostname))
2.  READ BDT = BBMD_Broadcast_Distribution_Table
3.  CHECK (BDT contains IUT, IP1, HN1, HN2 in any order)
4.  TRANSMIT ReinitializeDevice-Request
    'Reinitialized State of Device' =     ACTIVATE_CHANGES
5.  WAIT Activate Changes Fail Time
6.  WAIT until the IUT completes DNS resolution
7.  TRANSMIT
        DA = Local IP Broadcast,
        SA = D1,
        Original-Broadcast-NPDU,
        NPDU = Who-Is-Request
8.  RECEIVE
        DA = IP1,
        SA = IUT,
        Forwarded-NPDU,
            Originating-Device = D1,
            NPDU = Who-Is
9.  RECEIVE
```

DA = IP2,
SA = IUT,
Forwarded-NPDU,
    Originating-Device = D1,
    NPDU = Who-Is

10. READ BDT = BBMD_Broadcast_Distribution_Table~~-- re-read the table to determine the order the IUT~~
~~-- placed the entries~~
*11. CHECK (BDT contains IUT, IP1, HN1, HN2 in any order)*

*12. TRANSMIT*
*DA = IUT,*
*SA = D1,*
*Read-Broadcast-Distribution-Table*
13. 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 BDT) read from
BBMD_Broadcast_Distribution_Table)

**12.4 BACnet/IPv6 Functionality Tests**

**12.4.4 BBMD Tests**

**12.4.4.1 Positive Tests**

**12.4.4.1.5 Distribute-Broadcast-To-Network**

Reason for Change: Incorrect addressing. Added missing Test Concept.

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

*Test Concept: Send a Distribute-Broadcast-To-Network message containing a Who-Is request to the IUT from a registered foreign device. Verify that the IUT distributes it to all associated BBMDs and registered foreign devices. Also verify that the IUT processes the Who-Is request by checking that the IUT responds with an I-Am.*

Configuration Requirements: Register FD1 ~~as a foreign device~~ *and FD2 as foreign devices* 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 forwarded messages transmitted by the IUT is not significant.

Test Steps:

1. TRANSMIT
DA = IUT,
SA = FD1,
Distribute-Broadcast-To-Network,

```
        Who-Is-Request

-- verify the broadcast is sent to the local IPv6 multicast address
2.  RECEIVE
        DA = B/IPv6 Link Local Multicast Address,
        SA = IUT,
        Forwarded-NPDU,
        Original-Source-Virtual-Address = FD1,
        Original-Source-~~Virtual~~B/IPv6-Address = FD1,
        Who-Is-Request

-- verify the broadcast is sent to the broadcast to each peer BBMD
3.  RECEIVE
        DA = BBMD1,
        SA = IUT,
        Forwarded-NPDU,
        Original-Source-Virtual-Address = FD1,
        Original-Source-~~Virtual~~B/IPv6-Address = FD1,
        Who-Is-Request
4.  RECEIVE
        DA = BBMD2,
        SA = IUT,
        Forwarded-NPDU,
        Original-Source-Virtual-Address = FD1,
        Original-Source-~~Virtual~~B/IPv6-Address = FD1,
        Who-Is-Request
5.  RECEIVE
        DA = BBMD3,
        SA = IUT,
        Forwarded-NPDU,
        Original-Source-Virtual-Address = FD1,
        Original-Source-~~Virtual~~B/IPv6-Address = FD1,
        Who-Is-Request

-- verify the broadcast is sent to all other registered foreign devices
6.  RECEIVE
        DA = FD2,
        SA = IUT,
        Forwarded-NPDU,
        Original-Source-Virtual-Address = FD1,
        Original-Source-~~Virtual~~B/IPv6-Address = FD1,
        Who-Is-Request
7.  CHECK (that the IUT does not send the Who-Is request to FD1)

-- verify that the IUT sent the Who-Is to its own application layer as well by verifying
-- it responds to the request with an I-Am
8.  RECEIVE
        DA = B/IPv6 Link Local Multicast Address,
        SA = IUT,
        Original-Broadcast-NPDU,
        Original-Source-Virtual-Address = IUT,
        DNET = 65535 or absent,
        I-Am-Request
        | (
        DA = FD1,
        SA = IUT,
```

*Original-Unicast-NPDU,*
*Source-Virtual-Address = IUT,*
*Destination-Virtual-Address = FD1,*
*I-Am-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~~

### 12.4.4.2 Negative Tests

#### 12.4.4.2.1 Ignore Forwarded-NPDU from non-Participating BBMDs

Reason for Change: Test should not use FD3 as the source of the forwarded message. Missing Test Concept.

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

*Test Concept: The IUT is configured as a BBMD and is actively forwarding messages to registered devices. Validate that the IUT does not forward a message received from a BBMD not listed in the IUT's BDT.*

Configuration Requirements: ~~Empty the IUT's BDT. FD3 is a foreign device registered with the IUT.~~*TD shall operate as BBMD4 and is not listed in the IUT's BDT. FD1 is a foreign device registered with BBMD4. The IUT is configured with at least one foreign device.*

Test Steps:

```
1.  TRANSMIT
        DA = IUT,
        SA = ~~BBMD1~~*BBMD4*,
        Forwarded-NPDU,
        Source-Virtual-Address = ~~FD3~~*FD1*,
        Original-Source-B/IPv6-Address = ~~FD3~~*FD1*
        I-Am-Request
2.  CHECK (The IUT does not ~~issue any Forwarded-NPDU BVLCs~~*forward the message to any foreign device*)
```

**12.5 Secure Connect Functionality Tests**

**12.5.1 Basic Node Tests**

**12.5.1.1 Basic Node Positive Tests**

**12.5.1.1.16 Heartbeat-Request Initiation Test**

Reason for Change: modified per Addendum 135-2020cc-1.

Purpose: To verify that the device 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. W*~~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. *If the IUT claims Protocol_Revision 24 or greater heartbeat interval is the Network Port object, SC_Heartbeat_Timeout property.*

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) DO {
2.          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'41'}), -- Secure Path
                'BACnet NPDU' =
                    ReadProperty-Request,
                    'Object Identifier' =       (the IUT's Device object),
                    'Property Identifier' =     Object_Name
3.        RECEIVE Encapsulated-NPDU,
                'Message ID' =                      M1,
                -- '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 an heartbeat interval, only ½ of that interval is now given for the IUT to~~
~~-- generate a Heartbeat-Request~~
4.  BEFORE ½ of IUT's heartbeat interval + 10s
5.        RECEIVE Heartbeat-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
6.  TRANSMIT Heartbeat-ACK,
                'Message ID' =                      M2,
                -- 'Originating Virtual Address' absent
                -- 'Destination Virtual Address' absent
                'Destination Options' =             (absent or any valid value),
                -- 'Data Options' absent
```

### 12.5.1.1.17 Configurable Reconnect Timeout Test

Reason for Change: modified per Addendum 135-2020cc-1.

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*, minRT*, and no longer than *maxRT*~~600~~ seconds before attempting to reconnect. *If the IUT claims Protocol_Revision 23 or lower, minRT is a configurable parameter within the IUT and maxRT is fixed at 600 seconds. If the IUT claims Protocol_Revision 24 or greater, minRT is the Network Port object, SC_Minimum_Reconnect_Time property and maxRT is the SC_Maximum_Reconnect_Time property.*

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, *minRT*~~RT~~, within the range supported by the IUT and within 2 .. 300 seconds *and maxRT, within the range supported by the IUT and within 2 .. 600 seconds*. The IUT starts the test *disconnected from the TD*~~powered off~~. ~~If the IUT has a fixed reconnect timeout, this test shall be skipped.~~

Test Steps:

1. *MAKE(place the TD in a mode where it will not accept a websocket connection*~~incoming connections~~*)*
2. MAKE(the IUT *attempt to* connect to the TD)
3. *T1 = Local Time*

```
~~2.  CHECK(that the IUT attempts to open a new WebSocket with the TD)~~
4.  MAKE(place the TD in a mode where it will accept incoming connections)
~~4.  WAIT minRTRT seconds~~
5.  BEFORE 600~~ - RT~~ seconds
6.      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)
7.  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)
8.  T2 = Local Time
9.  CHECK (T2 - T1 >= minRT)
10. CHECK (T2 - T1 <= maxRT)
```

### 12.5.1.1.X1 Node Heartbeat-Request Execution Test

Reference: Addendum cc Clause AB.5.3.1.

Purpose: To verify that a node device accepts and responds to Heartbeat-Requests.

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

Configuration Requirements: The IUT is configured as a node and connected to the TD.

Test Steps:

```
1.  MAKE(the TD generate a Heartbeat-Request)
2.  RECEIVE PORT (TD-IUT hub WebSocket),
        Heartbeat-ACK,
        '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
```

### 12.5.1.1.X2 Must Understand Header Marker for Unknown Data Options

Reason for Change: No test exists for this functionality.

Purpose: Test if a node correctly processes a message that includes unknown data Data Options with and without 'Must Understand' Header Marker.

Test Concept: With the IUT connected to the BACnet/SC network, another node on the network (D3) sends a ReadProperty Request with a set of header options marked as 'Must Understand' = 1 and a Data Option between 2 and 30 in addition to the 'Secure Path' Data Option. Verify that the IUT does not process the message. Repeat with an additional ReadProperty Request with the same Data Options but 'Must Understand' = 0. Verify that this message was processed.

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

Test Steps:

```
1.  TRANSMIT Encapsulated-NPDU,
        'Message ID' = (M1: any valid value),
        'Originating Virtual Address' = (D3's VMAC),
        -- 'Destination Virtual Address' absent
        'Destination Options' (absent or any valid value),
        'Data Options' = ({X'C1'}, -- (more, M.U., no len, opt_type = 1 (Secure Path))
                          {X'5E'}),     -- (not more, M.U., no len, opt_type = 30 (unknown header option type))
        'BACnet NPDU' = ReadProperty-Request,
            'Object Identifier' = (the IUT's Device object),
            'Property Identifier' = Object_Name
2.  CHECK(that the IUT does not respond with ReadProperty-ACK)
3.  TRANSMIT Encapsulated-NPDU,
        'Message ID' = (M2: any valid value),
        'Originating Virtual Address' = (D3's VMAC),
        -- 'Destination Virtual Address' absent
        'Destination Options' (absent or any valid value),
        'Data Options' = ({X'C1'}, -- (more, M.U., no len, opt_type = 1 (Secure Path))
                          {X'1E'}), -- (not more, not M.U., no len, opt_type = 30 (unknown header option type))
        'BACnet NPDU' = ReadProperty-Request,
            'Object Identifier' = (the IUT's Device object),
            'Property Identifier' = Object_Name
4.  RECEIVE Encapsulated-NPDU,
        'Message ID' = (M3: any valid value),
        -- 'Originating Virtual Address' absent
        'Destination Virtual Address' = (D3'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)
```

### 12.5.1.2 Basic Node Negative Tests

### 12.5.1.2.2 Malformed BVLC Test for Nodes without Hub Function

Reason for Change: Changed test to be exclusively for devices without an active Hub Function. Also corrected/removed some of the negative tests.

Purpose: Verify the device NAKs malformed / unknown unicast BVLC and ignores malformed / unknown broadcast BVLC *when operating as a Node without an active hub function*.

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 *as required* ~~with an appropriate NAK~~ to each *message* ~~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~~. *The TD is the active hub for the BACnet/SC network.*

Test Steps:

-- ~~Invalid~~*Unknown* BVLC function
1. TRANSMIT
    'BVLC Function' = (IV: an ~~invalid~~*unknown* 1-octet value),
    'Message ID' = (M1: any valid value),
    *'Originating Virtual Address' = D3,*
    ~~-- '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' = D3*
    ~~-- 'Destination Virtual Address' absent~~
    'Destination Options' = (absent or a valid list of options),
    -- 'Data Options' absent
    'Result for BVLC Function' = IV,-- the supplied ~~invalid~~*unknown* 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, *-- error condition, required to be absent*
    -- 'Destination Virtual Address' absent
    'Destination Options' = (absent or a valid list of options),
    -- 'Data Options' absent
5. IF (receive a message) THEN
6. RECEIVE BVLC-Result,
        'Message ID' = M2,
        -- 'Originating Virtual Address' absent
        -- If this message is interpreted as an invalid Disconnect-Request from D3, the DVA = D3.
        -- If this message is interpreted as an invalid OVA, the DVA is be absent.
        'Destination Virtual Address' = ~~D3~~*(D3 or 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,~~
            *INCONSISTENT_PARAMETERS,*

```
                                                PARAMETER_OUT_OF_RANGE or OTHER)
7.  CHECK(that the IUT did not process the request)

-- Inclusion of a 'Destination Virtual Address when it is required to be absent
8.  TRANSMIT Disconnect-Request,
        'Message ID' =                  (M3: any valid value),
        -- 'Originating Virtual Address' absent,
        'Destination Virtual Address' = *(any non-broadcast VMAC), -- error condition, required to be absent* ~~(IUT's VMAC)~~,
        'Destination Options' =         (absent or a valid list of options),
        -- 'Data Options' absent
9.  IF (receive a message) THEN
10      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~~
                                                *INCONSISTENT_PARAMETERS*,
                                                PARAMETER_OUT_OF_RANGE or OTHER)
11. CHECK(that the IUT did not process the request)

-- A truncated message
12. 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 Virtual Address' absent*
        -- 'Destination Options' absent
        -- 'Data Options' absent
        -- no NPDU included in the message
13. RECEIVE BVLC-Result,
        'Message ID' =                  M4,
        -- 'Originating Virtual Address' absent
        'Destination Virtual Address' = ~~OVA~~*: D3*,
        '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
14. CHECK(that the IUT did not process the request)

-- A message with extra octets added on
15. 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'})

16. 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
17. 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,~~
~~*'Originating Virtual Address'= (D3)*~~
~~'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)~~

**12.5.1.2.3 Discard BVLC with Wrong Address Test**

Reason for Change: Delete this test from 135.1-2023.

~~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)~~

### 12.5.1.2.5 Connect-Request Response Wait Time Test

Reason for Change: modified per Addendum 135-2020cc-1.

Purpose: To verify that the device will close the WebSocket if a response to a Connect-Request is not received before the connection wait timer expires. *If the IUT claims Protocol_Revision 24 or greater connect wait timeout is the Network Port object, SC_Connect_Wait_Timeout property.*

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)
```

### 12.5.1.2.X1 Node Heartbeat-Request Initialization Failure Test

Reference: Addendum cc Clause AB.6.3.

Purpose: To verify that a Node will disconnect if a Heartbeat-ACK is not received.

Test Concept: With the IUT connected to the TD as the primary hub, allow the IUT to connect to the TD. OD sends a ReadProperty request to the IUT every HB / 2 seconds. HB is the value of the IUTs SC_Heartbeat_Timeout property. Stop sending messages to the IUT. Wait HB plus 10 seconds and verify the IUT sends a Heartbeat-Request, times out waiting for a Heartbeat-ACK and then the IUT sends a Disconnect-Request.

Configuration Requirements: Configure the SC_Heartbeat_Timeout property of the TD to be 2 times HB. Place the TD in a mode where it will not respond to Heartbeat-Requests.

Test Steps:

```
1.  REPEAT N = (1..Z) {
2.      TRANSMIT Encapsulated-NPDU,
            'Message ID' =                  (M: any valid value),
            'Originating Virtual Address' =     (OD'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
3.          RECEIVE Encapsulated-NPDU,
            'Message ID' =                    M,
            -- 'Originating Virtual Address' absent
            'Destination Virtual Address' =       (OD'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)
4.          WAIT HB / 2
    }
-- Since we already waited ½ of HB, only HB / 2 of that interval is now given for the IUT to
-- generate a Heartbeat-Request
5.      BEFORE HB / 2 + 10s
            RECEIVE Heartbeat-Request,
            'Message ID' =                    (M: any valid value)
            -- 'Originating Virtual Address' absent
            -- 'Destination Virtual Address' absent
            'Destination Options' =               (absent or any valid value),
            -- 'Data Options' absent
6.      BEFORE (2 seconds or Vendor specified timeout)
7.          RECEIVE Disconnect-Request,
            'Message ID' =                    (M: any valid value)
            -- 'Originating Virtual Address' absent
            -- 'Destination Virtual Address' absent
            'Destination Options' =               (absent or any valid value),
            -- 'Data Options' absent
```

### 12.5.1.3 Basic Node Configuration Tests

#### 12.5.1.3.1 Configuration Via PEM Test

Reason for Change: Certificate Authority is specified as the TD.

Purpose: To verify that the IUT's configuration tool supports PEM format certificates.

Test Concept: The IUT's configuration tool is made to export a certificate signing request in PEM format. The PEM signing request is imported into the *Certificate Authority (CA)* ~~TD~~ and *an operational* ~~a PEM format~~ certificate is exported in PEM format *along with the PEM formatted issuer certificate*. The *operational* ~~PEM~~ certificate is then loaded into the IUT with the IUT's configuration tool. *If the IUT requires the issuer certificate it shall also be loaded into the IUT with the IUT's configuration tool.* The IUT is then configured to connect to the TD as the primary hub. The IUT is allowed to connect to the primary hub, and a successful connection is verified.

Test Steps:

1. MAKE(the IUT's configuration tool export a certificate signing request in PEM format)
2. CHECK(that the PEM file is well formed)
3. MAKE(import the PEM file into the *CA*~~TD~~ and generate a PEM format*ted operational* certificate)
4. *MAKE(the CA generate a PEM formatted issuer certificate)*

5. MAKE(the IUT's configuration tool load the PEM format certificate into the IUT)
*6. IF (the IUT requires the issuer certifcate) THEN*
*7 MAKE(the IUT's configuration tool load the issuer certificate into the IUT)*
8. MAKE(the IUT connect to the TD using the new certificate)

### 12.5.2 Hub Tests

### 12.5.2.1 Hub Positive Tests

### 12.5.2.1.2 Local Broadcast Execution Test

Reason for Change: Allow for an invalid Destination Virtual Address. Disallowed for an absent DVA for a broadcast.

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' = (~~absent or X'FFFFFFFF'~~X'FFFFFFFFFFFF', -- the local broadcast VMAC)
        -- 'Destination Options' absent
        'Data Options' =   ({X'41'}), -- Secure Path
        'Payload'
            Who-Is-Request
2.  REPEAT Dx = (D2, D3) DO {
        RECEIVE PORT (Dx-IUT hub WebSocket),
            Encapsulated-NPDU,
            'Originating Virtual Address' = (D4's VMAC),
            'Destination Virtual Address' = ~~X'FFFFFFFF'~~ X'FFFFFFFFFFFF',       -- 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' = (absent ~~D4's VMAC~~ or ~~X'FFFFFFFF'~~ X'FFFFFFFFFFFF', 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)

### 12.5.2.1.3 Minimum NPDU Forwarding Size Test

Reason for Change: Incorrect Minimum NPDU specified.

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 *and D4*~~, 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~~
*X'3F*
*X'105C - Header Data length = 4188 octets*
*2 octets - V1, any vendor identifier <> IUT Vendor Identifier*
*1 octet - OT1, any proprietary option type*
*4185 octets - Payload, 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~~
*X'3F*
*X'105C - Header Data length = 4188 octets*
*2 octets - V1*
*1 octet - OT1*
*4185 octets - Payload*
*}),*

WriteProperty-Request,
'Object-Identifier' = O,
'Property-Identifier' = P,
'Property Value' = V

### 12.5.2.1.9 Duplicate Connection Test

Reason for Change: The test mandates a specific order of connect and disconnect requests when duplicate connection requests occur. See CRR-0528.

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. *While maintaining this first connection*~~When the connection is complete~~, *make* D3 *establish* ~~attempts to bring up~~ a second connection to the IUTs hub URI *with the same VMAC*. Verify that the IUT accepts the second connect request and closes the first connection. *Then, while maintaining the second connection, make D3 establish another new connection (third)*~~Repeat the reconnection, but~~ with a new VMAC for *D3;*~~D3 and~~ ensure that the new *connect-*request is accepted and the ~~existing~~ *second connection is closed*~~one dropped~~.

*Notes to Tester: For Steps 6, 7, and 8 and Steps 12, 13, and 14, the order in which the IUT transitions from the existing connection to the new connection is not significant.*

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~~*D3* 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)
```

**12.5.2.1.X1 SC_Hub_Function_Enable Property Test**

Reference: Addendum cc Clause 12.56.Y14.

Purpose: To ensure the IUTs hub function can be enabled and disabled using the SC_Hub_Function_Enable property.

Test Concept: With the IUTs SC_Hub_Function_Enable property set to TRUE, verify the IUT is operating as a hub. Change the IUTs SC_Hub_Function_Enable property to FALSE and verify the IUT is no longer operating as a hub.

Configuration Requirements: The IUT is configured as the primary hub and the value of the SC_Hub_Function_Enable property to TRUE. The TD is configured as the failover hub. The TDs primary hub URI is configured to reference the IUT, and the IUTs failover hub URI is configured to reference the TD.

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)
4.   IF (SC_Hub_Function_Enable is writable) THEN
5.        WRITE SC_Hub_Function_Enable = FALSE
6.        TRANSMIT ReinitializeDevice-Request
               'Reinitialized State of Device' = WARMSTART | ACTIVATE_CHANGES
               'Password' = (any valid password)
7.        RECEIVE BACnet-SimpleACK-PDU
     ELSE
```

```
8.          MAKE (SC_Hub_Function_Enable = FALSE)
9.    WAIT Activate Changes Fail Time
10.   CHECK(that the TD attempts and fails to open a WebSocket to the IUT)
11.   CHECK(that the IUT opens a WebSocket with the TD)
12.   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)
13.   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)
14.   VERIFY (SC_Hub_Function_Enable = FALSE)
15.   IF (SC_Hub_Function_Enable is writable) THEN
16.         WRITE SC_Hub_Function_Enable = TRUE
17.         TRANSMIT ReinitializeDevice-Request
                  'Reinitialized State of Device' = WARMSTART | ACTIVATE_CHANGES
                  'Password' = (any valid password)
18.         RECEIVE BACnet-SimpleACK-PDU
      ELSE
19.         MAKE (SC_Hub_Function_Enable = TRUE)
20.   WAIT Activate Changes Fail Time
21.   MAKE(the TD open a WebSocket to the IUT's hub function)
22.   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)
23.   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)
```

### 12.5.2.1.X2 Must Understand for Unknown Data Options Local Broadcast Execution Test

Purpose: To verify that IUT, as a hub, correctly accepts and distributes broadcast messages that contain an unknown Header Type.

Test Concept: With the IUT operating as a hub, D4 sends a broadcast message with unknown header type and Must Understand = TRUE to the hub. Verify that the message is forwarded to all hub connectors except D4. D4 sends a broadcast message with unknown header type and Must Understand = FALSE to the hub. In both cases, verify the messages are forwarded to all hub connectors except D4.

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:

```
-- Must Understand = 1
1.  TRANSMIT PORT (D4-IUT hub WebSocket),
        Encapsulated-NPDU,
        -- 'Originating Virtual Address' absent
        'Destination Virtual Address' =     X'FFFFFFFFFFFF', -- the local broadcast VMAC
        -- 'Destination Options' absent
        'Data Options' = ({X'C1'}, -- (more, M.U., no len, opt_type = 1 (Secure Path))
                        {X'5E'}), -- (not more, M.U., no len, opt_type = 30 (unknown header option
type)),
        'Payload'
            Who-Is-Request
2.  RECEIVE PORT (D3-IUT hub WebSocket),
        Encapsulated-NPDU,
        'Originating Virtual Address' =     (D4's VMAC),
        'Destination Virtual Address' =     X'FFFFFFFFFFFF',    -- the local broadcast VMAC
        -- 'Destination Options' absent
        'Data Options' = ({X'C1'}, -- (more, M.U., no len, opt_type = 1 (Secure Path))
                        {X'5E'}), -- (not more, M.U., no len, opt_type = 30 (unknown header option
type)),
        'Payload'
            Who-Is-Request
3.  CHECK (that the IUT does not generate an I-Am for its local node)

-- Must Understand = 0
4.  TRANSMIT PORT (D4-IUT hub WebSocket),
        Encapsulated-NPDU,
        -- 'Originating Virtual Address' absent
        'Destination Virtual Address' =     X'FFFFFFFFFFFF', -- the local broadcast VMAC
        -- 'Destination Options' absent
        'Data Options' = ({X'C1'}, -- (more, M.U., no len, opt_type = 1 (Secure Path))
                        {X'1E'}), -- (not more, no M.U., no len, opt_type = 30 (unknown header option
type)),
        'Payload'
            Who-Is-Request
5.  RECEIVE PORT (D3-IUT hub WebSocket),
        Encapsulated-NPDU,
        'Originating Virtual Address' =     (D4's VMAC),
        'Destination Virtual Address' =     X'FFFFFFFFFFFF',    -- the local broadcast VMAC
```

```
            -- 'Destination Options' absent
            'Data Options' = ({X'C1'}, -- (more, M.U., no len, opt_type = 1 (Secure Path))
                                    {X'1E'}), -- (not more, no M.U., no len, opt_type = 30 (unknown header option
type)),
            'Payload'
                  Who-Is-Request
-- D3 will initiate an I-Am on D3-IUT WebSocket
6.    RECEIVE PORT (D4-IUT hub WebSocket),
            Encapsulated-NPDU,
            'Originating Virtual Address' =       (IUT's VMAC)
            'Destination Virtual Address' =       (absent or X'FFFFFFFFFFFF', the local broadcast VMAC)
            -- 'Destination Options' absent
            'Data Options' = (Secure Path plus 0 or more valid data options)
            '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)
-- IUT will forward the I-Am from D3-IUT WebSocket to D4-IUT Websocket
```

**12.5.2.1.X3 Must Understand for Unknown Data Options Forwards Unicast BVLCs Test**

Purpose: To verify that a hub correctly forwards unicast BVLCs received that contain an unknown Header Type.

Test Concept: The IUT is operating as the primary hub. D3 sends a unicast message with unknown header type and Must Understand = TRUE to D4 via the hub. Verify that the IUT correctly forwards the message to D4. D3 sends a unicast message with unknown header type and Must Understand = FALSE 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
-- Must Understand = 1
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'C1'}, -- (more, M.U., no len, opt_type = 1 (Secure Path))
                                    {X'5E'}), -- (not more, M.U., no len, opt_type = 30 (unknown header option
type)),
            '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'C1'}, -- (more, M.U., no len, opt_type = 1 (Secure Path))
                        {X'5E'}), -- (not more, M.U., no len, opt_type = 30 (unknown header option
type)),
        'Payload' =
            Who-Is-Request
-- Must Understand = 0
3.  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'C1'}, -- (more, M.U., no len, opt_type = 1 (Secure Path))
                        {X'1E'}), -- (not more, no M.U., no len, opt_type = 30 (unknown header option
type)),
        'Payload' =
            Who-Is-Request
4.  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'C1'}, -- (more, M.U., no len, opt_type = 1 (Secure Path))
                        {X'1E'}), -- (not more, no M.U., no len, opt_type = 30 (unknown header option
type)),
        'Payload' =
            Who-Is-Request
```

### 12.5.2.2 Hub Negative Tests

### 12.5.2.2.2 Connect-Request Wait Time Test

Reason for Change: modified per Addendum 135-2020cc-1.

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. *If the IUT claims Protocol_Revision 24 or greater connect wait timeout is the Network Port object, SC_Connect_Wait_Timeout property*.

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)

### 12.5.2.2.X Malformed BVLC Test for Nodes operating as a Hub

Reason for Change: Test broken out from original test 12.5.1.2.2.

Purpose: Verify the device NAKs malformed / unknown unicast BVLCs and ignores malformed / unknown broadcast BVLCs when operating as a Node with an active hub function.

Test Concept: With the IUT connected to the BACnet/SC network and operating as the active hub, send a sequence of malformed unicast and broadcast BVLCs to the IUT. Verify that the IUT responds as required to each message and does not process nor route the messages.

Configuration Requirements: The IUT is connected to the BACnet/SC network and is operating as the active hub. The TD is connected to the BACnet/SC network as a node.

Test Steps:

```
-- Unknown BVLC function
1.  TRANSMIT
        'BVLC Function' =               (IV: an unknown 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 unknown 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, -- error condition, required to be absent
        -- 'Destination Virtual Address' absent
        'Destination Options' =             (absent or a valid list of options),
        -- 'Data Options' absent
5.  IF (receive a message) THEN
6.      RECEIVE BVLC-Result,
            'Message ID' =                  M2,
            -- 'Originating Virtual Address' absent
            -- A valid message from a hub requires DVA to be 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' =                  (HEADER_ENCODING_ERROR,
                                                INCONSISTENT_PARAMETERS,
                                                PARAMETER_OUT_OF_RANGE or OTHER)
7.  CHECK(that the IUT did not process the request)

-- Inclusion of a 'Destination Virtual Address when it is required to be absent
8.  TRANSMIT Disconnect-Request,
        'Message ID' =                      (M3: any valid value),
        -- 'Originating Virtual Address' absent,
        'Destination Virtual Address' =     (IUT's VMAC), -- error condition, required to be absent
        'Destination Options' =             (absent or a valid list of options),
        -- 'Data Options' absent
9.  IF (receive a message) THEN
10.     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),
            'Error Code' =                  (HEADER_ENCODING_ERROR,
                                                INCONSISTENT_PARAMETERS,
                                                PARAMETER_OUT_OF_RANGE or OTHER)
11. CHECK(that the IUT did not process the request)

-- A truncated message
12. TRANSMIT Encapsulated-NPDU,
        'Message ID' =                  (M4: any valid value),
        --'Originating Virtual Address' absent
        -- 'Destination Virtual Address' absent,
        -- 'Destination Options' absent
        -- 'Data Options' absent
        -- no NPDU included in the message
13. RECEIVE BVLC-Result,
        'Message ID' =                  M4,
        -- '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'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
14. CHECK(that the IUT did not process the request)

-- A message with extra octets added on
15. 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'})
16. 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
17.  CHECK(that the IUT did not process the request)

-- A truncated broadcast message
18.  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)
                                           })
19.  CHECK(that the IUT does not send a BVLC-Result, did not process the message, nor route the
message)
```

**12.5.2.2.X1 Hub Heartbeat-Request Initialization Failure Test**

Reference: Addendum cc Clause AB.5.3.1.

Purpose: To verify that a Hub initiates a Heartbeat-Request before attempting to terminate a connection.

Test Concept: With the IUT operating as a hub, the TD connects to the IUT. The TD sends a ReadProperty request to the IUT every HB / 2 seconds. HB is the value of the IUTs SC_Heartbeat_Timeout property. Stop sending messages to the IUT. Wait HB plus 10 seconds and verify the IUT sends a Heartbeat-Request, times out waiting for a Heartbeat-ACK and then the IUT sends a Disconnect-Request.

Configuration Requirements: Configure the SC_Heartbeat_Timeout property of the TD to be 2 times HB. Place the TD in a mode where it will not respond to Heartbeat-Requests.

Test Steps:

```
1.   REPEAT N = (1..Z) {
2.        TRANSMIT Encapsulated-NPDU,
             'Message ID' =                     (M: any valid value),
             -- 'Originating Virtual Address' absent
             'Destination Virtual Address' =         (IUT's VMAC),
             '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
3.        RECEIVE Encapsulated-NPDU,
             'Message ID' =                     M,
             'Originating Virtual Address' =         (IUT's VMAC),
             -- 'Destination Virtual Address' 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)
4.      WAIT HB / 2
    }
-- Since we already waited ½ of HB, only HB / 2 of that interval is now given for the IUT to
-- generate a Heartbeat-Request
5.  BEFORE HB / 2 + 10s
6.      RECEIVE Heartbeat-Request,
            'Message ID' =                    (M: any valid value)
            -- 'Originating Virtual Address' absent
            -- 'Destination Virtual Address' absent
            'Destination Options' =               (absent or any valid value),
            -- 'Data Options' absent
7.  BEFORE (2 seconds or Vendor specified timeout)
8.      RECEIVE Disconnect-Request,
            'Message ID' =                    (M: any valid value)
            -- 'Originating Virtual Address' absent
            -- 'Destination Virtual Address' absent
            'Destination Options' =               (absent or any valid value),
            -- 'Data Options' absent
```

### 12.5.3 Direct Connect Tests

### 12.5.3.1 Direct~~ion~~ Connect Basic Tests

### 12.5.3.1.1 Direct~~ion~~ Connect Basic Positive Tests

### 12.5.3.1.2 Direct~~ion~~ Connect Basic Negative Tests

### 12.5.3.1.2.X Malformed BVLC Test for Nodes with a Direct Connection

Reason for Changes: Test broken out from original 12.5.1.2.2.

Purpose: Verify the device NAKs malformed / unknown unicast BVLC and ignores malformed / unknown broadcast BVLC when it has a direct connection with the TD.

Test Concept: With the IUT connected to the BACnet/SC network, the TD sends a sequence of malformed unicast and broadcast BVLCs to the IUT. Verify that the IUT responds as required to each message and does not process nor route the messages.

Configuration Requirements: The IUT has a direct connection established with the TD.

Test Steps:

```
-- Unknown BVLC function
1.  TRANSMIT
        'BVLC Function' =                 (IV: an unknown 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 unknown 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' =         TD's VMAC, -- error condition, required to be absent
        -- 'Destination Virtual Address' absent
        'Destination Options' =                 (absent or a valid list of options),
        -- 'Data Options' absent
5.  IF (receive a message) THEN
6.      RECEIVE BVLC-Result,
            'Message ID' =                      M2,
            -- 'Originating Virtual Address' absent
            -- A valid message with Direct Connect requires DVA to be 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' =                      (HEADER_ENCODING_ERROR,
                                                    INCONSISTENT_PARAMETERS,
                                                    PARAMETER_OUT_OF_RANGE or OTHER)
7.  CHECK(that the IUT did not process the request)

-- Inclusion of a 'Destination Virtual Address when it is required to be absent
8.  TRANSMIT Disconnect-Request,
        'Message ID' =                      (M3: any valid value),
        -- 'Originating Virtual Address' absent,
        'Destination Virtual Address' =     (IUT's VMAC), -- error condition, required to be absent
        'Destination Options' =             (absent or a valid list of options),
        -- 'Data Options' absent
9.  IF (receive a message) THEN
10.     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),
            'Error Code' =                      (HEADER_ENCODING_ERROR,
                                                    INCONSISTENT_PARAMETERS,
                                                    PARAMETER_OUT_OF_RANGE or OTHER)
11. CHECK(that the IUT did not process the request)

-- A truncated message
12. TRANSMIT Encapsulated-NPDU,
        'Message ID' =                      (M4: any valid value),
        -- 'Originating Virtual Address' absent
        -- 'Destination Virtual Address' absent
        -- 'Destination Options' absent
        'Data Options' =                    ({X'41'}), -- Secure Path
        -- no NPDU included in the message
13. RECEIVE BVLC-Result,
        'Message ID' =                      M4,
        -- 'Originating Virtual Address' absent
        -- 'Destination Virtual Address' absent
        -- '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
14. CHECK(that the IUT did not process the request)

-- A message with extra octets added on
15. 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'})
16. 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
17. CHECK(that the IUT did not process the request)
```

**12.5.3.2 Accepting Direct Connect Tests**

**12.5.3.2.2 Accepting Direct Connect Negative Tests**

**12.5.3.2.2.1 Connect-Request Wait Time Test**

Reason for Change: Updated to include SC_Connect_Wait_Timeout property if available.

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. *If the IUT claims Protocol_Revision 24 or greater connection wait timer is the Network Port object, SC_Connect_Wait_Timeout property.*

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)

### 12.5.3.3 Initiating Direct Connection Tests

#### 12.5.3.3.1 Initiating Direct Connect Positive Tests

##### 12.5.3.3.1.1 Direct Connect Establishment Test

*Reason for change: The original test incorrectly expected the creation of a Direct Connection before the peer's direct-connect URI is discovered.*

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(*an action that will cause the* IUT *to initiate* ~~establish~~ a direct connection to D3)

```
2.  IF (the IUT supports discovering direct connection URIs) {
3.      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
4.      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)
    }
5.  CHECK(that the IUT opens a WebSocket to D3's WebSocket-URI)
6.  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)
7.  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)
```

### 12.5.X.1 Verify Time Synchronization Test

Reason for Change: No test for this functionality.

Purpose: To maintain IUT time accuracy, verify the IUT supports time synchronization by some means other than either TimeSynchronization or UTCTimeSynchronization services.

Test Concept: Read the Device object's (D1) Local_Date and Local_Time. Make the IUT time change. Verify the Local_Date and Local_Time change to the new time. Power cycle the IUT and validate Local_Date and Local_Time.

Configuration Requirements: None.

Test Steps:

```
1.  READ time1 = D1, Local_Time
2.  READ date1 = D1, Local_Date
3.  MAKE(IUT time change by some value t)
4.  READ time2 = D1, Local_Time
5.  READ date2 = D1, Local_Date
6.  VERIFY time2 ~= time1 + t
7.  VERIFY date2 = date1 + t
8.  MAKE (the IUT power cycle)
9.  VERIFY D1, Local_Time ~= time2 + the time the IUT requires to reboot
10. VERIFY D1, Local_Date = date2
```

**135.1-2023*u*-7 Update Backup and Restore Execution Test**

Rationale

Errors have been identified in the Backup and Restore execution test.

## 13 Special Functionality Tests

### 13.8 Backup and Restore Procedure Tests

### 13.8.1 Backup and Restore Execution Tests

### 13.8.1.1 Execution of Full Backup and Restore Procedure

Reason for Change: Fixed to check if a record exists before attempting to write the record. Updated test steps and added additional brackets for clarity.

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 {
5.      READ NAMES[X] = (OL1[X]), Object_Name
    }
6.  IF (Protocol_Revision is present AND Protocol_Revision >= 10) THEN {
7.      IF (Backup_Preparation_Time is present) THEN
8.          READ BPT = Backup_Preparation_Time
        ELSE
9.          READ BPT = APDU_Timeout
10.     IF (Restore_Preparation_Time is present) THEN
11.         READ RPT = Restore_Preparation_Time
        ELSE
12.         READ RPT = APDU_Timeout
13.     IF (Restore_Completion_Time is present) THEN
14.         READ RCT = Restore_Completion_Time
        ELSE
15.         READ RCT = APDU_Timeout
```

```
16.     IF (Backup_And_Restore_State is present or Protocol_Revision >= 13) THEN
17.         VERIFY Backup_And_Restore_State = IDLE
    }
18. TRANSMIT ReinitializeDevice-Request,
        'Reinitialized State of Device' =    STARTBACKUP,
        'Password' = (any valid password)
19. RECEIVE BACnet-SimpleACK-PDU
20. IF (Protocol_Revision is present AND Protocol_Revision >= 10) THEN {
21.     WAIT BPT
22.     IF (Backup_And_Restore_State is present or Protocol_Revision >= 13) THEN {
23.         READ BRSTATE = Backup_And_Restore_State
24.         READ CF = Configuration_Files
25.         WHILE (BRSTATE = PREPARING_FOR_BACKUP) DO {
26.             WAIT 1 second
27.             READ BRSTATE = Backup_And_Restore_State
28.             IF CF is an empty list THEN
29.                 READ CF = Configuration_Files
30.             IF CF is a non-empty list THEN
31.                 READ X = (the file referenced by Configuration_Files[1]).Name
            }
32.         CHECK (BRSTATE = PERFORMING_A_BACKUP)
        }
    }
33. READ CF = Configuration_Files
34. CHECK (CF is a non-empty array of BACnetObjectIdentifiers referring to File objects)
35. REPEAT X = (each entry in CF) DO {
36.     READ Y = X, File_Access_Method
37.     IF (Y = RECORD_ACCESS) THEN {
38.         WHILE (the last read resulted in an Ack with 'End Of File' == FALSE) DO {
39.             TRANSMIT AtomicReadFile-Request,
                    'Object Identifier' = X,
                    'File Start Record' = (the next unread record),
                    'Requested Record Count' = 1
40.             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 {
41.         WHILE (the last read did not indicate 'End Of File') DO {
42.             TRANSMIT AtomicReadFile-Request,
                    'Object Identifier' = X,
                    'File Start Position' = (the next unread octet),
                    'Requested Octet Count' = MROC
43.             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 if 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
            }
        }
    }
44. TRANSMIT ReinitializeDevice-Request,
        'Reinitialize State Of Device' = ENDBACKUP,
        'Password' = (any valid password)
45. RECEIVE BACnet-SimpleACK-PDU
46. VERIFY System_Status ! = BACKUP_IN_PROGRESS
47. IF (Backup_And_Restore_State is present or (Protocol_Revision is present and Protocol_Revision >= 13)) THEN
48.     VERIFY Backup_And_Restore_State = IDLE
49. IF (Database_Revision is changeable) THEN {
50.     MAKE (the configuration in the IUT different, such that the Database_Revision property increments)
51.     VERIFY Database_Revision <> DR1
52.     READ DR2 = Database_Revision
53.     CHECK (DR1 <> DR2)
    }
54. TRANSMIT ReinitializeDevice-Request,
        'Reinitialize State Of Device' = STARTRESTORE,
        'Password' = (any valid password)
55. RECEIVE BACnet-SimpleACK-PDU
56. IF (Protocol_Revision is present AND Protocol_Revision >= 10) THEN
57.     WAIT RPT
58.     IF (Backup_And_Restore_State is present or Protocol_Revision >= 13) THEN {
59.         READ BRSTATE = Backup_And_Restore_State
60.         WHILE (BRSTATE = PREPARING_FOR_RESTORE) DO {
61.             WAIT 1 second
62.             READ BRSTATE = Backup_And_Restore_State
            }
63.         CHECK (BRSTATE = PERFORMING_A_RESTORE)
        }
64. READ OL2 = Object_List
65. REPEAT X = (entry in CF) DO {
66.     BU_FS = (the file size of X)
67.     IF (X is not in OL2) THEN {
68.         TRANSMIT CreateObject-Request
                'Object Specifier' = X
69.         RECEIVE CreateObject-ACK
                'Object Identifier' = X
        }
~~        READ FS = X, File_Size~~
70.     IF (X, File_Size <> BU_FS~~is not equal to the size of the backed up file~~) THEN
71.         WRITE X, File_Size = 0
72.     IF (Y = RECORD_ACCESS) THEN {
73.         IF (BU_FS > 0) THEN {
74.             TRANSMIT AtomicWriteFile-Request
                    'File Identifier' = X
                    'File Start Record' = 0
                    'Record Data' = (file content for first record obtained in step 11)
75.             RECEIVE AtomicWriteFile-ACK
                    'File Start Record' = 0
```

```
76.                 REPEAT REC = (each record in the backup of this file) {
77.                     TRANSMIT AtomicWriteFile-Request
                            'File Identifier' = X
                            'File Start Record' = -1
                            'Record Count' = 1
                            'Record Data' = REC
78.                     RECEIVE AtomicWriteFile-ACK
                            'File Start Record' = -1
                }
            }
        }
        ELSE {
79.         REPEAT Z = (0 through the file size, in increments of MWDL) DO {
80.             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)
81.             RECEIVE AtomicWriteFile-ACK
                    'File Start Position' = Z
            }
        }
    }
82.  IF (Backup_And_Restore_State is present or (Protocol_Revision is present and Protocol_Revision >=
13)) THEN
83.      VERIFY Backup_And_Restore_State = PERFORMING_A_RESTORE
84.  TRANSMIT ReinitializeDevice-Request,
         'Reinitialize State Of Device' = ENDRESTORE,
         'Password' = (any valid password)
85.. RECEIVE BACnet-SimpleACK-PDU
86.  IF (Protocol_Revision is present AND Protocol_Revision >= 10) THEN {
87.      WAIT RCT
88.      IF (Backup_And_Restore_State is present or Protocol_Revision >= 13) THEN
89.          VERIFY Backup_And_Restore_State = IDLE
     }
90.  READ DR3 = Database_Revision
91.  CHECK (DR3 <> DR1)
92.  IF (Database_Revision was changed in step 16) THEN
93       CHECK (DR3 <> DR2)
94.  VERIFY Last_Restore_Time > LRT1
95.  READ OL3 = Object_List
96.  CHECK (that OL1 and OL3 contain the same set of objects)
97.  REPEAT X = (1 through length of OL1) DO {
97.      VERIFY (OL1[X]), Object_Name = NAMES[X]
     }
```

## 135.1-2023*u*-8 Add BACnet/SC Certificate Replacement Tests

Rationale

135-2020 Addendum cc included procedurs to manage BACnet Secure Connect certificates.

## 13.Y BACnet/SC Certificate Replacement Tests

### 13.Y.1 Extending Operational Certificate Validity Period

Reason for Change: There is no test for this functionality.

Purpose: To verify the IUT replaces the BACnet/SC Operational certificates in all nodes and hubs in a BACnet/SC network.

Test Concept: The IUT is made to initiate an update of the operational certificates in all nodes in a BACnet/SC network. This test verifies the successful replacement of the operational certificates by ensuring each node is able to reconnect to the BACnet/SC network.

Configuration Requirements: This test assumes each device on the BACnet/SC network is active and is at Protocol_Revision 24 or greater. The signing CA must be an external entity.

Notes To Tester: Individual steps can be performed on one or more nodes. For example, the IUT is allowed to upload all the CSRs before delivering them to the signing CA.

Test Steps:

```
1.  MAKE (IUT initiate the process to replace the Operational Certificates in every node in a BACnet/SC network)
2.  REPEAT X = (all nodes in the network with the hub node done last) DO
3.      CHECK (that the IUT attempts to write to the GENERATE_CSR_FILE command)
4.      IF (the TD return an BACnet-SimpleACK) THEN
5.          CHECK (that the IUT wait for Command=IDLE)
6.      CHECK (that the IUT uploads the File object that contains the CSR)
7.      CHECK (that the IUT makes the uploaded CSR available to the signing CA and accepts the new operational certificate based on the uploaded CSR)
8.      CHECK (that the IUT downloads the new operational certificate to the File object that represents the operational certificate)
9.      CHECK (that the IUT sends a ReinitializeDevice (ACTIVATE_CHANGES or WARMSTART, <password>) message to activate the changes to the Network Port object)
10.     WAIT (Activate Changes Fail Time specified for X)
11.     CHECK (the connection to X is re-established)
    }
```

### 13.Y.2 Extending Issuer Certificate Validity Period

Reason for Change: There is no test for this functionality.

Purpose: To verify the IUT replaces the BACnet/SC Issuer certificates in all nodes and hubs in a BACnet/SC network.

Test Concept: The IUT is made to initiate a download of a new issuer certificates, replace the operational certificates and remove the old issuer certificates of a BACnet/SC network. This test verifies the successful replacement of the issuer certificates by ensuring each node is able to reconnect to the BACnet/SC network.

Configuration Requirements: This test assumes each device on the BACnet/SC network is active and is at Protocol_Revision 24 or greater. The signing CA must be an external entity.

Notes To Tester: Individual steps can be performed on one or more nodes.

Test Steps:

```
1.  MAKE (IUT initiate the Process to replace the issuer Certificates in every node in a BACnet/SC network)
------- Add a new issuer certificate to every node
2.  REPEAT X = (all nodes in the network with the hub node done last) DO {
3.      CHECK (that the IUT determines the File object that represents the unused issuer certificate)
4.      CHECK (that the IUT downloads the new issuer certificate to the File object that represents the unused issuer certificate)
5.      CHECK (that the IUT sends a ReinitializeDevice (ACTIVATE_CHANGES or WARMSTART, <password>) message to activate the changes to the Network Port object)
6.      WAIT (Activate Changes Fail Time specified for X)
7.      CHECK (the connection to X is re-established)
    }

------- Replace the operational certificate in every node in the network
8.  REPEAT X = (all nodes in the network with the hub node done last) DO {
9.      CHECK (that the IUT attempts to writes the GENERATE_CSR_FILE command)
10.     IF (the TD returns a BACnet-SimpleACK) THEN
11.         CHECK (that the IUT waits for Command=IDLE)
12.     CHECK (that the IUT uploads the File object that contains the CSR)
13.     CHECK (that the IUT makes the uploaded CSR available to the signing CA and accepts the new operational certificate based on the uploaded CSR)
14.     CHECK (that the IUT downloads the new operational certificate to the File object that represents the operational certificate)
15.     CHECK (that the IUT sends a ReinitializeDevice (ACTIVATE_CHANGES or WARMSTART, <password>) message to activate the changes to the Network Port object)
16.     WAIT (Activate Changes Fail Time specified for X)
17.     CHECK (the connection to X is re-established)
    }

------- remove the old issuer certificate from every node in the network
18. REPEAT X = (all nodes in the network with the hub node done last) DO {
19.     CHECK (that the IUT determines the File object that represents the unused issuer certificate)
20.     CHECK (that the IUT deletes the content of the issuer certificate File object that represents the unused issuer certificate)
21.     CHECK (that the IUT sends a ReinitializeDevice (ACTIVATE_CHANGES or WARMSTART, <password>) message to activate the changes to the Network Port object)
22.     WAIT (Activate Changes Fail Time specified for X)
23.     CHECK (the connection to X is re-established)
    }
```

### 13.Y.3 Banning One or More Nodes from the Network

Reason for Change: There is no test for this functionality.

Purpose: This test verifies that the IUT can remove a node from a BACnet/SC network.

Test Concept: The IUT is made to initiate a download of new issuer certificates, replace the operational certificates, and remove the old issuer certificates of a BACnet/SC network while skipping the nodes to be removed. This test verifies that the IUT can remove certain nodes from a BACnet/SC network by ensuring that only the chosen nodes can reconnect to the BACnet/SC network and that the nodes designated for removal are no longer part of the network.

Configuration Requirements: This test assumes each device on the BACnet/SC network is active and is at Protocol_Revision 24 or greater. The signing CA must be an external entity.

Notes To Tester: Individual steps can be performed on one or more nodes.

Test Steps:

```
1.  MAKE (the IUT to initiate the process to ban one or more nodes from the Network)
-------Add a new issuer certificate to each node in the network skipping the nodes to be removed
2.  REPEAT X = (all nodes in the network, except the nodes to be removed, with the hub node being done last) DO {
3.      CHECK (that the IUT determines the File object that represents the unused issuer certificate)
4.      CHECK (that the IUT downloads the new issuer certificate to the File object that represents the unused issuer certificate)
5.      CHECK (that the IUT sends a ReinitializeDevice (ACTIVATE_CHANGES or WARMSTART, <password>) message to activate the changes to the Network Port object)
6.      WAIT (Activate Changes Fail Time specified for X)
7.      CHECK (the connection to X is re-established)
    }

------- Replace the operational certificate in every node in the network skipping the nodes to be removed.
8.  REPEAT X = all nodes in the network, except the nodes to be removed, with the hub node being done last) DO {
9.      CHECK (that the IUT attempts to write to the GENERATE_CSR_FILE command)
10.     IF (the TD returns a BACnet-SimpleACK) THEN
11.         CHECK (that the IUT waits for Command=IDLE)
12.     CHECK (that the IUT uploads the File object that contains the CSR)
13.     CHECK (that the IUT makes the uploaded CSR available to the signing CA and accepts the new operational certificate based on the uploaded CSR)
14.     CHECK (that the IUT downloads the new operational certificate to the File object that represents the operational certificate)
15.     CHECK (that the IUT sends a ReinitializeDevice (ACTIVATE_CHANGES or WARMSTART, <password>) message to activate the changes to the Network Port object)
16.     WAIT (Activate Changes Fail Time specified for X)
17.     CHECK (the connection to X is re-established)
    }

------- Remove issuer certificate from every node in the network skipping the removed nodes
18.. REPEAT X = all nodes in the network, except the nodes to be removed, with the hub node being done last) DO {
19.     CHECK (that the IUT determines the File object that represents the unused issuer certificate)
20.     CHECK (that the IUT deletes the content of the issuer certificate File object that represents the unused issuer certificate)
21.     CHECK (that the IUT sends a ReinitializeDevice (ACTIVATE_CHANGES or WARMSTART, <password>) message to activate the changes to the Network Port object)
22.     WAIT (Activate Changes Fail Time specified for X)
23.     CHECK (the connection to X is re-established)
    }
```

### 13.Y.4 Adding a New Issuer Certificate to the Device

New test per Addendum 135-2020cc-3. Added File object File_Size and Modification_Date checks.

Purpose: This test verifies the IUT can execute the procedure to add a new Issuer Certificate.

Test Concept: With the IUT connected to a BACnet/SC network, this test takes the IUT through the procedure to add a new Issuer Certificate to a file object referenced in the Network Port object, NP1, of the

IUT of the active BACnet/SC network. The procedure is specified in 135-2020 Addendum cc, Clause 19.Y.3.1.

Configuration Requirements: The IUT is actively connected to a BACnet/SC network.

Notes To Tester: 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 octets less than the minimum of the TD’s Max_APDU_Length_Accepted and the IUT’s maximum transmittable APDU length. MWDL shall be 21 octets less than the minimum of the TD’s maximum transmittable APDU length and the IUT’s Max_APDU_Length_Accepted. (See 135-2020 Addendum cc, Clause 19.Y.3.1)

Test Steps:

```
1.  READ NP1, IC = (Issuer_Certificate_Files 1, Issuer_Certificate_Files 2)
2   FS = READ (IC[1], File_Size)
3.  IF (FS > 0) THEN {
4.      WHILE (the last read did not indicate 'End Of File') DO {
5.          TRANSMIT AtomicReadFile-Request,
                ‘Object Identifier’ = IC[1],
                ‘File Start Position’ = (the next unread octet),
                ‘Requested Octet Count’ = MROC
6.          RECEIVE AtomicReadFile-ACK,
                'End Of File' = TRUE | FALSE,
                ‘File Start Position’ = (the next unread octet)
                ‘File Data’ = (IC[1] File contents of length MROC if 'End Of File' is FALSE
                    or of length MROC or less if 'End Of File' is TRUE)
        }
    }
7   FS = READ (IC[2], File_Size)
8.  IF (FS > 0) THEN {
9.      WHILE (the last read did not indicate 'End Of File') DO {
10.         TRANSMIT AtomicReadFile-Request,
                ‘Object Identifier’ = IC[2],
                ‘File Start Position’ = (the next unread octet),
                ‘Requested Octet Count’ = MROC
11.         RECEIVE AtomicReadFile-ACK,
                'End Of File' = TRUE | FALSE,
                File Start Position’ = (the next unread octet)
                ‘File Data’ = (IC[2] File contents of length MROC if 'End Of File' is FALSE
                    or of length MROC or less if 'End Of File' is TRUE)
        }
    }
12. IF (IC[1] File contents is = the IUT's known current issuer certificate) THEN {
13.     WRITE IC[2], File_Size = 0
14.     VERIFY IC[2], Modification_Date = (the current local data and time)
15.     VERIFY NP1, Changes_Pending = TRUE
16.     REPEAT Z = (0 through the file size, in increments of MWDL) DO {
17.         TRANSMIT AtomicWriteFile-Request
                File Identifier’ = IC[2]
                ‘File Start Position’ = Z
                ‘Record Data’ = (file contents of the new issuer certificate, the number of octets
                        being the lesser of (file size - Z) and MWDL)
18.         RECEIVE AtomicWriteFile-ACK
                ‘File Start Position’ = Z
        }
19.       WAIT Internal Processing Fail Time
20.       VERIFY IC[2], Modification_Date = (the current local data and time)
21.       VERIFY NP1, Changes_Pending = TRUE
    }
    ELSE
    {
22.       WRITE IC[1], File_Size = 0
23.       VERIFY IC[1], Modification_Date = (the current local data and time)
24.       VERIFY NP1, Changes_Pending = TRUE
25.       REPEAT Z = (0 through the file size, in increments of MWDL) DO {
26.           TRANSMIT AtomicWriteFile-Request
                  'File Identifier' = IC[1]
                  'File Start Position' = Z
                  'Record Data' = (file contents of the new issuer certificate, the number of octets
                          being the lesser of (file size - Z) and MWDL)
27.           RECEIVE AtomicWriteFile-ACK
                  'File Start Position' = Z
          }
28.       WAIT Internal Processing Fail Time
29.       VERIFY IC[1], Modification_Date = (the current local data and time)
30.       VERIFY NP1, Changes_Pending = TRUE
    }
31. TRANSMIT ReinitializeDevice-Request
          'Reinitialized State of Device' = WARMSTART | ACTIVATE_CHANGES
          'Password' = (any valid password)
32. RECEIVE BACnet-SimpleACK-PDU
33. WAIT Activate Changes Fail Time
34. CHECK (the connection is re-established)
35. VERIFY NP1, Changes_Pending = FALSE
```

### 13.Y.5 Replace the Operational Certificate

New test per *Addendum 135-2020cc-3.* Added File object File_Size and Modification_Date checks.

Purpose: This test verifies the IUT can execute the procedure to replace the operational certificate.

Test Concept: With the IUT connected to a BACnet/SC network, this test takes the IUT through the procedure to replace the Operational Certificate to a file object referenced in the Network Port object, NP1, of the active BACnet/SC network. The procedure is specified in 135-2020 Addendum cc, Clause 19.Y.3.2.

Configuration Requirements: The IUT is actively connected to the BACnet/SC network.

Notes To Tester: 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 octets less than the minimum of the TD's Max_APDU_Length_Accepted and the IUT's maximum transmittable APDU length. MWDL shall be 21 octets less than the minimum of the TD's maximum transmittable APDU length and the IUT's Max_APDU_Length_Accepted.

Test Steps:

```
1.  IF (the IUT is capable of generating a certificate signing request file) THEN
          -- request the IUT generate a CSR, and wait for it to timeout
2.        WRITE NP1, Command = GENERATE_CSR_FILE
3.        WHILE (NP1, Command <> IDLE) DO {}
    ELSE
4.        MAKE (The Certificate_Signing_Request_File of NP1 contain a new CSR File)
```

```
5.  READ CSR = NP1, Certificate_Signing_Request_File
6.  VERIFY CSR, File_Size > 0
7.  READ OC = NP1, Operational_Certificate_File
8.  WHILE (the last read did not indicate 'End Of File') DO {
9.      TRANSMIT AtomicReadFile-Request,
            ‘Object Identifier’ = CSR,
            ‘File Start Position’ = (the next unread octet),
            ‘Requested Octet Count’ = MROC
10.     RECEIVE AtomicReadFile-ACK,
            'End Of File' = TRUE | FALSE,
            ‘File Start Position’ = (the next unread octet)
            ‘File Data’ = (CSR File contents of length MROC if 'End Of File' is FALSE
                or of length MROC or less if 'End Of File' is TRUE)
    }
11. MAKE (The signing CA generate a new operational certificate based on the CSR File)
12. WRITE OC, File_Size = 0
13. VERIFY OC, Modification_Date = (the current local data and time)
14. VERIFY NP1, Changes_Pending = TRUE
15. REPEAT Z = (0 through the file size, in increments of MWDL) DO {
16.     TRANSMIT AtomicWriteFile-Request
            ‘File Identifier’ = OC
            ‘File Start Position’ = Z
            ‘Record Data’ = (file contents of the new operational certificate,
                the number of octets being the lesser of (file size - Z) and MWDL)
17.     RECEIVE AtomicWriteFile-ACK
            ‘File Start Position’ = Z
    }
18. WAIT Internal Processing Fail Time
19. VERIFY OC, File_Size > 0
20. VERIFY OC, Modification_Date = (the current local data and time)
21. VERIFY NP1, Changes_Pending = TRUE
22. TRANSMIT ReinitializeDevice-Request
        'Reinitialized State of Device' = WARMSTART | ACTIVATE_CHANGES
        'Password' = (any valid password)
23. RECEIVE BACnet-SimpleACK-PDU
24. WAIT Activate Changes Fail Time
25. CHECK (the connection is re-established)
26. VERIFY NP1, Changes_Pending = FALSE
```

### 13.Y.6 Removing an Outdated Issuer Certificate from the Device

New test per Addendum 135-2020cc-3. Added File object File_Size and Modification_Date checks.

Purpose: This test verifies the IUT can execute the procedure to remove an Issuer Certificate.

Test Concept: With the IUT connected to a BACnet/SC network, this test takes the IUT through the procedure to remove an Issuer Certificate in the Network Port object, NP1, of the active BACnet/SC network. The procedure is specified in 135-2020 Addendum cc, Clause 19.Y.3.3.

Configuration Requirements: The IUT is actively connected to the BACnet/SC network and the NP1, Issuer_Certificate_Files property contains two Issuer Certificates.

Notes To Tester: 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 octets less than the minimum of the TD’s Max_APDU_Length_Accepted and the IUT’s maximum transmittable APDU length. MWDL shall be 21 octets less than the minimum of the TD's maximum transmittable APDU length and the IUT's Max_APDU_Length_Accepted.

Test Steps:

```
1.  READ NP1, IC =(Issuer_Certificate_Files 1, Issuer_Certificate_Files 2)
2.  WHILE (the last read did not indicate 'End Of File') DO {
3.      TRANSMIT AtomicReadFile-Request,
            'Object Identifier' = IC[1],
            'File Start Position' = (the next unread octet),
            'Requested Octet Count' = MROC
4.      RECEIVE AtomicReadFile-ACK,
            'End Of File' = TRUE | FALSE,
            'File Start Position' = (the next unread octet)
            'File Data' = (IC[1] File contents of length MROC if 'End Of File' is FALSE
                    or of length MROC or less if 'End Of File' is TRUE)
    }
5.  WHILE (the last read did not indicate 'End Of File') DO {
6.      TRANSMIT AtomicReadFile-Request,
            'Object Identifier' = IC[2],
            'File Start Position' = (the next unread octet),
            'Requested Octet Count' = MROC
7.      RECEIVE AtomicReadFile-ACK,
            'End Of File' = TRUE | FALSE,
            File Start Position' = (the next unread octet)
            'File Data' = (IC[2] File contents of length MROC if 'End Of File' is FALSE
                    or of length MROC or less if 'End Of File' is TRUE)
    }
8.  IF (IC[1] File contents is = the IUT's known current issuer certificate) THEN
9.      WRITE IC[2], File_Size = 0
10.     VERIFY IC[2], Modification_Date = (the current local data and time)
    ELSE
11.     WRITE IC[1], File_Size = 0
12.     VERIFY IC[1], Modification_Date = (the current local data and time)
13. VERIFY NP1, Changes_Pending = TRUE
14. TRANSMIT ReinitializeDevice-Request
        'Reinitialized State of Device' = WARMSTART | ACTIVATE_CHANGES
        'Password' = (any valid password)
15. RECEIVE BACnet-SimpleACK-PDU
16. WAIT Activate Changes Fail Time
17. CHECK (the connection is re-established)
18. VERIFY NP1, Changes_Pending = FALSE
```